MULTIPLE MOBILE ROBOTS

— FUZZY BEHAVIOR BASED ARCHITECTURE AND BEHAVIOR EVOLUTION

Xiao Peng

(B.Eng, M.Eng)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

JANUARY 2006

Acknowledgements

I extend my sincere gratitude and appreciation to all those people who gave me the possibility to complete this thesis. Special thanks are due to my supervisors Dr. Prahlad Vadakkepat and Prof. Lee Tong Heng, whose valuable guidance, suggestions and encouragement helped me in the last four years. Their enthusiastic, optimistic and critical attitude in the research gave a strong impetus to my scientific work. Working with them proves to be a very rewarding and pleasurable experience that will also benefit my journey of life in the future.

Many thanks go to A/Prof. Ge Shu zhi, A/Prof. Xu Jian Xin, Dr. Tan Kay Chen, Dr. Tan Woei Wan and Dr. Wang Zhu Ping for their kind help and suggestions. Furthermore, I would like to express my appreciation to Ms. Liu Xin, Mr. Chan Kit Wai and Mr. Quek Boon Kiat for many constructive and stimulating discussions with them.

I am also grateful to all the members of in the Mechatronics & Automation Laboratory, Department of Electrical & Computer Engineering, National University of Singapore, for providing the solid research facilities, as well as a pleasant, friendly and at the same time challenging environment. I do cherish all the nice time I have spent there.

Acknowledgement is extended to National University of Singapore for giving me the chance to pursuit my PhD education and to do the research work with the university facilities.

Especially, I am deeply indebted to my beloved wife Gong Xia, for her love, understanding and encouragement in all aspects of my life.

Finally, I dedicate this work to my parents for their love and support all along.

Summary

Under the category of soft computing, fuzzy logic and genetic algorithms have been extensively developed in the past several decades and successfully applied to various kinds of problems, both academic and industrial. Developments in these two fields, as well as achievements on other technologies, enable robotic systems to play an important role in our world. In this thesis, interdisciplinary research works involving the fuzzy logic control (FLC), robotic system and genetic algorithms (GAs) are presented.

The thesis comprises of two parts focused on the fuzzy logic control of robotic behaviors and evolutionary fuzzy systems.

At first, a comprehensive fuzzy behavior based architecture is proposed to control multiple robots in a robot soccer system. The architecture sets up a hierarchical system to decompose the system into modules of roles, behaviors and actions, according to their complexity. Fuzzy logic is employed to realize all these modular behaviors, as well as the behavior coordination. In this architecture, both the behaviors and related fuzzy logic controllers are simple enough to develop. The successful implementation in a robot soccer system in the real-world environment demonstrates the effectiveness of the proposed architecture.

To further improve the system, an adaptive tuning methodology for the fuzzy behavior based architecture is proposed. The tuning method focuses on the adjustments of fuzzy membership functions. The methodology is suitable for off-line tuning of the fuzzy behaviors in a robot soccer system, helping the system to handle unpredictable system changes. Experimental results demonstrate the effectiveness

of this method.

With the help of a robot soccer simulator, genetic algorithm is used to evolve the fuzzy behaviors at different levels of the fuzzy behavior based architecture. Both the membership function tuning and rule base learning are utilized in the evolutionary fuzzy system. Fuzzy behaviors at different levels of the hierarchy architecture are evolved, resulting in performance improvements observed both in the simulation and real-world environments.

Associated with the work on evolutionary fuzzy system, DNA like coding methods for genetic algorithms are also developed and explored. Such coding methods are context dependent, redundant and allow variable lengths of individual strings. The proposed coding methods are applied to GA in rule base learning for role assignment in a robot soccer system. Two different DNA coding methods and the integer coding are used for the same application and comparisons are made. The context dependent DNA coding method shows advantages over position dependent coding methods in handling the negative effects of epistasis. The intron parts in DNA coding decease the chances of good schemata being destructed, while the redundancy increases the population diversity. Furthermore, the variable string length makes it possible for GA to optimize the size and structure of fuzzy rule base at the same time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis comprises of research on fuzzy logic controller (FLC), multiple robotic systems and genetic algorithms (GAs). A comprehensive fuzzy behavior based architecture is developed to control a multiple robotic system. The architecture is realized on a real world robot soccer system. To further improve this architecture, adaptive tuning is incorporated. Furthermore, DNA like coding genetic algorithms are developed and explored.

1.1 Background and Motivations

1.1.1 Fuzzy Logic

The concept of "fuzzy logic" is introduced by Prof. Lotfi A. Zadeh of University of California at Berkley in the 1960's as a means to model the uncertainty in natural language [1]. There are two ways of understanding the notion of fuzzy logic [2]. In a narrow sense, fuzzy logic is an extension of classic Boolean logic aiming to work with imprecise or vague data. It is a branch of multi-valued logic based on the paradigm of inference under vagueness [3, 4, 5]. On the other hand, fuzzy logic in the broad sense serves mainly as an apparatus for fuzzy control, analysis of vagueness in natural language and several other application domains [6, 7]. It is an important member of the class of techniques named as *soft-computing*, i.e.

computational methods tolerant to sub-optimality and impreciseness (vagueness) and providing quick, simple and sufficiently good solutions.

Solving problems using classical logic often requires a deep understanding of the system, exact equations, and precise parametric values. Fuzzy logic incorporates an alternative way of thinking, which allows complex systems to be modeled using a higher level of abstraction originating from human's knowledge and experience. Fuzzy Logic allows expressing this knowledge with natural linguistic concepts such as very hot, bright red, and a long time, which are mapped into numeric ranges. In this way, fuzzy logic resembles human decision making with its ability to handle approximate data.

Fuzzy logic has been successfully applied to control systems in the past decades. Starting form the first industrial application on the control of cement kiln [8], there are over thousands of commercial and industrial applications of fuzzy logic, ranging from domestic electronic products, high speed train to aeroplanes and missiles [9, 10, 11]. Other application areas of fuzzy logic include expert system [12, 13] and information retrieval system [14].

1.1.2 Genetic Algorithm

Genetic algorithm (GA) [15, 16, 17] belongs to the research field of evolutionary algorithm, which is a class of algorithms inspired by the biological evolution. Stimulated by the studies of cellular automata, GA directly mimics the natural processes driving the evolution.

In GA, the biological DNA chromosomes are modeled as strings of parameters representing trial solutions to certain problem. Each solution is evaluated and assigned a numerical value named as fitness, according to a fitness function. During successive iterations, the population of strings undergoes a process of fitness-based selection and parameter recombination in pairs. Such a process simulates the Darwin's principle of "survival of the fittest" in natural selection and the sexual recombination of genetic materials. As a result, a better population is supposed to

appear, and some characteristics of parent strings are inherited by offspring strings. The evolution process of population goes on until some criterion of fitness or time is satisfied.

The inception of the genetic algorithm is dated to 1975 when John Holland's *Adaption in Natural and Artificial Systems* [16] is published. Holland's prototype of GA is usually referred as the "Simple GA" (SGA). Different spinoffs of SGA are developed with modifications and enhancements on all aspects of GA: the representation, operators, evaluation, etc. Some important variants include the messy GA [18, 19, 20], parallel GA [21, 22, 23], distributed GA [24, 25, 26] and multi-objective GA [27, 28, 29]. The specific characteristics of GAs are quite dependant on the applications. However, the fundamental mechanism is the same, which consists of the evaluation of individual fitness, formation of a gene pool through selection and, recombination through crossover and mutation operations.

It is literally possible for GAs to operate on a problem without any knowledge of the task domain but utilizing only the fitness of the evaluated solutions. The applications of GA span a wide range of problems including industrial optimization and design [30, 31], neural network design [32, 33], management and financial systems [34, 35], communication network [36, 37] and many others.

1.1.3 Robots and Behaviors

The word "robot" originated from the Czech word *robota* for "forced labor", or "serf". It was firstly introduced by Czech playwright Karel Capek in his 1920 play R.U.R. (Rossum's Universal Robots). There is no standard definition for a robot. However, basically a robot consists of:

- A mechanical device, such as a wheeled platform, arm, or other construction, capable of interacting with its environment
- Sensors on or around the device which are able to sense the environment and provide useful feedback to the device

- Control systems that process the sensory input in the context of the device's current situation and instruct the device to perform actions in response to the situation
- Power unit to supply energy to the components of robot for its normal operation.

Robots often function to relieve human beings from dangerous and tedious works. They are also suitable for the jobs characterized by repetition and precision. Nowadays, robots are extensively used in fields like manufacturing, military operations and space explorations [38, 39].

Sometimes, a self-reliant robot like the planetary rover has to modify its actions to respond to a changing environment [39]. The need to sense and adapt to a partially unknown environment require intelligence. Thus, the research on robotics is closely associated with the Artificial Intelligence (AI). Knowledge based systems (KBS) was initially developed to simulate the human intelligence. KBS is effective in simulating abstract ways of exhibiting intelligence, for successfully solving problems or playing chess [40, 41]. It is difficult for KBS to simulate successfully "very simple" tasks (from an intellectual point of view), such as cleaning and parking a car. Basically, these tasks do not demand much intellectual efforts, but require a lot of coordinations and complex interactions with the environment. It is clear that modeling "simple" intelligence by KBS is neither easy, nor computationally efficient. To handle this issue, researchers began to model intelligence based on behaviors, instead of on knowledge.

The notion of behavior is subject to different forms of interpretations. A behavior can be a reaction to some stimulus from the environment. Meanwhile, a behavior can also be an exhibition of an action based on some inherent needs of the system to achieve a certain goal. These actions and reactions are primitive and reflexive by themselves. However, very complicated behaviors can emerge based on them, enabling the system to achieve its objectives [42]. The concept of behavior based robotics was popularized by Rodney Brooks in the mid-1980s [43]. The

behavior of robot can be certain loosely defined actions, which may be at a variety of levels of complexity and competence. For instance, both the actions "move backward" and "avoid obstacle" are behaviors, while the latter is obviously more complicated than the former.

1.2 Previous Works

1.2.1 Fuzzy behavior based robotic system

Together with the robotic behavior, Brooks also introduced the idea of behavior based system (BBS) [43]. Inspired by the field of ethology, which studies animal behaviors, Brooks proposed a layered behavior based subsumption architecture which decomposes the overall control systems into a set of reactive behaviors. The reactive behaviors represent the system's ability to interact with the environment. Different layers work on individual goals concurrently or asynchronously. Low level behaviors are able to run in real-time since they require less computation. It is observed that many systems consisting of a few simple components are capable of exhibiting highly complex behaviors.

The behavior based robotic system exploits such kind of inherent complexity. The basic idea is that a simple controller, carefully designed with particular attention to possible interactions with the environment, can display a surprising level of complexity and sophistication. On the other hand, the decomposition of a complicated system into various simpler behavioral modules seems to be an effective way of implementing large scale control systems.

Brook's subsumption architecture adopts a purely reactive behavior based approach. Behavior coordination in subsumption architecture is mainly accomplished by inhibition and suppression mechanisms, which are usually predefined and fixed. Only one behavior dominates at any time. Extensions to this architecture enable the system to handle more complex tasks [42]. For instance, the mission planner, spatial planner and plan sequencer can be used to advise a reactive component [44].

A planner can also acknowledge failure and adapt the reactive controller accordingly [45], or even produce continuous modification of a reactive system according to the high-level goal [46]. In general, the system incorporated with deliberative behaviors and enhanced coordination mechanism is capable of achieving multiple and conflicting objectives.

One of the major extensions to the behavior architecture is the incorporation of fuzzy logic. Being capable of inferencing and reasoning under uncertainty [47, 48], fuzzy logic makes itself favorable in the behavior architecture [49, 50, 51, 52, 53]. Meanwhile, fuzzy control can be adopted to coordinate the various behaviors of the system in response to the environment, just as how human beings manage their multitudes of behaviors and mannerisms while negotiating with reality. Furthermore, the combined usage of fuzzy control with behavior based architecture has the additional advantage of having a distributed fuzzy control system with smaller fuzzy sub-systems, instead of a big and centralized one. Such an approach saves a lot of computational expenses and sometimes this is the only way out to control very complex systems.

The study of fuzzy behavior based decision control in mobile robots can be considered at several levels. Simple behaviors of individual robot are realizable by fuzzy logic controller [54, 55, 56, 57, 58]. These fuzzy behaviors include robotic navigation, obstacle avoidance and objective seeking. When primitive behaviors are combined to generate more complex ones, the mechanism of behavior fusion and selection can also be fulfilled by fuzzy logic [59, 60, 61, 62, 63]. With coordination mechanism between individual robots, the concept of behavioral architecture implemented by fuzzy logic has been further extended to the multiple robot scenarios [64, 65, 66, 67]. Individual robot agents can display a certain behavioral aspect of the group, and together, they exhibit collective behaviors of the whole organization.

1.2.2 Evolutionary fuzzy system

Fuzzy logic system has been successful in a considerable number of applications. In most cases, the success of the fuzzy system is highly dependent on the availability of human expert's knowledge. Meanwhile, the construction of fuzzy membership functions appears to be the most time consuming aspect of fuzzy system design. The lack of learning and adaptation ability of fuzzy system has motivated research activities on combining the fuzzy systems with other techniques since the 1990's. One of the most successful approaches are the hybridization with genetic algorithms [68, 69], leading to evolutionary fuzzy systems.

Literature survey suggests that the prominent types of evolutionary fuzzy systems involve genetic learning or tuning of various components of a fuzzy rule-based system [68]. Genetic algorithms are applied at different levels of complexity [70], from membership function tuning to fuzzy rule generation, that is, adaptation and learning.

The first article addressing the union of GAs and fuzzy appeared in 1989 by Karr [71]. The article acknowledges the difficulty of selecting membership functions for an efficient fuzzy logic controller and describes an approach for membership function tuning involving the use of GA. It does not take a long time for the GA membership tuning to become popular and be widely applied to various fuzzy systems [72, 73].

In the tuning of membership functions, the membership functions associated to linguistic variables are parameterized and encoded as chromosome strings. The most common shapes for the membership functions are triangular (either isosceles [72, 74] or asymmetric [75]), trapezoidal [73] and Gaussian [76]. Accordingly, the number of parameters per membership function usually ranges from one to four, each parameter being either binary [77] or real coded [78].

Following Karr's works, other researchers soon extended the use of GA in the development of FLCs. Thrift suggested the use of GAs both for selecting the rule set and for tuning membership functions [79]. Thrift applied such an approach to

a simulated translating cart and the results indicate that the GA-designed FLC had the performance of an optimal controller.

Three major approaches are considered dominant in the genetic learning of rule bases: Pittsburgh [79, 80, 81], Michigan [82, 83, 84] and iterative rule learning [85, 86, 87].

The Pittsburgh approach is characterized by encoding the entire rule base as an individual string. The population is a pool of candidate rule bases manipulated by GA operations. The Michigan approach, on the other hand, represents the whole population as one rule base while each individual stands for a single fuzzy rule. Pittsburgh and Michigan approaches are the most wildly used methods for fuzzy rule learning. In the iterative rule learning approach, individual strings encode single rules. In each generation of GA, a new rule is adapted and added to the rule base in an iterative fashion.

The above works handle the membership tuning and rule learning as two independent procedures. In 1995, Homaifar and McCormick tried to combine the two processes into one by simultaneously developing the rule base and tuning the membership functions with GAs [88]. They argue that the performance of an FLC is dependent on the coupling of the rule base and the membership functions. Their results indicate that GAs do have the capability to generate a rule base and tune membership functions at the same time. However, whether or not the simultaneous development of the rule base and the membership functions is vital is still unclear.

One important milestone in the research on evolutionary fuzzy system is the development of adaptive fuzzy system. Certain fuzzy control systems contain time-varying parameters which do not always appear directly in the rule base. As a result, the control system is incapable of compensating the changes on the value of these parameters. Researchers have been successful in using GA tuning and adaptation of fuzzy systems on-line in response to the parameter variation that do not appear explicitly in the fuzzy rule base [73, 89, 90, 91].

1.3 Thesis Outline and Contributions

Chapter 2 contains background materials on fuzzy logic, including a brief introduction to fuzzy set theory and the fuzzy inference procedure. As a case study, a fuzzy logic controller is designed to control a two-wheeled mobile robot. The cascading of fuzzy rule bases helps to reduce the number of fuzzy rules which increases exponentially with the number of inputs. The experimental results from both the simulation and real world environments are provided.

Chapter 3 explains the basic components of the genetic algorithms. The structure of a simple genetic algorithm is analyzed, while the schemata theorem is briefly introduced. The GA is then applied to optimize the rule base of a fuzzy logic controller for a two-wheeled robot performing obstacle avoidance task.

The Chapter 4 is dedicated to the robot soccer system, which is utilized throughout the thesis as an experimental setup. The introduction covers the history of robotic soccer, the hardware setting, the architectures of the system and the soccer robot. A mathematical model of the soccer robot is developed, that is crucial in the development of a robot soccer simulator outlined in Chapter 7.

In Chapter 5, an extensive fuzzy behavior based architecture is proposed for the control of multiple mobile robots. Such an architecture decomposes the complex system into modules of roles, behaviors and actions, which are more easily and efficiently controlled. Fuzzy logic is used to realize those behaviors at different complexity levels, as well as for behavior coordination. The proposed architecture is then implemented on the robot soccer system in a real-world environment.

Chapter 6 discusses an adaptive tuning methodology for the fuzzy behavior based architecture proposed in Chapter 5. The tuning methods focus on the automatic adjustment of fuzzy membership functions. The methodology is suitable for tuning the fuzzy behavior system.

Chapter 7 deals with the evolutionary fuzzy behavior based architecture for a multi-robotic system. With the help of a simulator for robot soccer system, genetic

algorithm is used to evolve the fuzzy behaviors at different levels of the behavior architecture. Both the membership function tuning and rule base learning are explored. The effectiveness of such an approach is justified through simulation study and validated with real-world experimentations.

Chapter 8 is devoted to the novel DNA like coding methods for evolutionary algorithm. Such coding methods are context dependent and allow variable lengths for individual strings. To explore the features of the DNA coding methods, the proposed coding methods are applied to GA rule base learning for role assignment in the robot soccer system. Two different DNA coding methods and the integer coding are compared.

Finally the thesis concludes in Chapter 9 with a brief on the major results obtained and an outline of possible directions for future research.

The contribution of this thesis is summarized as follows:

- An fuzzy behavior based architecture for multiple robotic system is proposed.
- The proposed architecture is applied to a real world robot soccer system.
- An adaptive tuning method is applied to the fuzzy behavior based robot soccer system.
- Evolution of the fuzzy behaviors are realized on simulator developed in house.
- The DNA coding methods for GA are projected in a general scheme and their specific features are explored.

Chapter 2

Fuzzy Logic Systems

It has been 40 years since the concept of fuzzy logic is conceived by Lotfi A. Zadeh, a professor of the University of California at Berkley, in the 1960s [1]. Fuzzy technology is first developed in the United States and it has bloomed into a billion dollar industry in Europe and Japan. Fuzzy systems have demonstrated their ability by successful applications on different kinds of problems in various domains, from the control of washing machine to the medical diagnosis for patients.

This chapter begins with an introduction to the definition and origin of fuzzy logic. The fundamental fuzzy set theory is then outlined, followed by a section describing the structure of the fuzzy control system. Some of the complex issues related to fuzzy logic are further discussed. The chapter ends with a detailed example of applying fuzzy logic on a two-wheeled mobile robotic system.

2.1 Introduction to Fuzzy Logic

2.1.1 What is fuzzy logic?

Fuzzy logic is a mathematical problem-solving methodology which provides rules and functions to deal with natural language queries. Natural language abounds with vague and imprecise concepts, such as "It is very hot today." Such statements are difficult to translate into more precise language without losing some of their

semantic values. At how many degrees of temperature the weather can be called as "hot" and at which instant it changes from "cold" to "hot"? It is hard to provide precise and exact answers to these questions. In fact, there are some stages when it is both "cold" and "hot" to some extent . Conventional logic, which is by nature related to the Boolean conditions (true/false), is not suitable for such ambiguous statements. There is a loss of richness of meaning when one tries to translate natural language into conventional logic.

In the viewpoint of set theory, fuzzy logic is a super set of the conventional (or Boolean) logic which has been extended to handle the partial truth - truth value between the absolute truth and absolute false. Fuzzy logic differs from conventional logic in that the statements are no longer black or white, true or false, on or off. In traditional logic, an object takes on a value of either zero or one; in fuzzy logic, a statement can assume any real value between 0 and 1, representing the degree of truth. Fuzzy logic provides a simple way to draw a definite conclusion based upon vague, ambiguous, or even missing input information.

Fuzzy logic lends itself to implementations in systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or work-station based data acquisition and control systems. It can be implemented in hardware, software, or a combination of both. Human beings can reason with uncertainties and vagueness, and they are capable of highly adaptive and efficient control. Fuzzy approach to control problems mimics how a person makes decisions. With the tolerance to noisy and imprecise input, fuzzy logic based controllers are more effective and perhaps easier to implement.

2.1.2 Where did fuzzy logic come from?

Throughout the history, true and false relationships have been the primary focus in the logic development. Back to 500 B.C., Buddha in India developed his philosophy based on the thoughts that the world is filled with contradictions. He claims that almost everything contains some of its opposites, or in other words, that things can

be A and not-A at the same time. There is a clear connection between Buddha's philosophy and modern fuzzy logic.

In Europe, for several hundred years, philosophers such as Parminedes, Plato, and Aristotle, devoted themselves to devise a concise theory of logic, and later mathematics. Due to their efforts, the so-called "Laws of Thought" were posited. One of these, the "Law of the Excluded Middle," states that every proposition must either be true or false. Even when Parminedes proposed the first version of this "law of non-contradiction" around 400 B.C., there were strong and immediate objections. For example, Heraclitus argues that contradictions not only exist but are essential and the basis of a thing's identity.

It was the Greek philosopher Plato who laid the foundations for the fuzzy logic by proposing a third region beyond true and false where the two notions tumbled together. Other, more modern philosophers echoed his sentiments, notably Hegel, Marx, and Engels. But it was Lukasiewicz who first proposed a systematic alternative to the bi-valued logic of Aristotle.

In the early 1900's, Lukasiewicz described a three-valued logic, along with the mathematics to accompany it. A new truth value was added to the truth logic 0 and the false logic 1. This third value was termed possible with a logic value of 1/2. Eventually, Lukasiewicz proposed an entire notation and axiomatic system from which he hoped to derive modern mathematics. Later, he explored four-valued logics, five-valued logics, and then declared that in principle there was nothing to prevent the derivation of an infinite-valued logic. Lukasiewicz felt that three- and infinite-valued logics ware the most intriguing, but he ultimately settled on a four-valued logic because it seemed to be the most easily adaptable to Aristotelian logic. Unfortunately, the logic of Lukasiewicz never gained wide acceptance and remained unknown by most people outside of professional logisticians.

It was not until relatively recently that the notion of an infinite-valued logic took hold. In 1965, using the ideas of multi-valued logic, Lotfi A. Zadeh derived the multi-valued logic rules in terms of set theory [1]. Zadeh aimed to develop a model that could more closely describe the natural language process. He defined some of

the basic terminology associated with fuzzy logic such as: fuzzy set theory, fuzzification, fuzzy quantification and fuzzy events. Fuzzy set theory allows function (or the values False and True) to operate over the range of real numbers [0.0, 1.0]. New operations for the calculus of logic were proposed, and seemed to be in principle at least a generalization of classic logic. It took a long time until fuzzy logic got accepted even though it fascinated some people right from the beginning. Besides engineers, philosophers, psychologists and sociologists soon became interested in applying fuzzy logic into their sciences.

2.2 The Fuzzy Set Theory

The rather abstract concept of a set forms a fundamental building block of modern mathematics and logic. Without exception, the formal basis for the fuzzy logic is known as fuzzy set theory, originally described by Zadeh.

There is a strong relationship between the traditional (crisp) set and the concept of fuzzy set.

A traditional or crisp set can formally be defined as the following:

- A subset U of a set S is a mapping from the elements of S to the elements of the set $\{0, 1\}$. This is represented by the notation: $U : S \rightarrow \{0, 1\}$.
- The mapping is represented by one ordered pair for each element S where the first element is from the set S and the second element is from the set $\{0, 1\}$. The value zero represents non-membership, while the value one represents membership.

Essentially such a definition means that an element of the set S is either a member or a non-member of the subset U. There is no partial member in traditional sets, which is known as the "dichotomy principle".

For conventional sets, the memberships of the elements are determined by precise properties. For example, set H is the subsect of the real number set R and,

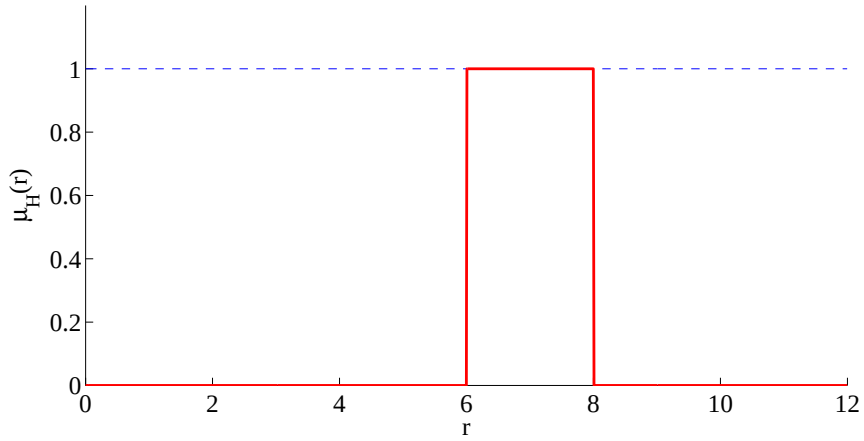


Figure 2.1: A traditional set

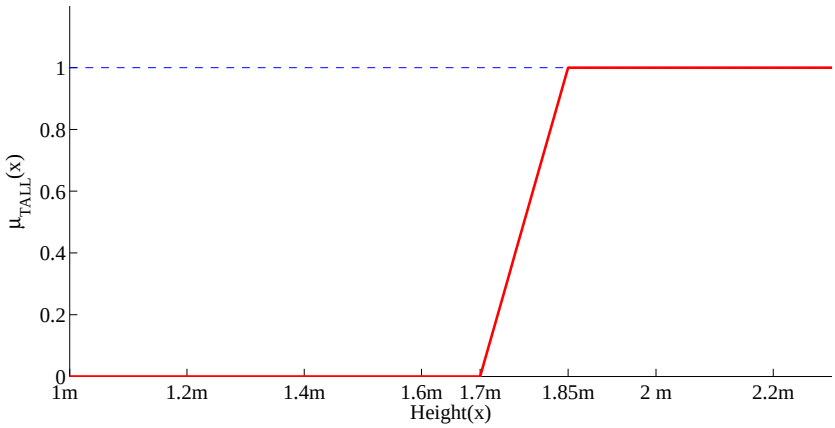


Figure 2.2: A fuzzy set

H contains all the real numbers between 6 and 8: $H = \{r \in R \,|6 \leq r \leq 8\}$. Equivalently, H is described by its membership function, μ_H:

$$\mu_H(r) = \begin{cases} 1 & : 6 \leq r \leq 8, \\ 0 & : otherwise. \end{cases} \tag{2.1}$$

The membership function μ_H is depicted in Figure 2.1. In the figure, the interval on the r-axis between 6 and 8 has a membership of 1 which indicates that any number in this interval is a member of the set H. Any number that has a membership of 0 is considered to be a non-member of the set H.

A fuzzy set is a set whose elements have degrees of membership. That is, a member of a set can be full member (100% membership status) or a partial member (eg. less than 100% membership and greater than 0% membership). These can formally be defined as the following:

- A fuzzy subset F of a set S can be defined as a set of ordered pairs. The first element of the ordered pair is from the set S, and the second element of the ordered pair is from the interval $[0, 1]$,
- The value zero is used to represent non-membership; the value one is used to represent complete membership, and the values in between are used to represent the degrees of membership.

The set S is referred to as the "universe of discourse" for the fuzzy subset F. Frequently, the mapping between elements of the set S and values in the interval $[0, 1]$ is described as the membership function of F.

For example, "tallness" of people are described using fuzzy sets. In this case the set S (the universe of discourse) is the set of people. A fuzzy subset $TALL$ is defined to answer the question "to what degree the person x is tall?" To each person in the universe of discourse, a degree of membership is to be assigned in the fuzzy subset $TALL$. That is done by a membership function $\mu_{TALL}(x)$ based on the person's height $height(x)$ (Figure 2.2) .

$$\mu_{TALL}(x) = \begin{cases} 0 & : height(x) < 1.7m, \\ \frac{height(x)-1.7}{0.15} & : 1.7m \leq height(x) < 1.85m \\ 1 & : height(x) \geq 1.85m. \end{cases} \tag{2.2}$$

Given this definition, if Sean's height is 1.73m, the degree of truth of the statement "Sean is TALL" is 0.20.

2.3 Operations of Fuzzy Set

The traditional set theory developed by Cantor contains some fundamental operations on sets: the complement, intersection and union operations. Zadah formally defines the counterparts of these operations for the fuzzy sets.

2.3.1 Complement

Given a fuzzy set A with membership function μ_A, the membership function of the complement set $\bar{A}$ is defined as follows (Figure 2.3).

$$\mu_{\bar{A}} = 1 - \mu_A$$

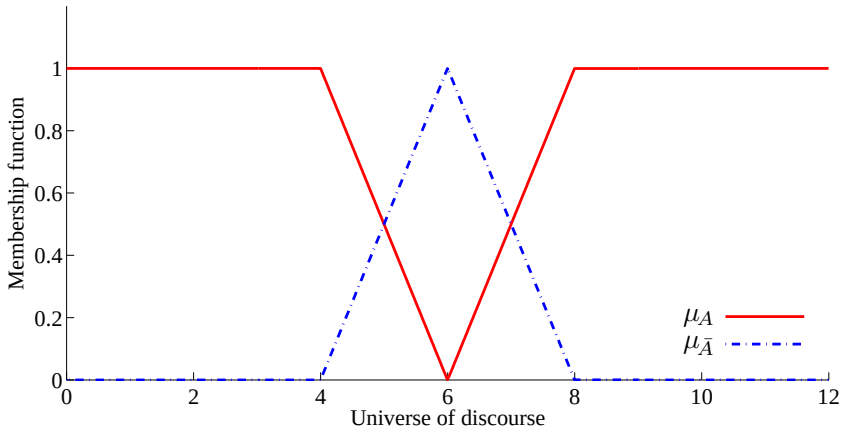


Figure 2.3: The complement operation on fuzzy set

The complement operation in fuzzy set theory is the equivalent of the NOT operation in Boolean algebra.

2.3.2 Intersection

Under classical set theory, the intersection of two sets is that set which satisfies the conjunction of both the concepts represented by the two sets. However, under fuzzy set theory, an item may belong to both sets with differing memberships without

having to be in the intersection. The membership function of the intersection of two fuzzy sets A and B with membership functions μ_A and μ_B respectively is defined as the minimum of the two individual membership functions (Figure 2.4). This is called the minimum criterion.

$$\mu_{A \cap B} = min(\mu_A, \mu_B)$$

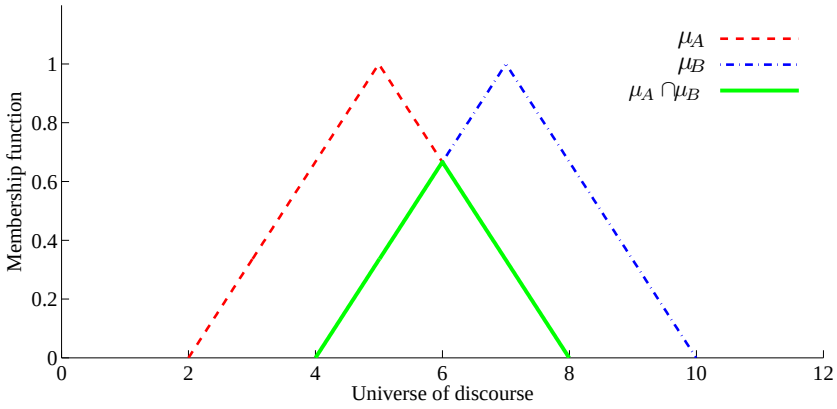


Figure 2.4: The intersection operation on fuzzy set

The intersection operation in fuzzy set theory is the equivalent of the AND operation in Boolean algebra.

2.3.3 Union

The membership function of the union of two fuzzy sets A and B with membership functions μ_A and μ_B respectively is defined as the maximum of the two individual membership functions (Figure 2.5). This is called the maximum criterion.

$$\mu_{A \cup B} = max(\mu_A, \mu_B)$$

The Union operation in fuzzy set theory is the equivalent of the OR operation in Boolean algebra.

It worths mention that the last two operations, Intersection (AND) and Union (OR), represent the clearest point of departure from a probabilistic theory for sets to fuzzy sets.

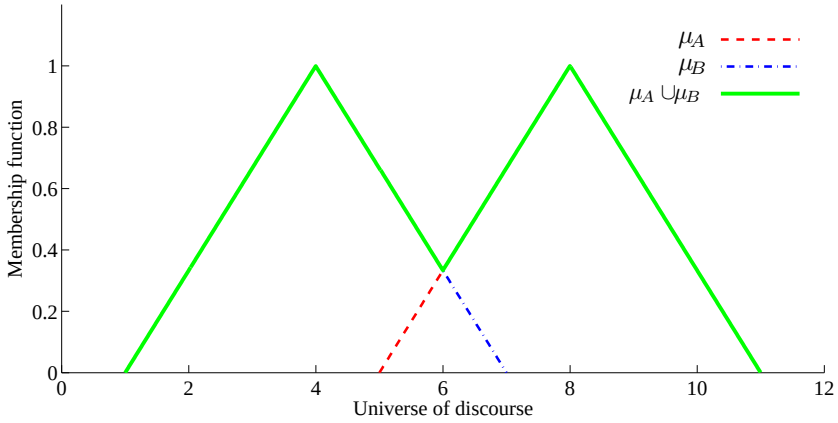


Figure 2.5: The union operation on fuzzy set

2.3.4 Algebraic Symmetries

The fuzzy set operations obey the same algebraic symmetries as crisp sets. The following rules which are common in crisp set theory also apply to fuzzy set theory.

Associativity

$$(A \cap B) \cap C = A \cap (B \cap C)$$

$$(A \cup B) \cup C = A \cup (B \cup C)$$

Commutativity

$$A \cap B = B \cap A, \qquad A \cup B = B \cup A$$

Distributivity

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

De Morgan's law

$$\overline{(A \cap B)} = \overline{A} \cup \overline{B}, \quad \overline{(A \cup B)} = \overline{A} \cap \overline{B}$$

2.4 Linguistic Variables

One of the most important tools in applications of fuzzy set theory is the concept of linguistic variables. The linguistic variables play a central role in the modeling of approximate reasoning by fuzzy sets. Just as numerical variables take numerical values, in fuzzy logic, linguistic variables take on linguistic values which are words (linguistic terms) with associated degrees of membership in the set.

Zadeh's original definition of a linguistic variable is rather inspired by computational linguistics and classical artificial intelligence. The formal definition is very sophisticated and general. The linguistic variable is a quintuple (N, G, T, X, S), where N, T, X, G, and S are defined as follows:

1. N is the name of the linguistic variable.
2. G is a grammar.
3. T is the term-set.
4. X is the universe of discourse.
5. S is a $T \rightarrow f(X)$ mapping which defines the semantics - a fuzzy set on X - of each linguistic expression in T.

The motivation for such a sophisticated structure is to provide the freedom and integrality. In practice, only three of these elements are important. At first, there is the name N of the linguistic variable itself, such as "Hight". The second important element is the term-set T, which lists the possible members of the linguistic variable. The members of the linguistic variable are sometimes called "linguistic terms" or "linguistic values". For instance, the linguistic variable "Speed" may be a discrete fuzzy set whose members (term-set) are "Low", "Medium" and "Tall". The third important element of a linguistic variable is the membership function S. These functions map an input number onto grades of membership of the linguistic terms. Membership functions are almost always continuous fuzzy sets. Sometimes, especially in engineering-oriented domains like fuzzy control, the name of a member

Key Word	Effect on set characteristics
• about	Approximate the set
• near	
• close to	
• approximately	
• not	Complement the set
• somewhat	Dilute the set
• rather	
• quite	
• very	Intensify the set
• extremely	

Table 2.1: Linguistic effects of hedges

of a linguistic variable is also used to denote its membership function. For instance, "Low" is a member of the discrete fuzzy set "Height", but "Low" is also used to denote its membership function.

An important concept relating to the linguistic variable is hedging. Hedges are a common set of operations on linguistic variables. Just as in the English language, hedges can be described as modifier for linguistic variables which are not only adjectives, but also verbs, adverbs and certain complete statements. Hedges modify a linguistic variable's shape, or membership function, to reflect the variation on its semantics.

When referring to a fuzzy set, hedges are used to adjust the characteristics of that fuzzy set by either: approximating, complementing, diluting or intensifying. Some specific words and their effects on the fuzzy set are shown in Table 2.1.

In general, when a hedge is used to dilute a set, the set is expanded. When a set is intensified with a hedge, the set is compressed. Figure 2.6 visualizes the effect of hedges on membership functions for the "very", "somewhat" and "indeed." The overlap between sets, such as the Medium and Tall sets, is not an error. It is in this region that a variable can have multiple memberships, overcoming the

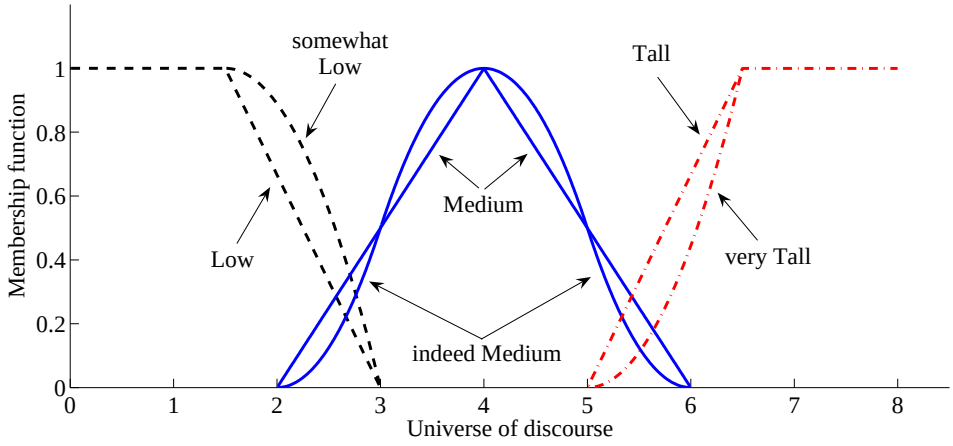


Figure 2.6: The effect of the hedges on the membership function

shortcomings of the binary logic.

2.5 Fuzzy Inference

Armed with the theoretical foundations of fuzzy set theory, it is possible to manipulate information represented as degrees of membership of fuzzy sets through the fuzzy inference system. Fuzzy inference is the process of formulating the mapping from a given input to an output, in the form of if-then rules, using fuzzy logic.

2.5.1 Fuzzy if-then rules

With the linguistic variables and fuzzy operators, one can construct if-then rule statements to formulate the conditional statements that comprise fuzzy logic. A single fuzzy if-then rule assumes the form "IF X is a, THEN Y is b", where X and Y are linguistic variables, with a and b as linguistic values.

The IF condition of the rule is called the antecedent or premise, while the THEN implication is known as the consequent or conclusion.

Interpreting an if-then rule involves two distinct parts: a) evaluating the antecedent (which involves fuzzifying the input and applying any necessary fuzzy operators); and b) applying that result to the consequent. In the case of two-valued or binary logic, if the premise is true, then the conclusion is true. For fuzzy rules, the consequent is set to be true to the same degree as the antecedent. In other words, if the antecedent is true to some degree of membership, then the consequent is also true to that same degree.

Both the antecedent and consequent of a rule can have multiple statements.

"IF X is a AND Y is b, THEN U is c AND V is d."

In such cases, all parts of the antecedent are calculated simultaneously and resolved to a single number using the logical operators like fuzzy union (OR) and intersection (AND). The resultant antecedent membership is equally applied to all parts of the consequent.

2.5.2 The process of fuzzy inference system

Fuzzification

Real-world crisp data, such as the statistics over a digital image, must be fuzzified before it can be subject to fuzzy rules. Fuzzification is process of determining the degree of membership of data. It makes the translation from real-world values to fuzzy values using membership functions. The essence of this step is therefore in the determination of the form of the fuzzy sets. This can be derived from empirical results or from expert domain knowledge.

Fuzzy rule evaluation

Using the fuzzified data, the fuzzy rules are evaluated as described above. In some applications, it is desirable to use modified fuzzy rules for the union and the intersection. The application of the antecedent evaluation to the consequents is commonly achieved either by clipping or scaling the consequent membership functions.

Clipping simply places an upper threshold on the consequent membership functions at the level of the antecedent evaluation. Some information about the consequent fuzzy sets is lost during this clipping, but it is used for its computational simplicity. Scaling adjusts the consequent membership functions by multiplying them by the antecedent evaluation result. Although it is used less often, scaling does preserve the forms of the consequent membership functions.

Combination of rule implications

The membership functions of the clipped or scaled consequents are aggregated into a single fuzzy set. Almost without exception, the membership functions are summed to provide the final fuzzy set.

Defuzzification

After computing the fuzzy rules and evaluating the fuzzy variables, it is necessary to translate the results back to the real world, in other words, the crisp value. As the result, the final fuzzy set is defuzzified. There are a variety of differing methods for defuzzification. The most intuitive and common one is the "center of area" (CoA) method. The center of area of the final fuzzy set is

$$\overline{x} = \frac{\int \mu_A(x) \cdot x dx}{\int \mu_A(x) dx},$$

which is evaluated over the universe of discourse for the fuzzy set. To simplify the computation, the integrals can be discretised sums.

$$\overline{x} = \frac{\sum_{i=1}^{n} \mu_A(x_i) \cdot x_i}{\sum_{i=1}^{n} \mu_A(x_i)}$$

The total procedure is summarized in Figure 2.7.

Mamdani and Sugeno inference

There are two types of fuzzy inference systems: Mamdani-style and Sugeno-style. The system introduced so far is the Mamdani inference [92]. A computationally

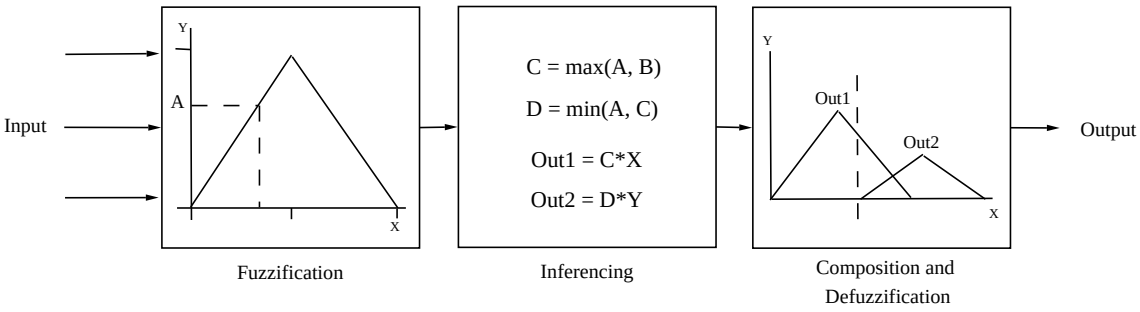


Figure 2.7: Summary of Mamdani fuzzy inference system

cheaper alternative is the Sugeno inference [93], which differs from the Mamdani inference in the way of determining the outputs. Sugeno replaces the membership degree resulting from the antecedents with a singleton, a membership function with a value of zero everywhere except at one chosen point where its value is One. The aggregation therefore leads to a sum of scaled singletons. This leaves the defuzzification as a very simple weighted average:

$$\overline{x} = \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

In practise, Mamdani inference is used when a system aims to emulate the intuitive human expert thought process. Sugeno inference is used in optimization and adaptive algorithms, particularly for control systems. The following case study presents more details on Sugeno inference.

2.6 Case Study: Fuzzy Sensor Fusion for Reactive Navigation of Mobile Robot

Fuzzy logic has been applied in many areas. Fuzzy control system is one of the first practical applications. There have been successful commercial applications, from self-focusing cameras, washing machines to braking control on the subway system. In the artificial intelligence (AI) field, decision making and expert systems are developed, such as the fuzzy medical diagnosis and finance analysis systems.

This section will demonstrate the usage of fuzzy logic with a case study of application in robotics, which is an overlapped field of AI and control. Fuzzy logic controllers are designed to provide the two-wheeled mobile robot with obstacle avoidance behavior. The cascading of fuzzy rule bases reduces the explosion in the number of rules resulted from the increase in the number of inputs. The algorithm is tested within the Webots simulation package [94, 95] and with a real world Khepera robot [96].

2.6.1 Introduction

In recent years, there is an increasing interest in the area of autonomous mobile robots [97, 98]. Autonomous robots typically have some means of propulsion (usually wheels or tracks), sensor array, on-board power supply, and sufficient on-board processing capacity to analyze the sensory inputs and make decisions in accordance with its functional objectives. While traditional industrial robots perform repetitive pre-programmed tasks in a well defined environment, autonomous robots are expected to deal with uncertain environments.

In real world environments, a large degree of uncertainty is present. The robot may be expected to operate in an environment where no prior knowledge of the layout (map) is available. Even if a map is available, several factors limit its usefulness [98, 99]: some details of the environment may have been omitted from the map; data acquired by the robot for navigational purposes through the sensors may be inaccurate; inclement observation conditions or noise in the sampling. Furthermore, real world environments are usually dynamic and subject to change by other agents operating within.

One approach to tackle the uncertainty is to focus on the engineering of the robot and environment. However, such a solution increases the costs, limits autonomy and restricts the range of environments that the robot can operate within [98]. A better alternative is to make use of a suitable control approach to deal with the uncertainty. Fuzzy logic allows for situations where the available data is vague,

imprecise or uncertain [100, 101]. Being not dependent on precise data, fuzzy logic controllers are robust in uncertain environments.

Among the typical functions that an autonomous robot need to perform, obstacle avoidance is considered as an elementary or instinctive behavior and is one of the most representative type of reactive behaviors [97]. In the ALLIANCE architecture [102], a behavior based framework is implemented and is divided into higher-level behaviors such as map building and exploring, and lower-level behaviors such as obstacle avoidance. The higher-level is capable of inhibiting the lower level behaviors when necessary. As a large amount of environmental uncertainty is involved, fuzzy logic is a suitable candidate to realize the obstacle avoidance behavior.

For a fully autonomous robot, with limitations on on-board processing power, the obstacle avoidance component of the controller should consume as little processing power as possible. However, in a fuzzy logic controller the number of rules exponentially increases with the number of inputs and so the processing power required. Cascading method [103, 104, 105] is employed to handle this problem.

This work addresses the design of a fuzzy logic controller for the autonomous Khepera mobile robot [96]. The robot has eight IR/light sensors and two wheel encoders. The 6 front-sensor readings constitute the controller inputs. In order to reduce the complexity associated with a large number of fuzzy inputs, the fuzzy controller rule bases are cascaded.

Khepera robot and Webots software

The Khepera robot [96] is widely used in research laboratories. The Khepera mobile robot is 30mm in height, 55mm in diameter and weigh 70g (Figure 2.8). Khepera supports a large number of hardware extension modules, such as gripper, vision turret and radio turret. At the software level, it has a very efficient library of on-board applications. Programs for Khepera can be developed in standard and well known tools, such as C/C++ and Matlab. With the program downloaded to

Figure 2.8: The Khepera robot

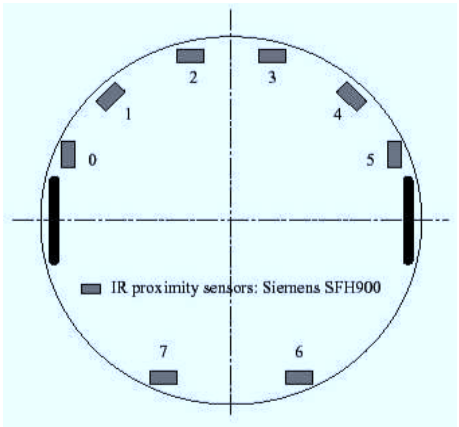


Figure 2.9: The eight infra-red sensors on Khepera robots

its memory, Khepera can run independently.

The detecting system of Khepera robot includes eight infrared proximity and ambient light sensors. Sensors are deployed around the robot (Figure 2.9). Each sensor can return a value between 0 to 512 for ambient light (higher values refer to darker regions) and 0 to 1023 for proximity to obstacles (higher values refer to being closer to obstacles).

Khepera's motion system consists of two wheels, each controlled by a DC motor with an incremental encoder. The maximum speed of Khepera robot is 60 cm/s. For safety reasons, the speed range is set between -20 and 20 units, where the unit of speed is 8 mm/s.

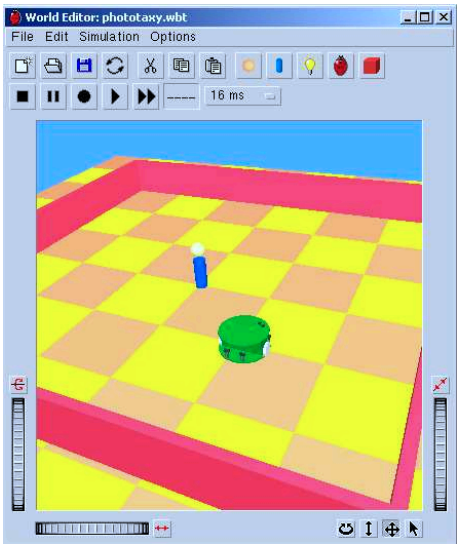


Figure 2.10: The Webots simulation environment

Khepera supports many simulation softwares. Among the softwares, Webots [94, 95] is the most powerful 3D mobile robotics simulator available. It supports the Khepera robot as well as other robots such as Alice and Koala [94, 95]. The user can program virtual robots using C/C++ libraries. The 3D environment editor (Figure 2.10) allows the user to customize the simulation world. Objects such as walls, balls, cans and lamps can be added into the world. The properties such as the size, color, position and orientation of the objects, can be user-defined. Supervisor program allows the programmer to create a supervisor controlling the process of an experimentation.

Webots also provides serial port communication facility to control the real Khepera robot directly. It is also possible to recompile the source code developed using the Khepera cross-compiler and download the resultant program onto the real robot via the serial interface.

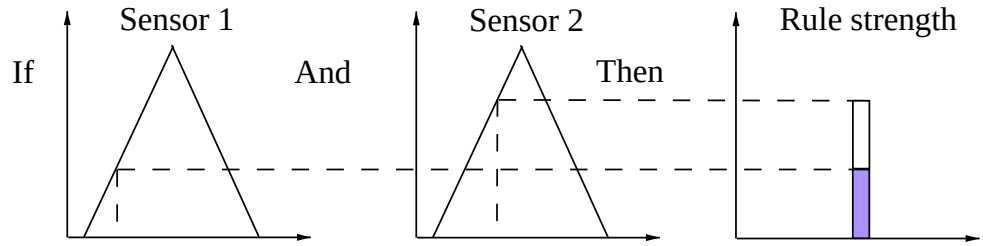


Figure 2.11: Sugeno's method for evaluating rule truth value

2.6.2 Cascaded fuzzy logic controller

Inference engine

The two commonly used methods in the inference process of fuzzy controllers are Mamdani–style and Sugeno–style. The most significant distinction between them is that, all output membership functions are singleton spikes in Sugeno's method, whereas they are distributed fuzzy sets in Mamdani's method.

In this work, the Sugeno's method of determining the rule's truth value is used due to its simplicity and efficiency within the cascaded structure followed. The evaluation of each rule's truth value is based on the minimum of the peak fuzzy membership values of the inputs (Figure 2.11). The truth value is regarded as the rule's strength.

In the early stages of the cascaded fuzzy controller, the truth value and the consequence of each rule are collated and passed on to the subsequent stage as inputs. In the final stage, the rule with maximum truth value is identified and triggered to decide the output action.

Cascaded fuzzy controller for the Khepera Robot

A simple two-input and single-output fuzzy controller is considered, with each input having three linguistic values (high, medium, low). Defining a rule for each combination of the input linguistic values requires nine (3×3) rules. If each input

now has five linguistic values, then twenty-five (5×5) rules are needed, almost three times as many. If there are four inputs instead of two, and all combinations of linguistic values are considered at once, eighty-one (3^4) rules are required.

The Khepera's six front sensor readings form the inputs to the fuzzy logic controller. With three linguistic values for each of the six sensor readings, the possible combinations require $3^6 = 729$ rules. As a result, the evaluation of the rule strengths in each cycle of the controller operation demands considerable processing power.

The cascaded fuzzy controller is utilized to bring down the number of rules, without neglecting any of the sensor readings. Instead of considering all possible combinations of linguistic variables in one stage, the sensors are considered in pairs in different stages. The output of one stage is collated and passed on to the following, which in turn is either evaluated against another input or against the output from a parallel stage.

The cascaded approach has two advantages. The first and most obvious one is that it greatly reduces the number of rules need to be defined and evaluated. Four sensor readings, each with three linguistic values, result in eighty-one rules. However, by taking the sensor inputs in pairs in the first stage, and collating the outputs of each pairs to five output actions, the second stage would need only twenty-five (5×5) rules. This results in forty-three rules, which is almost one half of the rules with all possible input combinations considered. The reduction of rules provides even better returns as the number of inputs or linguistic values per input increases. The second advantage of the cascaded approach is that in a scenario where there is some sort of symmetry (a left pair and a right pair of sensors), the designer may only need to define one set of rules for one pair, and apply a mirror image of those rules to the other, which further simplifies the design.

	Low		Medium			High	
Sensor Value	0	306	206	512	818	717	1023
Membership Value	1	0	0	1	0	0	1

Table 2.2: Parameters of the membership functions

Output Vector	Signal to Left motor	Signal to Right Motor
Forward	15	15
Forward 1/2 speed	10	10
Forward Left	-5	10
Hard Left	-10	5
Forward Right	10	-5
Hard Right	5	-10
U-Turn	20	-20

Table 2.3: Finalized settings for output actions

Membership functions and motor parameters

Triangular membership functions are used and their parameters are tabulated in Table 2.2. The settings of the motor parameters for each output action are in Table 2.3. The fuzzy controller's inference system is of the standard Sugeno type as discussed in Section 2.5. Thus, the details of membership functions and inference engines are not provided here. The focus of this work is the development of the rule base, especially, the cascaded structure.

The two rear sensors are not utilized and the robot is expected to move forward, rotate or perform a combination of forward motion and rotation. Two types of controllers are designed and tested. Firstly, four sensors, two sensors each from the left and right sides, are used. Afterward, the two front sensors are also utilized in addition to the four left and right sensors.

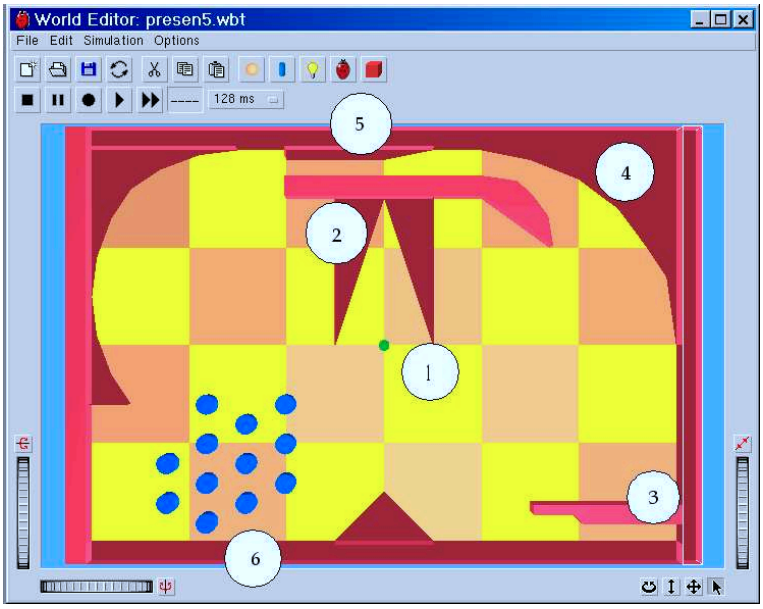


Figure 2.12: Standard test setup

Experimental setup and testing

Two controllers, with four and six sensory inputs respectively, are tested in the same simulation testing ground (Figure 2.12). The setup contains obstacles with various features designed to observe the robot's response in different scenarios. The algorithm is also tested on the Khepera robot in a real world setup.

The various positions of interest in the test field are labeled in Figure 2.12. The Khepera robot's default starting point is the center (the small circular object at point ① of the testing ground. The Khepera is pointed upward, and the first obstacle it encounters is at point ②, which is a narrowing dead end. The purpose of such an obstacle is to check how the robot reacts when it enters a channel of narrowing width. The square dead end at point ③ tests the robot's ability to deal with a dead end similar to encountering a wall head on. A curved surface at point ④ is useful to observe the reaction of the robot to an obstacle that triggers mainly the side sensors. This path leads to the narrowing channel at point ⑤. The narrowing channel tests the ability of the Khepera to opt for a path between obstacles on either sides. A collection of cylindrical objects is at point ⑥, each approximately twice the size (in terms of area) of the Khepera robot. These cylinders provide

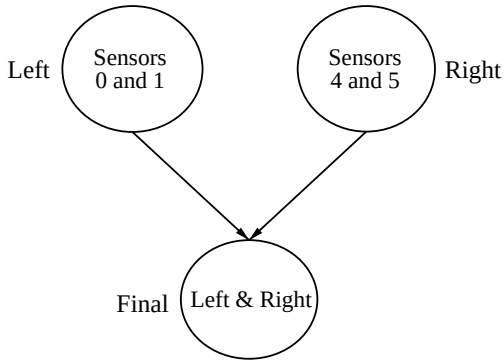


Figure 2.13: Cascaded flow for four-sensor input controller

an environment which is full of sensory input for the Khepera to explore its way through.

Real world testing environment (Figure 2.16) for the Khepera is constructed with obstacles crafted out of grey mounting board. Obstacles are designed to replicate the specific scenarios in simulation (points ②, ③, ④ and ⑤). Scenarios different from those in simulation are also set up to observe the controller's response to new situations, such as a continuously narrowing channel with the end just wide enough for the Khepera to pass (Figure 2.16.(f)) through without the need for a U-turn.

2.6.3 Four-sensor input controller

Cascaded flow and rule set

The cascaded flow for the four-sensor input controller is shown in Figure 2.13. The circles in parallel are inference engines at the same stage. The items described in the circles are inputs to the corresponding inference engine. Among the rules with the same consequent output action, the maximum truth value is identified. Such a maximum value is determined for every possible output action. The maximum truth values for each action are passed as fuzzified input values to the subsequent stage, while the corresponding output actions of that stage are the input variables

to the following stage. The label next to the circle is the name of the rule set associated with that inference engine. The labels are also used as identifiers of the inputs to the following stage.

In Figure 2.13, there are three sets of rules: "Left" for sensors 0 and 1, "Right" for sensors 4 and 5, and "Final" for inputs "Left" and "Right". These rule sets are listed in Tables 2.4, 2.5 and 2.6 respectively. It should be noted that the tables for "Right" and "Left" are identical except that all the right-turns are replaced by left-turns and visa-versa. In fact, only one rule set is designed and the other is derived from it as a mirrored rule base. A total of 27 rules are evaluated when the controller is in operation, that is 33.3% of a conventional rule set with four sensors inputs ($3^4 = 81$ rules). Furthermore, since the rules for the "Left" and "Right" stages are mirrored rule bases, only 18 rules need be generated (22.2% of a conventional rule set).

Left		Sensor 1 (45° Left)		
		Low	Medium	High
Sensor 0 (Extreme Left)	Low	Forward	Forward Right	Forward Right
	Medium	Forward	Forward Right	Hard Right
	High	Forward Right	Hard Right	Hard Right

Table 2.4: Rule set for stage "Left"

Right		Sensor 4 (45° Right)		
		Low	Medium	High
Sensor 5 (Extreme Right)	Low	Forward	Forward Left	Forward Left
	Medium	Forward	Forward Left	Hard Left
	High	Forward Left	Hard Left	Hard Left

Table 2.5: Rule set for stage "Right"

Final		Right		
		Forward	Forward Left	Hard Left
Left	Forward	Forward	Forward Left	Hard Left
	Forward Right	Forward Right	Forward 1/2 Speed	Forward Left
	Hard Right	Hard Right	Forward Right	U-Turn

Table 2.6: Rule set for stage "Final" – the four-sensor case

Testing and observations

The controller performed poorly in the beginning. The robot made a "U-turn" almost every time when an obstacle is encountered. This issue is taken care of by reducing the number of rules at the "Final" stage whose output action is "U-turn". Only one rule is included with "U-turn" as output action in Table 2.6. "U-turn" as an output action is not considered in the rule sets at earlier stages, as it is a premature decision to make a U-turn at the first stage itself with inputs from one side of the robot alone.

In both the simulation and real world testing, it is observed that the robot is able to avoid the straight wall ahead but not the cylindric obstacles. In the later scenario, the robot is observed to collide with obstacles and not able to move out. This is the limitation of the four-sensor input controller where the two front sensors are not utilized. The left and right sensors (sensors 1 and 5 at 45° in Figure 2.9) on either side can detect a straight wall in the front but not the cylindrical objects. The front sensors (2 and 3) are included in the six-sensor input controller to take care of such scenarios.

2.6.4 Six-sensor input controller

Cascaded flow and rule set

The first version of six-sensor controller is designed by making use of the previous four-sensor one. One more stage is added, where the inputs are the output from

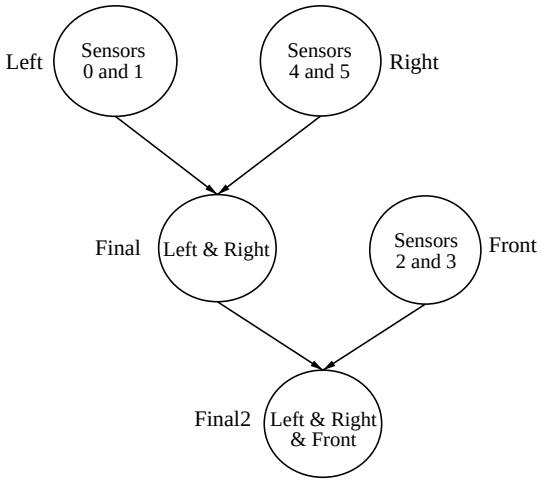


Figure 2.14: Cascaded flow for six-sensor input controller (1st version)

stage "Final" of the four-sensor case, plus the input from the combination of sensors 2 and 3 (Figure 2.14). With the resultant controller the robot got stuck in the triangular dead end (point ② in Figure 2.12). It is observed that: to get out of the trap point, the robot tried to rotate to the left or right. However, in the triangular end, the robot met obstacle in the other side. The controller made the robot to turn in a reverse direction, bringing it back to its original orientation. The robot then repeated these behaviors, falling into a loop.

Modifications on the above controller result in the final version of controller which maintains a symmetrical information flow throughout the cascade (Figure 2.15). The output of first stage of four-sensor controller is paired with the readings from sensors 2 and 3 as inputs to the second stage which is "Left 2" and "Right 2". The following is an example rule for "Left 2":

If "Left" is Forward and Sensor 2 is High, then output is Hard Left.

The output of this rule appears contrary to all the other outputs in Table 2.7, where the direction is either straight or to the right. However this rule is a reasonable one arrived on a "trial and error" basis. In situations where the sensors on the two sides indicate no obstacles, but the forehead sensor for one side does (Here is left side), it could be concluded that there are more obstacles on that (left) side of the robot. Thus the robot should turn back, rather than moving towards it.

Left2		Sensor 2 (Front Left Sensor)		
		Low	Medium	High
Left	Forward	Forward	Forward Right	Hard Left
	Forward Right	Forward Right	Hard Right	Hard Right
	Hard Right	Hard Right	Hard Right	Hard Right

Table 2.7: Rule set for stage "Left2"

Final		Right2			
		Forward	Forward Left	Hard Left	Hard Right
Left2	Forward	Forward	Forward Left	Hard Left	Forward Right
	Forward Right	Forward Right	Forward 1/2 Speed	Forward Left	Hard Right
	Hard Right	Hard Right	Forward Right	U-Turn	Hard Right
	Hard Left	Forward Left	Hard Left	Hard Left	Hard Right

Table 2.8: Rule set for stage "Final" – the six-sensor case

Both simulation and real world testing show that the robot with this controller has no trouble in navigating through any of the obstacles in the testing environment. Especially, the triangular dead end obstacle did not trap the robot into a looping behavior.

The cascaded flow for the final six-sensor input controller is shown in Figure 2.15. Another stage is added at the output of the "Left" and "Right" stages named "Left2" and "Right2" respectively. The controller attempts to maintain a symmetrical flow throughout. The output from the "Left" ("Right") stage and the reading from "sensor 2" ("sensor 3") form the inputs to the "Left2" ("Right2") rule base. Rule sets for the first stage remain the same as in Tables 2.4 and 2.5. The related rule sets for the additional and final stages are listed in Tables 2.7 and 2.8. The table for "Right2" is omitted as it is a lateral inversion of Table 2.7.

It is observed that maintaining a high degree of symmetry in the cascaded flow simplified the development of rule sets, and also increased the efficiency of the controller. A traditional six sensor fuzzy logic controller needs $3^6 = 729$ rules. The controller discussed requires the evaluation of just 52 rules (7.1% of 729 rules).

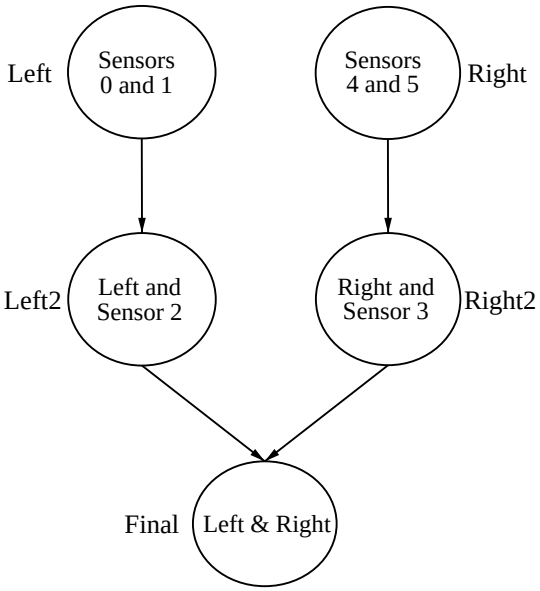
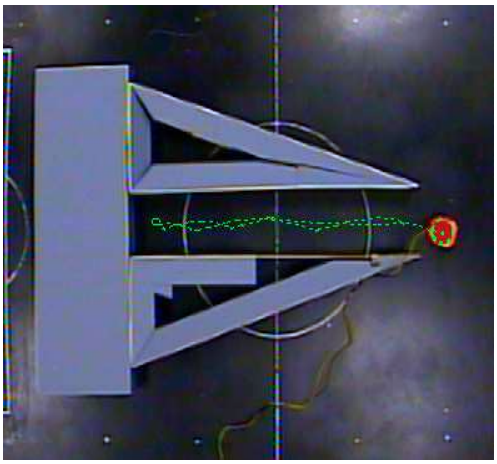


Figure 2.15: Cascaded flow for six-sensor input controller (final version)

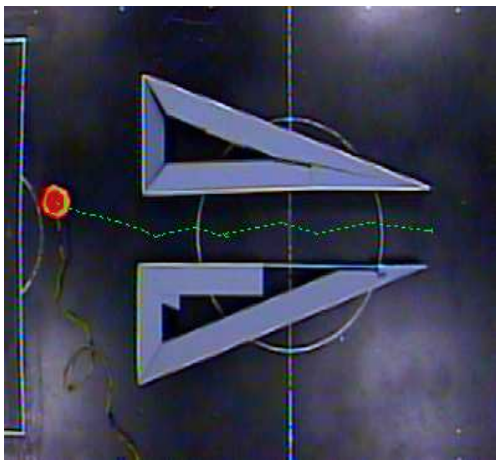
Furthermore, due to the symmetry in the first two cascaded levels of the controller, only 34 rules need to be designed (4.7% of 729).

Testing and observations

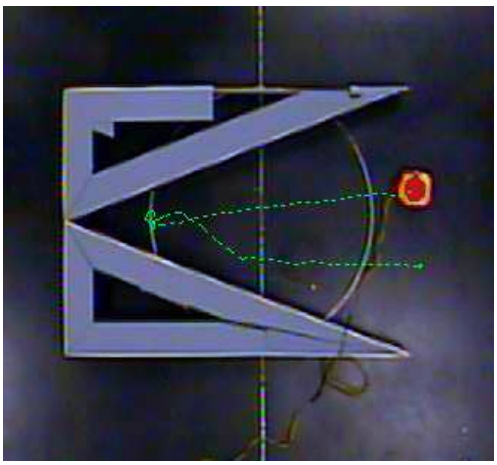
In the simulation environment, the robot moves smoothly around all the obstacle points (Figure 2.12). It also performed well in the real world environment, despite the disturbances in sensor reading and mechanical motion. The trajectories of robot navigation in the real world environment are depicted in Figure 2.16. Situations in (a), (b), (c) and (d) are identical to those at the points ③, ⑤, ② and ④ in the simulation set-up (Figure 2.12). Two new scenarios are constructed as in (e) and (f). In situation (e), while the triangular opening is narrow, the robot is able to turn back. As long as the opening is wide enough for the robot, as in situation (f), but not much wider than in situation (e), the robot could find its way through.



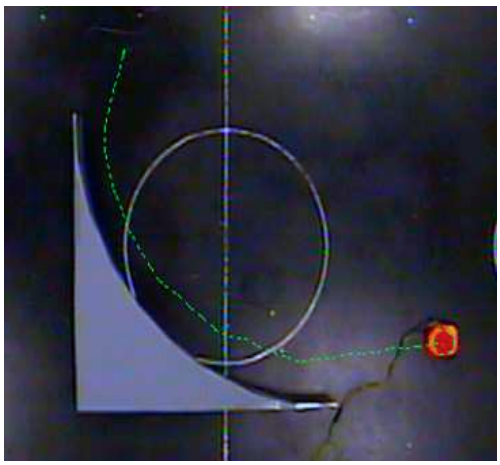
(a) Rectangular dead end



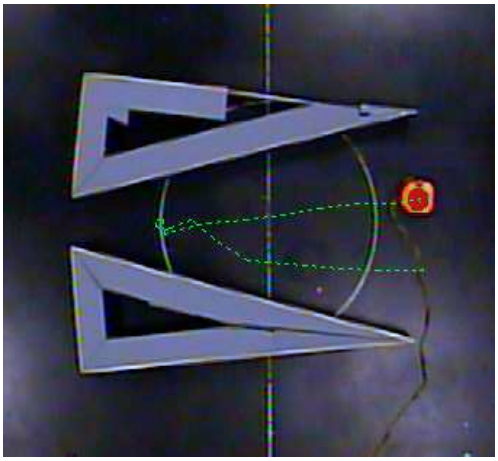
(b) Rectangular opening



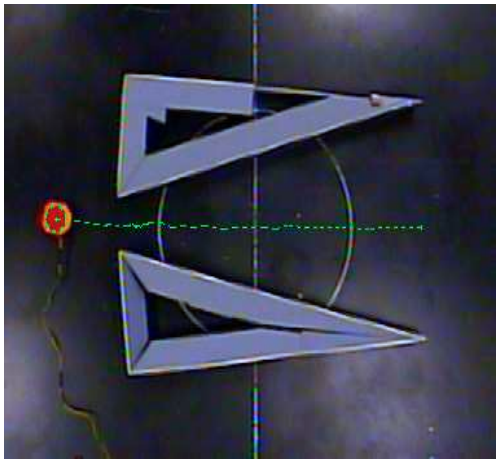
(c) Triangular dead end



(d) Round wall



(e) Triangular narrow opening



(f) Triangular wide opening

Figure 2.16: Robot trajectories in real world environment

2.6.5 Conclusion

In Section 2.6.3, a four-sensor input fuzzy logic controller is presented. With the cascaded-controller approach, the number of rules for the traditionally designed fuzzy logic controller is reduced to almost one-third. Proper planning of rules at the lower level of the cascaded controller is necessary to ensure proper behavior generation at the higher level.

In Section 2.6.4, a six-sensor input controller is discussed. The development of this controller highlighted the effect of following a symmetrical structure in cascaded flow. The controller enables the robot to navigate smoothly in environments with different kinds of obstacles.

It is found that the manual approach in the development of fuzzy logic controller has some disadvantages, which is addressed later in this thesis. In the later part of the thesis, application of evolutionary algorithms to the development and fine tuning of the rule sets, as well as the membership functions, are discussed in detail.

Chapter 3

Genetic Algorithms

3.1 Introduction to Genetic Algorithms

3.1.1 Evolutionary algorithms and search types

Evolutionary algorithm is an interdisciplinary research field which has its roots and application domains in biology, artificial intelligence, numerical optimization, and decision support systems in almost any engineering discipline. It describes computer-based techniques which perform the computation by simulating different aspects of evolution. A variety of techniques are classified as evolutionary algorithms, including:

- Genetic algorithms
- Genetic programming
- Evolutionary programming
- Evolution strategies

The common concept of these algorithms are based on the Darwin's principle of "survival of the fittest". The natural biological evolution is simulated via the processes of *selection*, *mutation* and *reproduction*. More precisely, evolutionary algorithms maintain a population of individuals. The population evolves according to

rules of selection and other operators, which are referred to as "genetic operators", such as recombination and mutation. Each individual in the population receives a measure of its fitness in the environment. Reproduction favors the individuals with high fitness, thus exploiting the available fitness information. Recombination and mutation perturb the individuals, providing general heuristics for exploration. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

As one class of the major evolutionary algorithms, genetic algorithms are the essential tools used in this work, and is introduced in detail in this chapter.

The current literature identifies three main types of search methods [15]:

- Calculus-based
- Enumerative
- Random

Calculus-based methods are commonly used for searching relatively smooth search spaces. Calculus-based methods rely on searching by solving mathematical formulas. Such methods can be very efficient and useful in single-peak domains, because they employ the notion of hill climbing by seeking the local best in the steepest permissible direction. While calculus-based methods have been improved and extended, they show lack of robustness. The optima they seek are the best in the neighborhood of a current point, and they depend on the existence of derivatives. The second problem can be overcome for certain applications using different techniques but implementations of these techniques are difficult and computationally expensive.

Enumerative methods are search algorithms that start checking objective function values at every point in a finite or bounded infinite searching space, taking one at a time. Although the algorithm seems attractive due to it's simplicity, it lacks efficiency in some practical applications. Many real world search spaces are

simply too large to search in this way. Even some of the best enumerative schemes like the dynamic programming break down on problems of moderate size [15].

Random search algorithms walk along randomly chosen points within the search space and simply choose the point with the best objective function value. They have achieved increasing popularity as the limitations of calculus-based and enumerative techniques became obvious. Monte Carlo methods are well-known random search algorithms. However, in the long run, random searches can be expected to do no better than enumerative schemes.

Random search methods should be separated from randomized techniques. Randomized search does not necessarily imply directionless search. Genetic algorithm is an example of a search procedure that uses random choice as a tool to guide a highly exploitative search through the coding of a parameter space.

3.1.2 What are genetic algorithms?

Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. GAs were inspired by the studies of cellular automata, which is a collection of "colored" cells evolving according to a rule set based on neighboring cells' states. In 1975, John Holland at the University of Michigan proposed and analyzed GAs in his book – *Adaptation in Natural and Artificial Systems* [16], which is generally acknowledged as the beginning of the research in this field.

GAs are modeled on the natural evolution mechanism using the following foundations:

- Individuals in a population compete for resources and mates.
- Those individuals most successful in each "competition" produce more offsprings than those individuals performing poorly.
- Genes from "good" individuals propagate throughout the population so that

two good parents sometimes produce offsprings which are better than either parent.

- Each successive generation becomes more suited to their environment.

A population of individuals are generated in a random or heuristic way. Each individual is represented by a finite string of symbols encoding a possible solution in a given problem space. The individuals then go through generations of evolution which mimic the biological evolutionary theory. In each generation, multiple individuals from current population are selected, usually in proportion to their fitness, to enter the next generation. The fitness is a numerical value returned by a objective function, to evaluate how good the individuals are. The selected individuals may also undergo the mutation or recombination to form a new population, which goes on with the next iteration of the algorithm.

GAs differ from the normal optimization and search procedures in four ways:

- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs use objective function information, not derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

In GAs, the finite-length string representing the solution of problem is analogous to the chromosome in biological systems. Taken from the some finite-length alphabet, the characters, features or detectors in the string are analogous to genes. Each feature takes on different values (alleles) and may be located at different positions (loci). The total package of strings is called a structure or population (or, genotype in biological systems). The correspondence between the biological and GA terminologies is provided in Table 3.1.

Biological	Genetic Algorithm
chromosome	string
gene	feature, character or detector
allele	feature value
locus	string position
genotype	structure, or population
phenotype	parameter set, alternative solution, a decoded structure

Table 3.1: Comparison of biological and GA terminologies

3.2 Structure of a Simple Genetic Algorithm

In this section, the structure and techniques of a simple genetic algorithm (SGA) is explained with a typical pseudo code. Each item of this code is examined and demonstrated through simple examples.

3.2.1 The pseudo code

A pseudo code provides an abstract view of an algorithm. The pseudo code for a standard genetic algorithm is shown in (Figure 3.1).

$i = 0$	set generation number to zero
init-population P_0	initialize a usually random population of individuals
evaluate P_0	evaluate fitness of all initial individuals of population
while (not done) do	test for termination criterion (time, fitness, etc.)
begin	
$i = i + 1$	increase the generation number
select P_i from P_{i-1}	select a sub-population for offspring reproduction
recombine P_i	recombine the genes of selected parents
mutate P_i	perturb the mated population stochastically
evaluate P_i	evaluate its new fitness
end	

Figure 3.1: Pseudo code of a standard genetic algorithm

In following sections, the mechanisms associated with each item of the pseudo

No.	String	(x, y)
1.	100001	(x=4, y=1)
2.	001100	(x=1, y=4)
3.	110010	(x=6, y=2)
4.	000100	(x=0, y=4)

Table 3.2: Example of an initial population

code are explained with examples to illustrate how a genetic algorithm works.

3.2.2 Initial population

A genetic algorithm needs to start with a population of strings, or so called individuals. The individuals of the initial population are usually randomly generated. If domain knowledge of the problem is available, some advanced heuristic approaches can be applied in the population initialization.

A simple problem is considered to demonstrate the operation of a genetic algorithm. The problem is to find the maximum of the function $u(x, y) = (x - 7)^2 + (y - 3)^2$, where both x and y lie in an integer interval [0,7].

At first, the binary coding method is selected to encode the solution of the problem. Six binary bits are used to represent the x and the y values, three successive bits for each of variables. For example, 011101 means: $x = 011_b = 3$ and $y = 101_b = 5$. Table 3.2 shows a possible randomly generated population of four six-bit individuals.

3.2.3 Evaluation

In GAs, it is necessary to distinguish between good and bad individuals, and to tell how good or bad they are. For this purpose, every individual of the newly generated population must be evaluated according to the objective function. This is done by mapping the objective function to a "fitness function" which produces

No.	String	(x, y)	Fitness
1.	100001	(4, 1)	13
2.	001100	(1, 4)	37
3.	110010	(6, 2)	2
4.	000100	(0, 4)	50

Table 3.3: Evaluation of the initial population

a non-negative figure of merit. The mapping is usually done as the following [15]:

- when the objective is to maximize a utility or profit function $u(x)$, the problem of negative $u(x)$ values must be handled by transforming the fitness function $f(x)$ as follows: $f(x) = u(x) + C_{min}$ when $u(x) + C_{min} > 0$, $f(x) = 0$ otherwise.

- when the objective is to minimize a cost function $g(x)$, it is necessary to transform the minimization problem to a maximization problem and assure that the measure is non-negative by using the following cost-to-fitness transformation:

  $f(x) = C_{max} - g(x)$ when $C_{max} - g(x) > 0$, $f(x) = 0$ otherwise.

C_{min} or C_{max} may be chosen as an input coefficient, as the absolute value of the smallest u-value or the largest g-value in the current or last k generations. They can also be functions of the population variance.

As to the example problem under consideration, of which the maximum is to be found, C_{min} can be set to zero because the objective function $u(x, y)$ will never be negative. Thus, the fitness function $f(x, y)$ is equal to $u(x, y)$. The evaluation of the four initial individuals is shown in Table 3.3:

3.2.4 Selection

The selection operator in GA is to decide which individuals should be chosen to generate the new population for the following generation. According to the "survival of the fittest" principle, individuals with higher fitness values should have priority to pass on their genes to the following generation. For example, with Holland's original fitness-proportionate selection, an individual A which is twice as fit as an individual B would be expected to appear twice as much in the next generation. Note that the GA usually does not select individuals directly by their ranks in the population, thus the best individual is not guaranteed to be selected. There are some solutions such as elitist strategy to handle this issue.

The fitness-proportionate selection is one of the simplest selection scheme and is widely used. To implement fitness-proportionate selection, a biased roulette wheel is created, in which each individual of the current population obtain a roulette wheel slot sized in proportion to the individuals fitness. For this reason, such a method is also called roulette wheel selection. n new individuals are selected by simply spinning the weighted roulette wheel for n times [15]. By dividing an individual's fitness value with the average fitness values of all, the expected count of this individual in the next generation can be calculated. For the example problem, the average fitness functions of all individuals is 25.5. The expected copies of individual No. 1 in the next generation is $13/25.5 = 0.51$. Table 3.4 shows the expected counts of the four individuals, as well as the normalized fitness values, which are equal to the percentages of the individuals' fitness values in the sum of all fitness values. The normalized fitness indicates the chance of an individual to be selected in a random spinning.

In the algorithm for realizing the roulette wheel selection, the i-th individual is at first assigned a number $S_i = \sum_{j=1}^{i} fitness_i$, which is the sum of all fitness values from individual No. 1 to individual No. i. An integer is randomly and uniformly chosen between 0 and the sum of the fitness values of all. The first individual whose S_i is equal to or greater than this random integer is selected. The S_i values are shown in Table 3.4. For example, suppose the randomly chosen number is 53, then

No.	String	(x, y)	Fitness	Normalized	S_i	Expected count	Actual
1.	100001	(4 ,1)	13	12.7%	13	0.51	1
2.	001100	(1 ,4)	37	36.3%	50	1.45	1
3.	110010	(6, 2)	2	02.0%	52	0.08	0
4.	000100	(0, 4)	50	49.0%	102	1.96	2

Table 3.4: Results of reproduction

individual No. 4 is selected because S_4 is the first value which succeeds 53. This routine is repeated until enough individuals are selected. For the example problem, four individuals are selected (Table 3.4).

3.2.5 Recombination

Following the selection operation, the recombination operation is performed upon the selected individuals to generate new individuals for the following generation pool. Most genetic algorithms have a single tweak-able probability of crossover P_c, typically lying in a rang of 0.6 and 1.0, which represents the probability that two selected individuals generate new offsprings. A random number between 0 and 1 is generated, and if it falls under the crossover probability, the recombination operation is executed; otherwise, the individuals are propagated to the next generation unchanged. The most commonly performed recombination operator is the crossover. Quite a lot of different crossover operators have been developed to handle different problems. At this point, the simple one-point crossover is discussed as example. This operator randomly and uniformly selects an integer k from the range of $[1, l-1]$. Two new strings are created by swapping all the characters between positions k and l (Figure 3.2).

Before crossover	After crossover
A B C D | E F G H	A B C D | M N O P
I J K L | M N O P	I J K L | E F G H

Figure 3.2: The one-point crossover operator

The idea beneath the crossover operation is that: by recombining portions of good individuals, it is likely to create even better individuals. The role of the crossover operator is to lead the evolutionary process to move toward "promising" regions of the search space. The crossover is the prime distinguishing factor of the genetic algorithm from other optimization algorithms.

3.2.6 Mutation

Besides the crossover operator, mutation is another operation in GA which creates new individuals. Typical genetic algorithms have a fixed, very small probability of mutation (P_m) of perhaps 0.01 or less. A random number between 0 and 1 is generated; if it falls under the P_m, the new individual's chromosome is randomly mutated in some way. Usually, the mutating operator simply tosses a biased coin with probability P_m at each bit and, according to that result, alter the bit from 1 into 0 or vice versa (Figure 3.3).

The importance of mutation in genetic algorithm is still a matter of debate. Some believe that mutation plays a secondary role in the simple genetic algorithm. The effect of mutation is to reintroduce divergence into a converging population. In the latter stages of a genetic algorithm run, the algorithm may be converging upon a local maximum. Mutating some chromosomes may randomly explore new points in searching space and enable the evolution to find a way past the local maximum. The biological inspiration behind this operator is the fact that a chance mutation in a natural chromosome can lead to the development of desirable traits. Those traits provide the individual displaying certain characteristics an advantage over its competitors [106].

Before mutation	After mutation
1 0 **1** 1 0 0	1 0 **0** 1 0 0

Figure 3.3: The mutation operator

In the example case, the third bit from the left of individual No. 1 is changed by mutation and the new population is shown in Table 3.5, as well as the new

fitness values.

No.	Selected parents	After crossover	After mutation	New fitness
1.	10|0001	101100	10**0**100	10
2.	00|1100	000001	000001	53
3.	00010|0	000100	000100	50
4.	00010|0	000100	000100	50

Table 3.5: New population and fitness after crossover and mutation

As shown in Table 3.5, a new string with high fitness has appeared. The sum of the fitness values has increased from 102 to 163 and the average has increased from 25.5 to 40.8 just in one generation. Following the selection, crossover and mutation, individuals No. 1 and 2 of the initial population are selected once (average fitness), individual No. 3 is not selected (low fitness) and individual No. 4 is selected twice (high fitness). Crossover generates the high-fitness string 000001 (string No. 2) but also the low-fitness string 101100 (string No. 1) in which a mutation takes place which, in this case, increases the fitness.

The simple genetic algorithm is a powerful tool which is able to converge rapidly to an optimum of many different objective functions. The example problem is a two-variable function, but a function of more variables is easy to implement. The user has to create an encoding scheme, a fitness function and implement these into the genetic algorithm, whose mechanisms are easy to transfer to a computer program.

3.3 Theoretical Background

The theoretical basis of genetic algorithms relies on the concept of schema (pl. schemata). A schema is a template describing a subset of chromosomes with similarities at certain string positions. If A denotes the alphabet of symbols in chromosomes, schemata are strings of symbols defined over alphabet $\{A \cup \{*\}\}$. The extra-symbol "$*$" is interpreted as a wildcard, which indicates the occupied loci are

undefined and accept any symbol of A. The more "$*$" symbols a schema contains the less specific it becomes, i.e. the more strings it can describe.

A chromosome is said to be an instance of a schema if it matches the defined positions where the symbols are already determined from A. For example:

> The string 10011010 matches the schemata 1******* and **011*** among others, but does not match *1*11*** because they differ in the second gene (the first defined gene in the schema).

A schema can also be interpreted as a hyperplane in a n-dimensional space representing a set of solutions with common properties. Obviously, the number of solutions belonging to the schema H depends on the number of defined positions in it, which is defined as the *order* $o(H)$ of schema H. The smaller the order is, the more instances belong to the schema. Another related concept is the *defining-length* $\delta(H)$ of a schema, which represents the distance between the first and the last defined positions in the schema. The defining length determines the likelihood of an instance of the schema being disrupted by crossover operation. Examples are provided in Table 3.6.

Schema H	$o(H)$	$\delta(H)$
$* * * *\ 0\ 0\ 1\ *\ 1\ 1\ 0$	6	6
$* *\ 0\ * *\ 0\ 1\ * *\ 0\ *$	4	7
$0\ 1\ 1\ 0\ 1\ * *\ 0\ 0\ 1\ *$	8	9

Table 3.6: Examples of schemata

The idea of a schema provides a powerful and compact way to study well-defined similarities among finite length strings over a finite alphabet. Usually, by evaluating the fitness of any one string (whether it be large or small), one might expect to obtain information about other strings which have a structure similar to it. A string of length l built from alphabet of cardinality (number of alphabet characters) n is the instance of n^l different schemata. Every time the fitness of a single chromosome is accessed, all of the schemata to which it belongs also undergo

a trial. This phenomenon is known as *implicit parallelism* [16], which is a major part of the explanation of the power of the genetic algorithm.

According to the implicit parallelism, by operations on chromosomes, GA is performing a large number of trials in the searching space of schemata in parallel. As a searching algorithm, GA needs to take care of the tradeoff between exploiting the current best schemata and exploring new schemata by optimal allocation of the number of trials. It turns out that the optimal strategy is to allocate an exponentially increasing number of trials to those schemata which seems to be the best [16]. Due to the mechanics of fitness based selection, GAs do allocate the trials an the optimal manner.

The general schema growth equation for proportional selection is formulated as,

$$m(H, t+1) = m(H, t) \cdot \frac{f(H)}{\bar{f}}, \tag{3.1}$$

where $m(H, t+1)$ is the expected number of instances of schema H in generation $t+1$ and $m(H, t)$ is the number of instances in generation t. $f(H)$ denotes the average schema fitness of the instances in generation t which belong to schema H while the $\bar{f}$ is the average fitness of all population. Under the assumption that $f(H)$ is above the average fitness, Equation 3.1 can be reformulated as,

$$m(H, t+1) = m(H, t) \cdot \frac{\bar{f} + c \cdot \bar{f}}{\bar{f}} = m(H, t) \cdot (1 + c), \tag{3.2}$$

and which finally leads to,

$$m(H, t+1) = m(H, 0) \cdot (1 + c)^t, \tag{3.3}$$

with a constant value of $c > 0$. Equation 3.3 clearly indicates that proportional selection allocates exponentially increasing (decreasing) number of trials to above (below) average schemata.

$m(H, t+1)$ is further affected by the crossover operator and mutation operations. A schema is destroyed by one-point crossover when the crossover point falling within the schema's defining length. As a result, the probability that the schema H survives a crossover with the probability P_c is,

$$P_{surv-cross}(H) = 1 - P_c \cdot \frac{\delta(H)}{l-1}, \tag{3.4}$$

where l is the length of the chromosome. Similarly, a schema is disrupted by mutation if the mutation takes place in the schema's defined position. The survival probability for mutation is given by,

$$P_{surv-mute}(H) = (1 - P_m)^{o(H)} \approx 1 - P_m \cdot o(H), for P_m \ll 1. \tag{3.5}$$

Combining Equations 3.1, 3.4 and 3.5, an estimation of the number of instances of a schema in the generation $t+1$ is obtained, which is summarized as the *schema theorem* [16]:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \cdot \left\{ 1 - \left(P_c \cdot \frac{\delta(H)}{l-1} \right) - P_m \cdot o(H) \right\}. \tag{3.6}$$

The basic statement of the schemata theorem is that short, low-order, above-average schemata (so-called building blocks) receive exponentially increasing trials in the following generations.

Schemata theorem is regarded as the milestone in the development of the theorems to describe the working of GAs. However, it is not general and its conclusions are of limited use. Considering the following,

- No positive (constructive) effects of the GA operations are considered, resulting in a lower-bound expectation of the schemata growth.
- Operates under the assumption that the solution must be an instance of fit schemata, which is not always true.
- Fails to provide any indication of the convergence rate towards the optimal or near optimal, or how good a solution is eventually found.

The shortcomings of schemata theorem have led to several modern approaches to the theorems associated with GA [107, 108].

3.4 Case Study: Genetic Algorithm for Fuzzy Logic Control of Mobile Robot

The adaptive capability, robust nature, and simple mechanics of genetic algorithms make them suitable for various kinds of problems in the science, engineering and business world. Furthermore, hybrid systems which combine GAs with other computation methods, such as fuzzy logic or neural network, seem to be promising and have attracted intense research interests. Studies have shown that GAs are effective at optimizing the rule base and membership functions of fuzzy logic controllers (FLCs) [74, 73, 109, 110, 111, 112]. With the great search power, GAs are up-and-coming tools for knowledge acquisition for complicated systems and enable us to establish a suboptimal fuzzy rule base, if not the global optimal one.

In this section, GA is applied to search for optimal decision-making rules for a fuzzy logic controller. The controller is designed for the mobile Khepera robots to perform obstacle avoidance task. The Khepera robots and the associated simulation software Webots have been introduced in Section 2.6.2. Different from the work in Section 2.6, all the eight infra-red sensors of Khepera robot are utilized. The GA is applied to generate and optimize the FLC rule base, which means the rule base is automatically designed. Simulation and experimental results show that the GA optimized FLC works well on Khepera robots. With the GA evolved fuzzy controller, the Khepera robot is able to navigate in unknown environments.

3.4.1 Fuzzy logic controller for Khepera robots

As introduced in Chapter 2, fuzzy logic can deal with uncertain and imprecise situations. Linguistic variables are used to represent the domain knowledge, with their membership values varying from 0 to 1. Basically, a fuzzy logic controller consists of four major components [113]: the fuzzification interface, the knowledge base (rule base), the inference engine and the defuzzification interface.

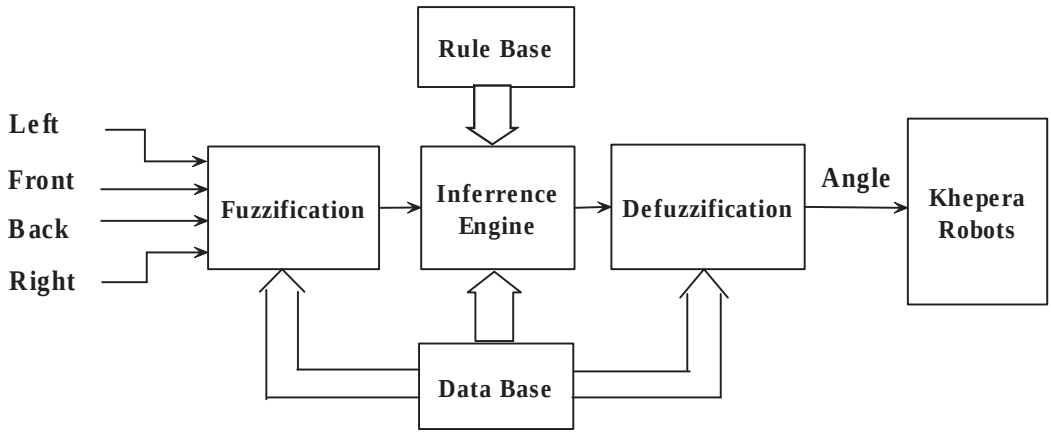


Figure 3.4: Structure of the fuzzy controlled Khepera robot system

The fuzzy logic controller for the Khepera robot is designed to control the moving direction of the robot. The FLC is also made up of the four major components (Figure 3.4). Furthermore, there is an auxiliary database providing environmental information and parameter setting data to the FLC.

The proximity values received from the eight sensors are taken to generate the inputs to the FLC. These eight sensors, marked from 0 to 7, are divided into four groups to detect the obstacles in four directions. Each sensor's input value ranges from 0 to 1024. For instance, the sum of the values from sensors 0 (IR_0) and 1 (IR_1) is designated as the proximity sensor reading from the left side. The grouping of the sensors is as follows:

$$\text{Left} = IR_0 + IR_1 \qquad \text{Front} = IR_2 + IR_3$$
$$\text{Right} = IR_4 + IR_5 \qquad \text{Back} = IR_6 + IR_7$$

The "Left", "Front", "Right" and "Back" variables resulted from grouping are the inputs to the FLC. A bigger value indicates a closer object in that direction. Three linguistic values, "Large", "Medium" and "Small", are used to represent each input. A typical fuzzy rule often consists of antecedent (or premise), consequent (or conclusion) and fuzzy relations. The input linguistic variables form the *antecedents* of a fuzzy rule. Four linguistic values, "Forward", "Small Turn", "Large Turn" and "Backward", are used to represent the output in the *consequent* of a rule. The logic

operation is "AND". A typical fuzzy rule is:

If Left is Small AND Front is Large AND Right is Small AND Back is Small, the action is Backward.

The *consequent* part indicates the angle by which the robot should turn to. This angle, with the scope of 0 to 180 degree, is the final output of the FLC. From the combination of three linguistic values for four inputs, there are $3^4 = 81$ input states. Thus, the FLC rule base for Khepera robot consists of 81 rules.

Due to their smoothness and concise notation, the popular bell-shaped membership functions (Figure 3.5) are used to fuzzify the inputs into linguistic variables and defuzzify the *consequent* to crisp output. The functions are realized by

$$\mu_{Li}(x, a_i, b_i, c_i) = \frac{1}{1 + [[(x - c_i)/a_i]^2]^{b_i}}, \tag{3.7}$$

where a_i decides the spread and b_i decides the flex of the two sides of the bell-shaped functions centered at c_i. Altogether, the parameter set a_i, b_i, c_i describes a bell-shaped membership function. The sets for different linguistic variables form the tunable parameters of the controller.

3.4.2 Genetic coding method and operators

The kernel of the FLC is its rule base, a set of linguistic control rules which are used to simulate human thinking. The rule base plays a key role in FLC and determines the control system's performance under different situations. The conventional fuzzy control rules are constructed on the basis of experts' knowledge and experience. However, the experts' knowledge about the system is more or less limited, and sometimes is even not available at all. On the other hand, GAs operate without any knowledge of the task domain and utilize only the fitness of the evaluated individuals, that make them promising tools for the fuzzy rule base development and/or optimization.

All the three parts of a fuzzy rule: antecedent, consequent and fuzzy relation, can be evolved by GA. In this case, there are a total of 81 input states, for which

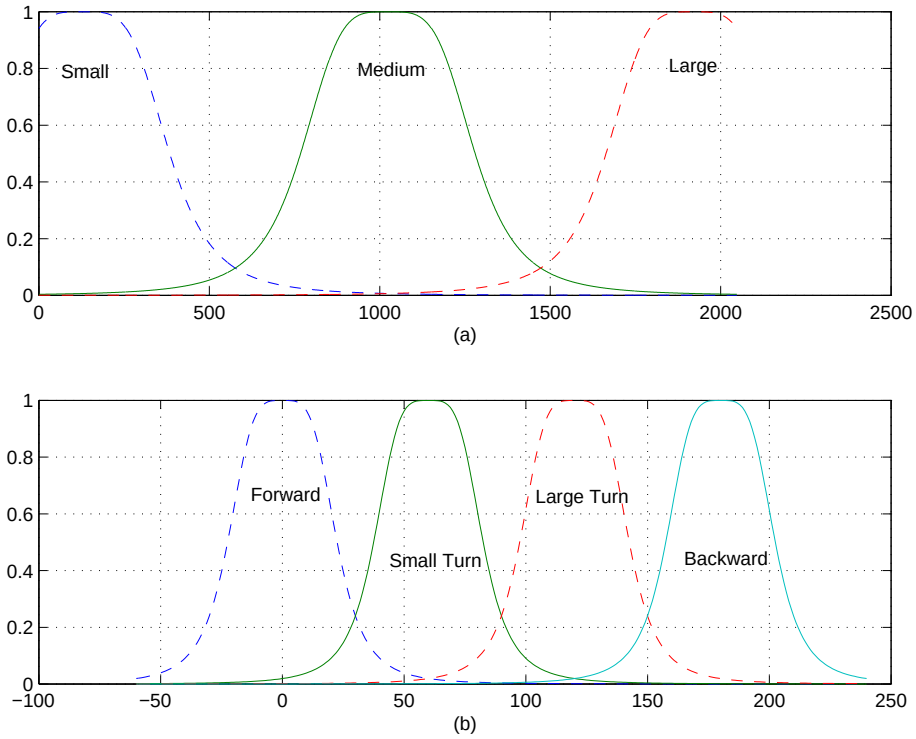


Figure 3.5: Membership functions for inputs and output

the possible consequent parts are evolved by GA. Integer coding method is used. The four consequent linguistic values: "Forward", "Small Turn", "Large Turn" and "Backward", are represented by integer numbers from 1 to 4. For the 81 rules in one rule table, 81 integer numbers are combined into one chromosome as an individual. Each integer number in the chromosome stands for one consequent part, thus one fuzzy rule. In other words, each fuzzy rule has its associated consequent part on a fixed position of the chromosome. The entire integer string represents the whole rule base.

Furthermore, the rule table has a kind of a symmetry, resulting in part of the rules having the same consequent parts. For example, if the two rules have antecedents as follows:

Left is Large AND Front is Large AND Right is Small AND Back is Small...
Left is Small AND Front is Large AND Right is Large AND Back is Small...,

they should have the same consequent parts. Then the length of the chromosome

can be shortened to 54 and that simplifies the computation a lot.

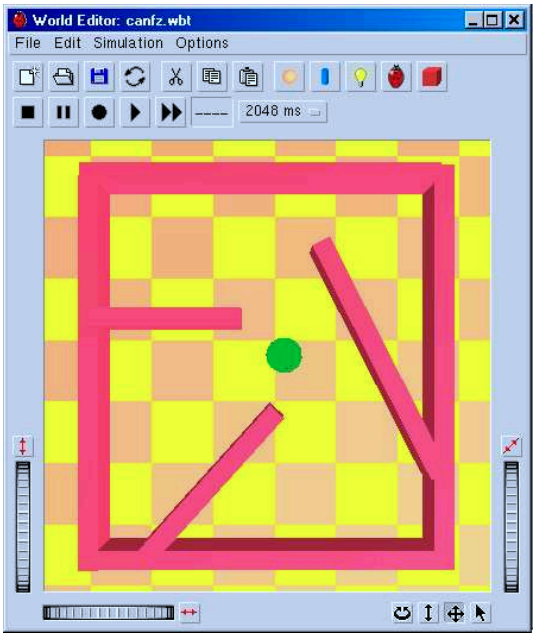
Another issue worth mention is that the robot should always move ahead when no obstacle is around. Thus the consequent of the rule for this situation can be fixed as "Forward", which means turn by 0 degree. This prevents the robot from spinning around its axis.

In the beginning of evolution, the fitness of the chromosome is initialized to zero. The robot takes n steps to evaluated the performance of FLC. In each step, the robot processes the sensor readings, decides the direction of motion and moves along that direction for a certain distance. The fitness is decreased by unity if obstacles are detected, otherwise the fitness is increased by unity. If the robot keeps approaching an obstacle and the proximity value exceeds a predefined limit, the fitness is decreased by ten units as punishment. Furthermore, there is a penalty function associated to the angle turned by. The fitness is decreased by a value which is k times the degree of the angle. k is selected between 0.0055 and 0.0111. This penalty function in fact encourages the robot to act more efficiently, i.e., to turn by the necessary angle to avoid obstacles.

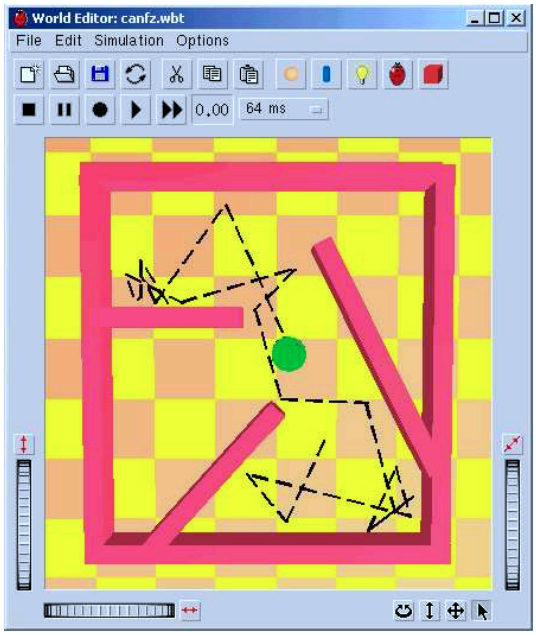
In this case, the evolution operations on the chromosome are the standard crossover and mutation. In each generation, individuals are sorted by their fitness. Those with higher fitness values are selected as elite candidates to generate offsprings. The evolution stops when the best individual has not made any improvement for a certain period of time.

3.4.3 Simulation, experimental results and discussion

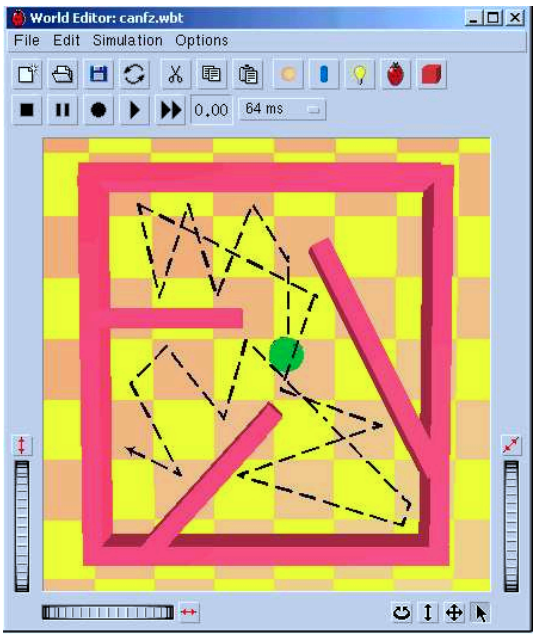
The Webots package is used in the simulation phase. Programs are coded using MS C/C++. The initial position and orientation of the robot are fixed in the training world (Figure 3.6(a)). The robot is set to run 400 steps to evaluate the fitness of each individual. In each step, the robot decides whether to turn or not and, moves along the decided direction for 64 ms. The forward speed is set to 12 units (about 96 mm/s). The angle to turn is decided by the FLC with the rule base decoded



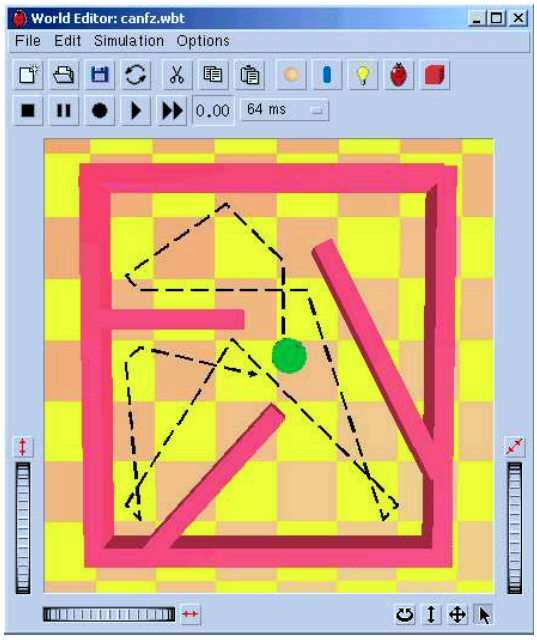
(a) Training world.



(b) Trajectory Before Evolution.

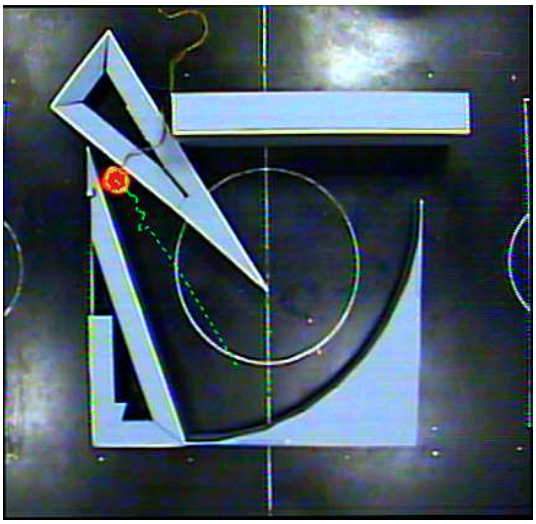


(c) Trajectory after 50 generations.

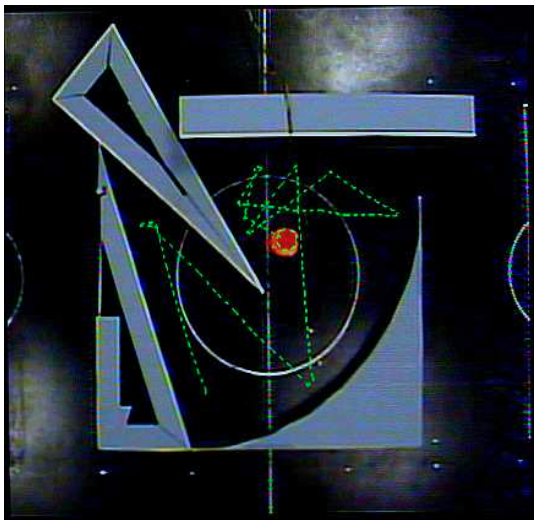


(d) Trajectory After Evolution.

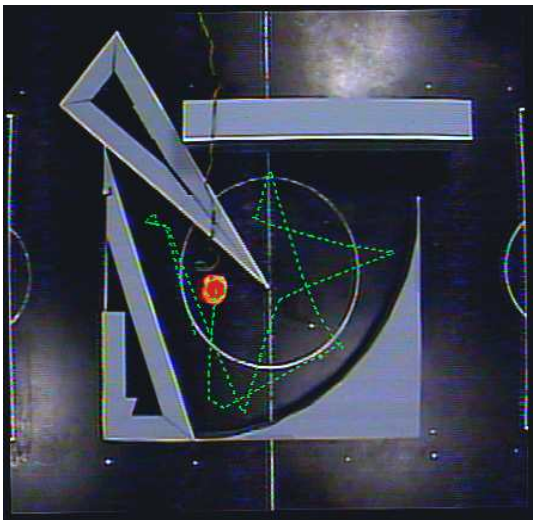
Figure 3.6: Robot's trajectories in simulation setup



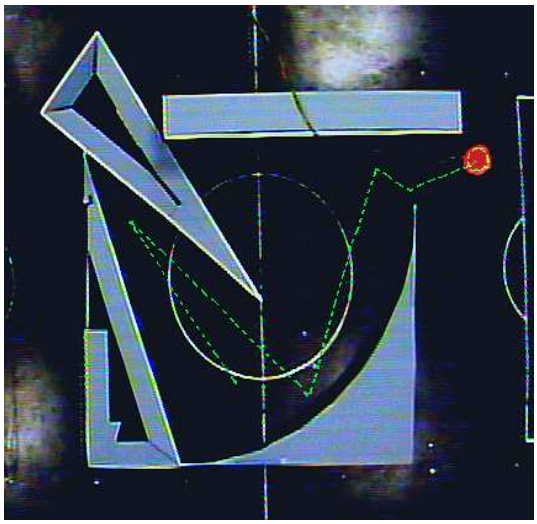
(a) Before evolution



(b) At an early stage of evolution



(c) At the late stage of evolution



(d) After evolution

Figure 3.7: Robot's trajectories in real world experimentations

from the individual under evaluation. To be more realistic, a random white noise, which is about 10% of the reading is added to the sensor measurements.

A small population size of 60 is used for the rather simple "obstacle avoidance" task. The elite population size is 20. Based on the elites, 40 offsprings are generated by cross-over and 20 of them are mutated. The rather high mutation proportion helps to avoid the "premature convergence" which often happens with a small population size. The algorithm is stopped when no improvement is observed with the best individual in 20 successive generations.

At the beginning, the individuals are generated by arbitrarily assigning the four consequent linguistic values to each rule. As a result, the robot wandered around and crashed often. The robot also tends to linger at the corners of the obstacles for a long time before moving out. As the evolution process carried on, the robot could avoid obstacles "effectively" and crashed seldom. At the end of the evolution, the best individual enabled the robot to avoid obstacles "efficiently" and escape from trap points quickly. The robot trajectories are shown and compared in Figures 3.6(b), 3.6(c), and 3.6(d).

The Khepera robot with the rule bases obtained at different evolution stages in simulation is tested in real environment (Figure 3.7) for experimental verification. The experimental setting is much different from the training world and is totally unknown to the robot. The robot is initially located near the lower-left corner of the setting, facing towards the dead end in the upper-left corner. At first, with a randomly generated rule base, the robot is trapped in the dead end (Figure 3.7(a)). Then with the best rule base in the early stage of evolution, the robot is able to move around, although its behaviors showed some kind of random motions (Figure 3.7(b)). With a further evolved rule base, the robot can move more efficiently and swiftly (Figure 3.7(c)). At the end of evolution, the robot with the optimized rule base can even navigate out of the surrounding obstacles (Figure 3.7(d)). The swiftness in avoiding obstacles provided the robot a better chance to reach the exit.

It is noticed that the robot could not move around so elegantly as in the simulation. Sometimes it just turned by an angle which is more than necessary. The drop

in performance is due to the disturbances coming from the real world environment. For instance, the environment's color and lighting condition can disturb the sensor readings and, the inertia and friction can affect the robot's mechanic motion. However, the FLC's performance is not tampered that much by the disturbances, depicting some grade of robustness and noise tolerance.

The proposed genetic algorithm has constructed an optimized rule base for the FLC of Khepera robot. The FLC enables the robot to perform well in "obstacle avoidance" task. The robot could move around the experimental world and never crashed into or trapped by the obstacles.

Since the "obstacle avoidance" is a fairly simple task, the FLC designed here is rather simple too. The following chapters will focus on more complex problems and further combination of GA with Fuzzy logic to exploit the power of evolutionary algorithms. In the following chapters, a fuzzy behavioral based robot soccer system is developed, which provides a much more complex and non-trivial problem.

Chapter 4

The Robot Soccer System

The previous chapters have introduced fuzzy logic and genetic algorithms in detail. Several case studies have been described for illustration purpose. However, those case studies are trivial in the sense of complexity and significance in practical applications. The rest of this thesis discusses more complicated problems in a real world multiple robotic system, which is the robot soccer system (RSS). Since the robot soccer system serves as a platform for all the works in the rest of this thesis, it is beneficial to provide a thorough introduction to the same. Both the system's hardware architecture and mathematical model are analyzed in this chapter.

4.1 Robot Soccer Activities

Robot soccer based competitions and research activities have been growing steadily over the last two decades. Robot soccer system originates in the mid-90's, marked by the foundation of Robot World Cup Initiative (RobotCup) [114] and the Federation of International Robot Soccer Association (FIRA) [115]. Nowadays, there are many leagues of robot soccer championships being held under the auspices of FIRA and Robocup.

Since its birth, robot soccer has been an intriguing interdisciplinary field of research. The robot soccer system is an instance of multiple robotic system which

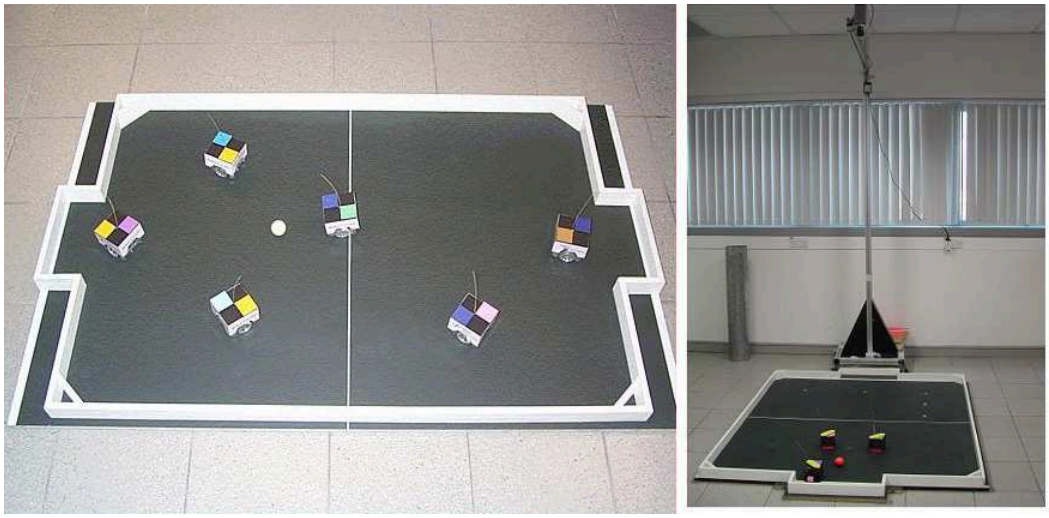


Figure 4.1: The robot soccer field

covers a large area of research themes like robotics, intelligence control, communication, computer technology, sensor technology, image processing, mechatronics and artificial life. The robot soccer based research aims to achieve an increased level of autonomy and collaboration between artificial agents through the medium of a very complex game soccer.

Robot soccer system is an excellent research platform for a number of reasons. Researchers have to deal with a number of real world problems, such as environment noise and robot failure. The competition element also works as a motivator for researchers.

Specifically, the robot soccer system discussed in this thesis belongs to small league Micro-Robot Soccer Tournament (MiroSot), which is a division of robot soccer games held by FIRA annually at international scale.

Under small league MiroSot rules, the robot soccer game can be described briefly as follows:

1. In the 3 vs 3 MiroSot game, six mobile robots are grouped as two soccer teams, trying to score against each other to win the match. Each team has one robot as the goalie.

2. The soccer field is black colored with dimensions $150cm \times 130cm$. The field

is bounded on all sides by a 5cm high wall to prevent the robots and the ball from tipping over the field (Figure 4.1).

3. An orange colored golf ball of the standard size acts as the soccer ball.

4. The robots are radio controlled and the size of each robot is limited to $7.5cm \times 7.5cm \times 7.5cm$, except for the height of the antenna.

5. There is a camera placed over the field at a height of 2m, which provides the host computer with the video image of the match field with the robots and the ball inside (Figure 4.1).

6. The robots must have their team color patch on top. The team color is either blue or yellow, as assigned by the organizers. The teams are allowed to use other color patches to distinguish between home robots, except for any color patch resembling the color of the opponent team color and the ball.

7. No human intervention is allowed during the game, except for stopping all the robots according to the referee's commands.

Similar to human soccer players, soccer robots should be agile, have good strategy and need to collaborate.

4.2 Robot Soccer System Architecture

The hardware setting of robot soccer consists of three parts: the vision system, the host computer and the robots (Figure 4.2). The vision system consists of a CCD camera, a frame capture card and drivers. The video camera is mounted above the robot soccer field and captures the overview of the playground. The vision system serves as the sensor feedback of the system. As the "brain" of the system, the host computer manipulates the vision data and controls the robots via radio frequency (RF) communication. The soccer robots move on the playground following the host computer's commands and transfer the team strategy into reality.

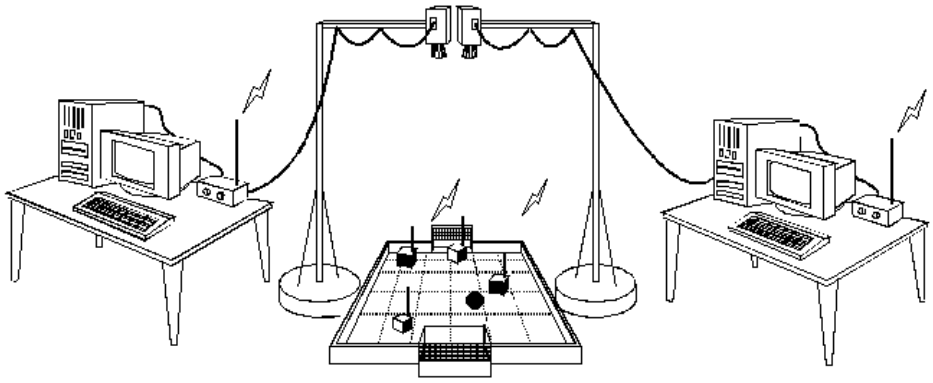


Figure 4.2: Hardware setting of a robot soccer system

The camera transmits analog video signals to the frame capture card inside the host computer for digitalization. The image of the entire playground is captured every 20ms and stored into the video buffer in RGB format. The vision algorithm running in the host computer is employed to identify the robots and the ball, according to their colors. The relative position data of each robot and the ball are also obtained from the color pixel data by the vision-process module. Based on the position data, the strategy module decides the robots' subsequent actions and compute the relative velocity settings of robots. The host computer outputs the velocity commands to a RF transceiver through the serial part. The transceiver in turn formats the commands into specific protocol signals and sends them to the soccer robot, whose motion system is formed by two direct current (DC) motors in a two-wheeled differential drive configuration. The control signals received by the robot are then converted into voltages, which produce the torques that provide angular velocities to the wheels and finally determine the posture and velocity of the robot. The complete procedure is summarized in Figure 4.3.

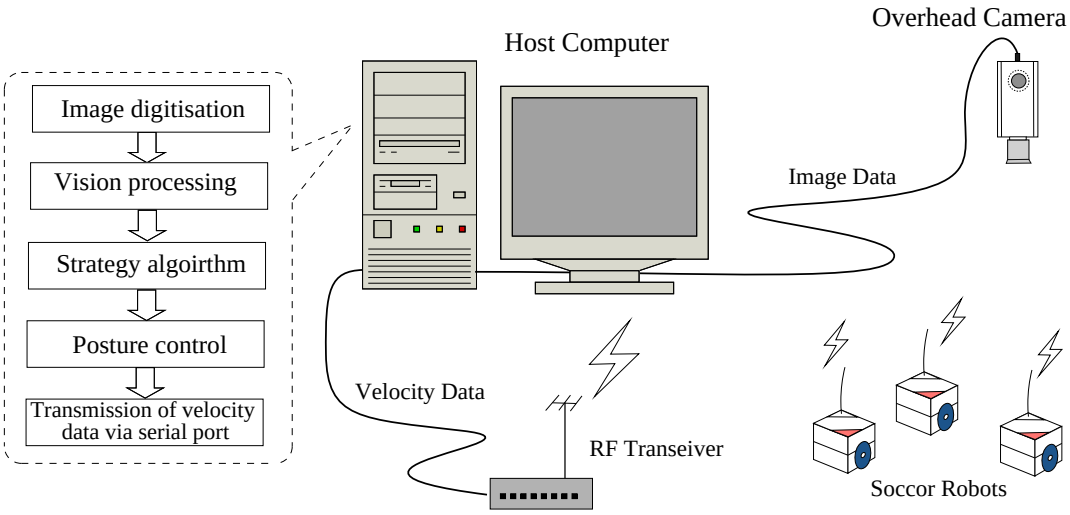


Figure 4.3: Overview of robot soccer system architecture

4.3 Soccer Robot Architecture

The soccer robot used in MiroSot is usually cube-shaped, equipped with two wheels (Figure 4.4). Each wheel is driven by a DC motor controlled by the pulse width modulator (PWM). The speed of the wheel is detected by an optical encoder. A 7.2v battery powers the robot. The wireless signal sent from host computer is received by a RF module, which is connected to the serial port of the on-board micro-controller. As the kernel of the robot, the micro-controller is integrated with central processor, memory and I/O port. The proportional-integral-derivative (PID) control algorithm is embedded into the micro-controller. The architecture of the soccer robot is depicted in Figure 4.5.

The micro-controller, the motor and the encoder make up of a real-time close loop control system. Upon receiving of the control signals from the host computer, the on-board PID controller of robot ensures that the robot achieves its desired velocity and consequently the desired position and orientation.

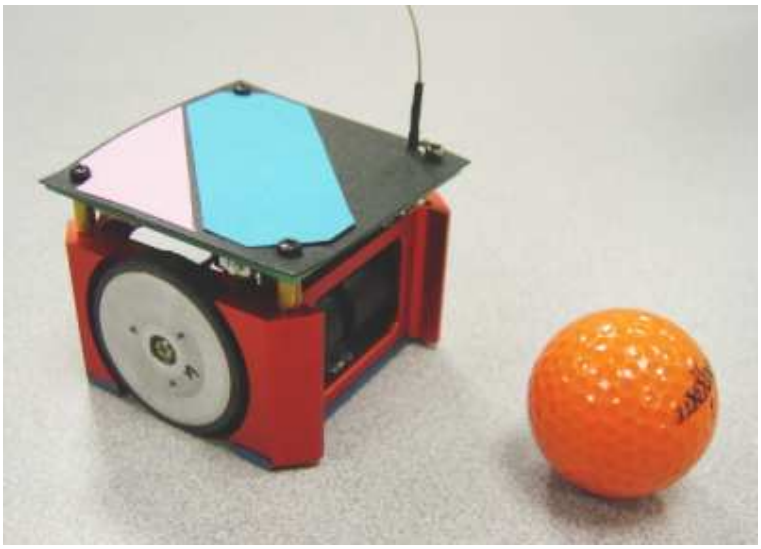


Figure 4.4: The soccer robot

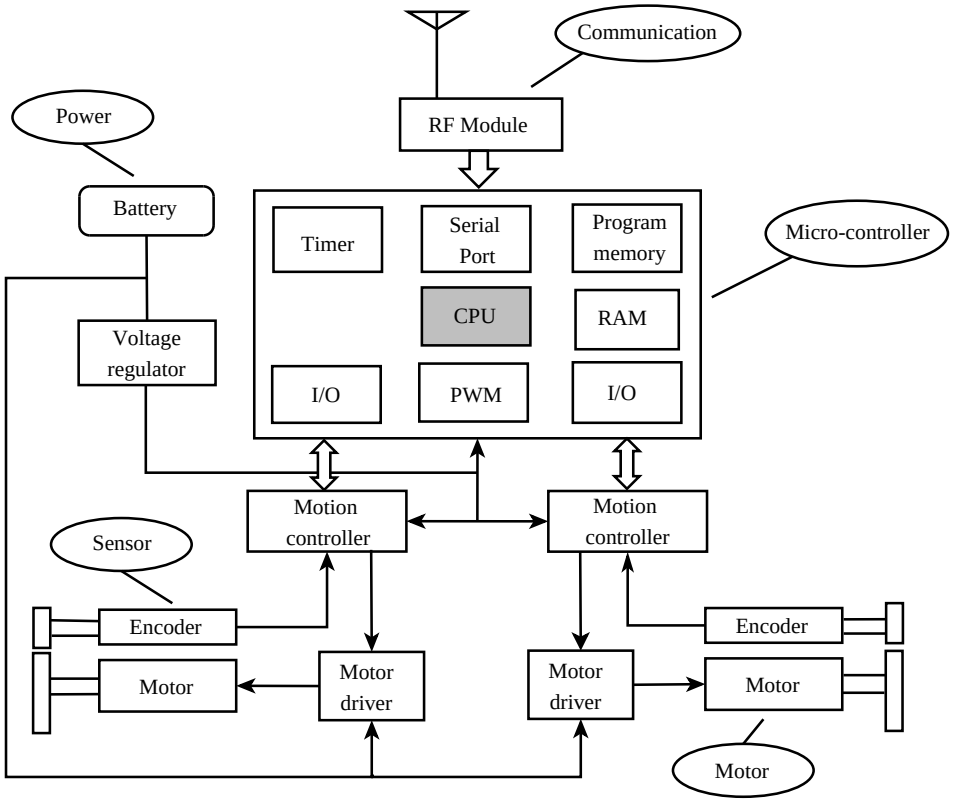


Figure 4.5: Soccer robot hardware structure

4.4 Mathematical Model of Soccer Robot

The motion of robot is controlled with two independent control signals for the left and right wheel motors which are decided by the host computer. Obviously, the relationship between the control signal and the resultant motion of robot is essential for the robot motion control. For this reason, a mathematical model is constructed for the better understanding of the kinematics of the soccer robot. Furthermore, the model is especially important for the development of the robot soccer system simulator which is vital for evolving the robot behaviors (Chapters 7 and 8).

The kinematic states of the robot are described in Cartesian Plane by two vectors: P (posture) and V (velocity):

- $P = [\ x \quad y \quad \theta\ ]^T$, where x, y, and θ refer to the robot's x and y coordinates and the robot's heading angle with respect to the positive X-axis (Figure 4.6(a)).
- $V = [\ v \quad \omega\ ]^T$, where v and ω are the translational and rotational velocities of the robot in the local coordinate system of the robot (Figure 4.6(b)). Combinations of v and ω result in curved paths of different turning radii.

The posture P and velocity V vectors are related by a Jacobian matrix $J(\theta)$ given by:

$$\dot{P} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \sin\theta & 0 \\ \cos\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} = J(\theta)V. \tag{4.1}$$

A non-holonomic constraint of the following form exists under such a configuration:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0. \tag{4.2}$$

The non-holonomic constraint in Equation (4.2) describes the inability of two-wheeled differential drive robots to move along a direction perpendicular to its current heading, provided that the wheels are non-slipping. The angular velocities

of the wheels are translated into linear motion through the traction of the wheel on the contact surface:

$$\mathbf{v} = \begin{bmatrix} v_L \\ v_R \end{bmatrix} = \mathbf{r} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}, \tag{4.3}$$

where v_L and v_R are the left and right wheel velocities, ω_L and ω_R are the left and right wheel rotational velocities, and r is the radius of the wheels. From the wheel velocities, the translational and rotational velocities of the soccer robot can be obtained:

$$\begin{aligned} \mathbf{V} &= \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{v_L + v_R}{2} \\ \frac{v_R - v_L}{L} \end{bmatrix} \\ &= r \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \mathbf{L} \cdot \mathbf{v}, \end{aligned} \tag{4.4}$$

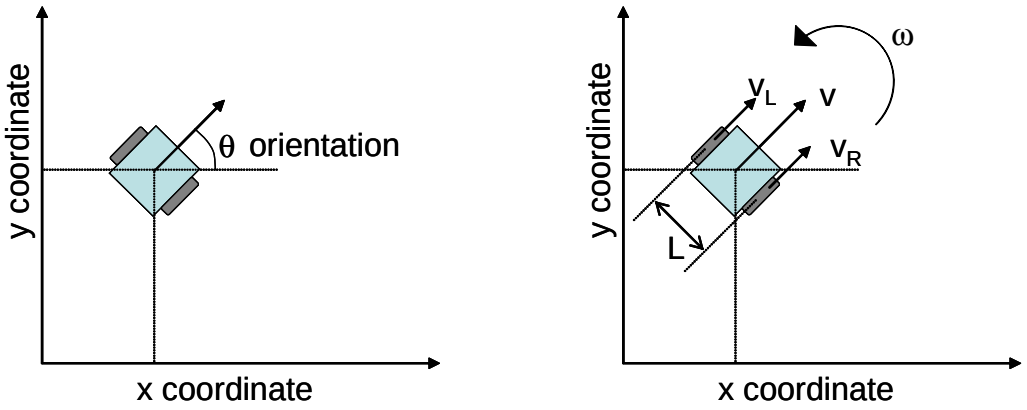
where L is the orthogonal distance between the two wheels. In most systems, the left and right wheel control signals are specified in terms of byte values, hence a relationship must be formed between these control signals and the resultant wheel velocities:

$$\mathbf{v} = \mathbf{G} \cdot \mathbf{u} = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \begin{bmatrix} u_L \\ u_R \end{bmatrix}, \tag{4.5}$$

where u_L and u_R are the left and right wheel control signals sent to the robot and $\mathbf{G}$ is a system gain matrix that is to be determined. Combining Equations (4.1) and (4.4) result in Equation (4.6).

$$\begin{aligned} \dot{\mathbf{P}} &= \mathbf{J}(\theta)\mathbf{L}\mathbf{v} \\ &= \mathbf{J}(\theta)\mathbf{L}\mathbf{G}\mathbf{u} \end{aligned} \tag{4.6}$$

The system gain matrix $\mathbf{G}$ is a characteristic of the hardware (Equation (4.7)). The determination of $\mathbf{G}$ can be simplified by assuming that the DC motors of both wheels are matched ($g_{11} = g_{22} = g$), and the input signals and output velocities for left and right wheel are decoupled from each other ($g_{12} = g_{21} = 0$). As a result, the determination of G is reduced to finding a value for g. The value of g varies across different system settings, and is usually decided by experimental



(a) Robot position and orientation (b) Translational and rotational velocity

Figure 4.6: Kinematic state definition

Exp. No.	u_L	u_R	$v_{avg}(cm/s)$	g
1	20	20	35	1.75
2	30	30	54	1.80
3	50	50	82	1.64
4	80	80	135	1.69

Table 4.1: Experiments' summary for the determination of g value.

measurement. The experimentation is conducted by placing the robot at rest, then providing equal control signals to the left and right wheels. The trajectory of the robot as well as its instantaneous velocities are recorded and plotted. When the robot has accelerated to a steady state velocity, the value of g is calculated by dividing the measured velocity by the input control signal. The velocity of robot is obtained through the vision system feedback within the setup.

$$\mathbf{G} = \begin{bmatrix} g_{11} & 0 \\ 0 & g_{22} \end{bmatrix} = \begin{bmatrix} g & 0 \\ 0 & g \end{bmatrix} = \begin{bmatrix} \frac{v_L}{u_L} & 0 \\ 0 & \frac{v_R}{u_R} \end{bmatrix} \tag{4.7}$$

Table 4.1 summarizes the velocities used in the experimentation and the values of g. The average of g values obtained from different experimentations is considered in Equation (4.8). It is worth mentioning that the g value varies across robots and/or the field of operation.

$$g = (1.75 + 1.80 + 1.64 + 1.69)/4 = 1.72 \approx 1.7 \tag{4.8}$$

The knowledge of the mathematical model is used to derive expressions for

control signals based on inverse kinematics. Combining Equations (4.4) and (4.6) provides equation (4.9).

$$\begin{aligned} \mathbf{V} &= \mathbf{LGu} \\ \Longrightarrow \mathbf{u} &= \mathbf{G^{-1}L^{-1}V} \\ &= \begin{bmatrix} g & 0 \\ 0 & g \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{L} & \frac{1}{L} \end{bmatrix}^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix} \\ &= \frac{1}{g} \begin{bmatrix} 1 & -\frac{L}{2} \\ 1 & \frac{L}{2} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \end{aligned} \tag{4.9}$$

For any desired robot motion represented in terms of the translational (v) and rotational (ω) velocities, the corresponding digital control signals for the left (u_L) and right (u_R) wheels are,

$$u_L = \frac{1}{g}\left(v - \frac{\omega L}{2}\right) = \frac{V_{common} - V_{diff}}{g}, \tag{4.10}$$

$$u_R = \frac{1}{g}\left(v + \frac{\omega L}{2}\right) = \frac{V_{common} + V_{diff}}{g}, \tag{4.11}$$

where $V_{common} = v$ and $V_{diff} = \frac{\omega L}{2}$.

Chapter 5

Fuzzy Behavior Based Control of Multi-Robotic System

In this chapter, an extensive fuzzy behavior based architecture is proposed for the control of mobile robots in a multi-agent environment. The behavior based architecture helps to decompose the complex multiple robots system into smaller modules of roles, behaviors and actions, which are more easy and efficient to control. The use of fuzzy control introduces a dimension of human-like reasoning into the behavior based architecture, which is based on a biological foundation as well. Fuzzy logic is used in implementing individual behaviors, coordinating the various behaviors within a single robot, and selecting roles for each robotic agent. Two methods of behavior coordination, fuzzy rule-base coordination and activity and action contribution, are explored. The proposed architecture utilizes the former method as the backbone for coordinating deliberative behaviors as well as selecting and assigning roles, and the latter for reactive behavior arbitration.

The robot soccer system is used as the test platform. The system provides an environment which is complex and dynamic enough to study the robot behaviors. As a result, it is also frequently used by researchers as a benchmark platform for multi-agent systems. The effectiveness of fuzzy behavior architecture can be evaluated conveniently and comprehensively in this platform.

The architecture is implemented on a team of three robots performing the roles of attacker, midfielder, defender and goalie interchangeably. Issues relating to the

design of effective behaviors are looked into. In particular, the robot behaviors and roles are designed to be complementary to each other, so that a coherent team of robots exhibiting good collective behavior is obtained.

The major practical issues that influence the implementation of the behavior based architecture on actual hardware are real-time vision processing, RF communication and robot mechanical motion. These issues create real-time problems, such as disturbances and delays, which inevitably add to the challenges in the design and implementation.

5.1 Introduction

In this work, a fuzzy behavior based architecture is developed to manage a team of mobile robots, by decomposing the whole team into different roles, each role into different behaviors, and then behaviors into primitive actions. Complex and intelligent behaviors are expected to emerge from simple and primitive sub-behaviors or actions, while group behaviors emerge from the cooperative or competitive behaviors of individual robots. This work explores the areas from design and implementation of robot actions and behavior, coordination of behaviors within an individual robot, right up to role building and role selection for multiple mobile robots within a team. Fuzzy concepts are brought in for behavior building and coordination, as well as in robot perception, decision-making and speed control.

To conduct research on such an extensive architecture, a robot soccer system is selected as a platform that provides a sufficiently dynamic environment for the multiple mobile robots to operate in. Due to the various human references made in the game, the robot soccer system is a suitable setup for the study of the fuzzy behavior based architecture. The robot soccer system is a typical multi-agent robotic environment where robots need to cooperate or compete with each other to fulfill certain tasks. In addition, each robot soccer player needs to undergo a cognition process much similar to a human player, to decide which behaviors to exhibit in order to win the game. In the field of robot soccer, fuzzy logic has

already been employed in implementing individual robot behaviors and actions, in particular for shooting and obstacle avoidance actions [58, 57]. Comparatively, there are fewer initiatives for the behavior coordination aspect in robot soccer. Among the material found in literature include a brief introduction to the use of hierarchical fuzzy control in negotiating behavior based architecture [67], and the use of fuzzy logic in game strategy selection [116].

This work focuses on mimicking the knowledge and cognition process used in the human game as much as possible. With the understanding that the human cognition process is a highly complex process, no single method of control can be used across the board. This work attempts to explore the use of principles for robot navigation, and extends them to the robot soccer scenario, which is a much more dynamic and competitive environment where rival robots are involved. Various methods for behavior coordination are explored and used for different subsets of behaviors within the architecture, and in particular, marking a distinction between deliberative and reactive behaviors. The proposed algorithm utilizes different methods of coordination for different behaviors under various scenarios so as to handle the highly dynamic environment.

This chapter is organized as follows. In Section 5.2, the design principles behind the behavior based architecture, that is, the decomposition and coordination of behaviors, are discussed. Sections 5.3 to 5.7 describe the design and implementation of the various levels of the behavior hierarchy, from action, to behavior, to roles, and finally, the team formation. The uses of fuzzy logic for robot perception, decision-making and speed control are introduced. A brief summary of the results is provided in Section 5.8, and the chapter concludes, in Section 5.9, by putting the work presented into perspective.

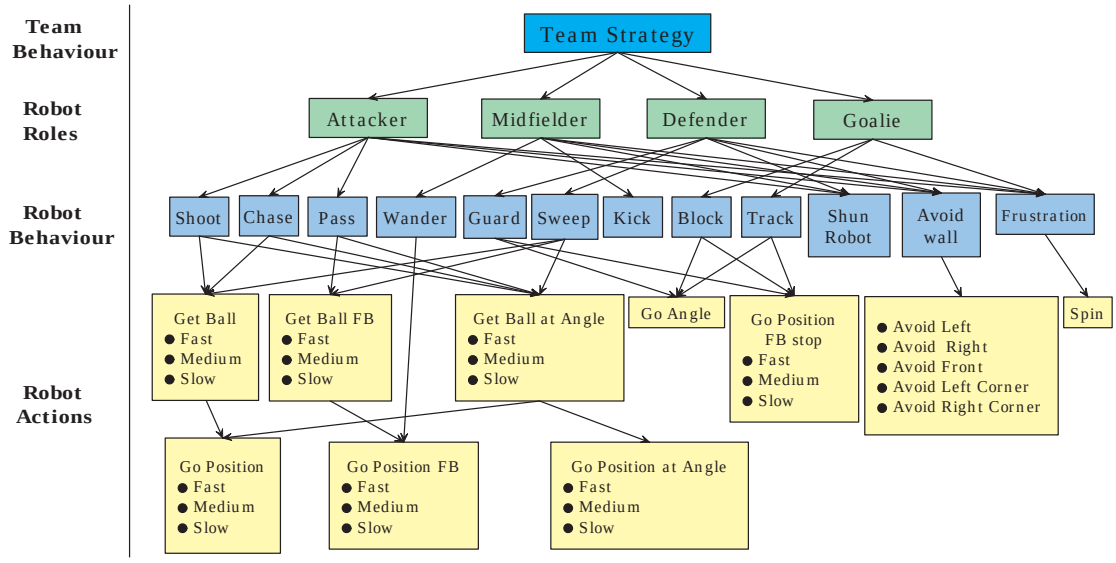


Figure 5.1: The behavior architecture for a team of soccer robots

5.2 Design Concept

This section describes the principles and theory behind the design of the fuzzy behavior based architecture. How behavior is perceived and decomposed in a behavior based architecture is described. Detailed descriptions are provided on two methods of behavior coordination, the fuzzy rule-base coordination and, activity and action contribution method. How these two methods are implemented in the proposed architecture is also discussed.

The principles of the proposed fuzzy behavior based architecture can be applied for various organizations of multiple mobile robots. The architecture is implemented on a team of soccer robots, and references are made to this scenario in illustrating the theory.

5.2.1 The behavior based architecture

A robot, which is assigned with a specific task, is deemed to display a task-achieving behavior. With a broadened scope of behaviors, the whole team of robots is expected to display a collective behavior. Due to different degrees of interpretations of behavior, there is a need to clearly define how behaviors are decomposed. The

proposed architecture decomposes behaviors into a hierarchy based on behaviors' complexity. The aim is to decompose complex behaviors into simpler and manageable sub-behaviors as one progresses from the top to bottom of the hierarchy. Figure 5.1 shows the behavior architecture with examples of behaviors at each layer of the hierarchy.

At the top of the hierarchy is the overall team behavior representing the team strategy, which can be aggressive or defensive, depending on the match situation. This is the most complex behavior as it represents the collective conduct of an organization of different agents. At the second layer, the team is split into four robot roles: attacker, midfielder, defender and goalie. Which role a robot has to perform is decided by the fuzzy inference engine. The four roles are subdivided into task-specific behaviors at the third level, such as "shoot", "chase", "pass", "block", etc. Each role is fulfilled by a set of behaviors. Some of the behaviors ("avoid wall" and "frustration") are utilized in most of the roles while others ("shoot"/"guard") belong to specific roles ("attacker"/"defender"). These behaviors are further decomposed into primitive actions, including "go–position", "go–angle" and "spin". These basic actions, which are repeatedly used to implement various behaviors, make up the bottom layer of the hierarchy. It is worthy of mention that some actions are comprised of sub-actions characterized with different degrees of speed. For instance, the "get-ball-at-angle" action consists of sub-actions with three speed settings: fast, medium and slow. The fuzzy decision process decides on how much of each sub-action contributes to the resultant robot speed. Essentially, the aim of the architecture is to decompose a system into simpler sub-modules until the sub-modules are straightforward enough to be handled on its own. The entire behavior based architecture for robot soccer system involves the building and integration of:

- Four soccer robot roles,
- twelve individual robot behaviors, and
- fourteen primitive robot actions, some of which have three speed settings.

In the architecture, obstacle avoidance is explicitly fulfilled by independent

behaviors, such as shun-robots and avoid-wall, while other behaviors need not take it into consideration. This is in contrast to other reported approaches where obstacle avoidance is included as an additional consideration when implementing behaviors like shoot ball [58, 57]. In the proposed architecture, obstacle avoidance behaviors are reactive behaviors, which are stimulated directly in response to the environment.

In addition, a frustration behavior is incorporated in. Frustration is used as a measure of the time in which the robot is in some form of difficulty, like being stuck together with another robot, and then activates a haphazard action like spinning to release itself from that situation. One might view frustration as an emotion of being in difficulty rather than a behavior.

Apart from regarding the architecture as a top-down decomposition of more complex functions to simpler ones, another way is to view the architecture as a bottom-up building of more complex behaviors from simpler ones. For example, the avoid-wall behavior is built up from the avoid-front, avoid-left, avoid-left-corner, avoid-right and avoid-right-corner actions. To perform the role of an attacker, the robot needs to possess behaviors like shoot, chase and pass ball. The attacker, midfielder, defender and goalie roles build the collective behavior and strategy of the whole soccer team. These two ways of interpreting the architecture also coincide with two schools of thought in behavior coordination, which is further discussed in the following section.

5.2.2 Action and behavior coordination

The coordination of different actions into behaviors, behaviors into roles and the selection of roles for different robot agents are performed using fuzzy techniques. Two methods of coordination are explored, namely, fuzzy rule-base coordination, and, activity and action contribution.

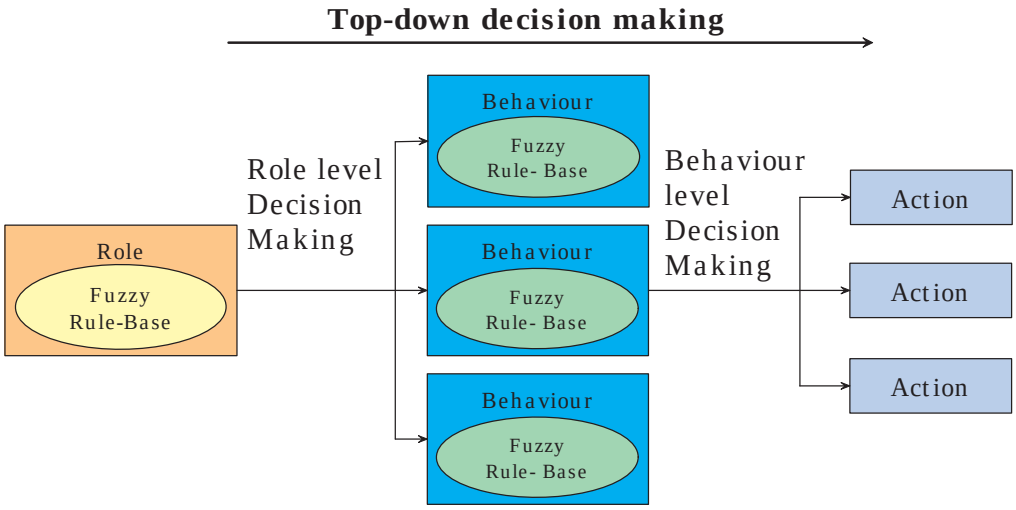


Figure 5.2: Fuzzy rule-base coordination

Fuzzy rule-base coordination

The fuzzy rule-base coordination method uses a fuzzy rule-base to select roles, behaviors or actions. This method adopts a top-down decision making approach. The output of each rule in the rule-base is a fuzzy singleton representing one behavior or action. For example, in the attacker's rule-base, there can be a rule: "IF (Ball is within shooting range) THEN (Shoot ball)". The whole fuzzy rule-base contains a series of such rules, which assesses the conditions within the playground and the team's game strategy, and proposes a suitable action. Figure 5.2 depicts the process of fuzzy rule base coordination.

The defuzzification method in the fuzzy decision process is important to coordination. Two defuzzification methods are explored: center of area (CoA) and "max" criterion [9]. The use of either of the defuzzification methods depends on the behaviors and actions at hand. For the CoA method, rules with positive strength activate respective behaviors by recommending a pair of speeds for the left and right wheels. These recommendations are then weighted according to the associated rules' strengths and the final average forms the overall behavior of the role. The CoA method is used to "merge" behaviors and actions which are compatible to each other. The "max" criterion activates only one behavior at any time, which

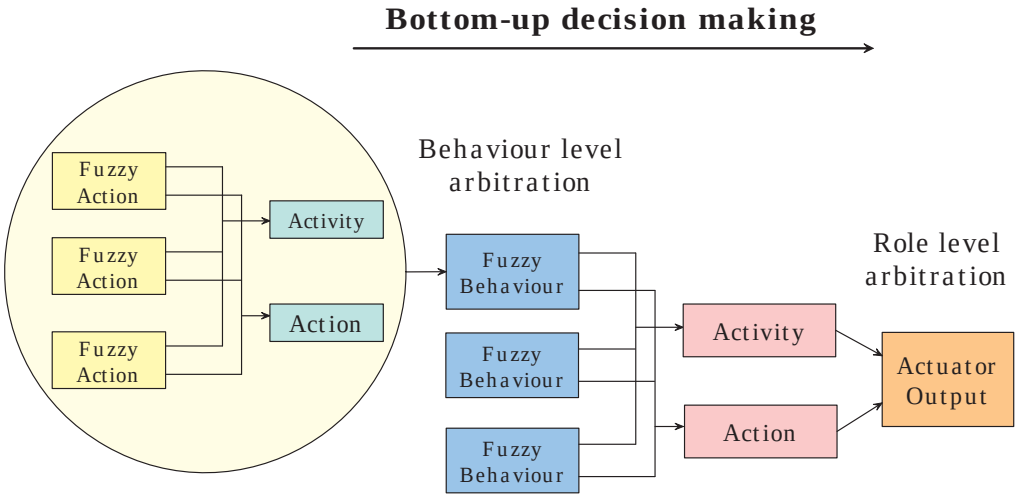


Figure 5.3: Activity and action contribution

is determined by the rule with the highest strength. In general, the "max" criterion is used for coordinating behaviors and actions that are mutually exclusive; for example, the chase and shoot ball behaviors for attacker role.

Activity and action contribution

The activity and action contribution method is based on the work by Abreu and Correia [55], and is in contrast to the rule-based approach due to its bottom-up approach in behavior building. Figure 5.3 shows the activity and action contribution method.

In activity and action contribution, each fuzzy action provides a pair of activity and action values. Action represents the output of the fuzzy action, and in this case, is the pair of left and right wheel speeds recommended by the fuzzy action. Activity represents the degree of contribution a fuzzy action has on the overall behavior. The activity value in the range of [0, 1] is determined in a fuzzy manner. One way of evaluating the activity value is by accessing the strength of the fuzzy rule which holds the greatest significance to the fuzzy action. The main idea here is that the activity value is obtained from the fuzzy statements within the rule-base of the fuzzy action. It is the action that assesses the environmental conditions and then expresses an opinion. The top-level arbitration merely reviews the different

opinions and then adopts one of the opinions or an aggregate of opinions. That is why the activity and action contribution method is considered as a bottom-up approach.

For the behavior coordination, the CoA and "max" criterion methods are also utilized. The CoA method takes the average of the action values recommended by the fuzzy actions, weighted by the activity values. The "max" criterion activates the fuzzy action with the greatest activity value. The application of either of the methods varies from behavior to behavior, and generally depends on whether the actions are mutually exclusive or not. The final action value computed forms that behavior's action value which is recommended to the relevant role in the upper layer. As to a behavior's final activity value, the determination methods also vary for different behaviors. A simple way is to take the highest activity values from its sub-actions as the overall activity value of the behavior.

Method of coordination

Based on the methods discussed, the architecture for coordination of roles, behaviors and actions is proposed. For the assignment of roles to the robot soccer players, the fuzzy rule-base coordination method is used. The antecedents of the fuzzy rules contain the decision factors relating to the conditions on the playground, the positions of opponents and the ball, and the game strategy of the whole team. The roles assigned to the robot players should show cooperative and supportive behaviors among team-mates so as to exhibit collective group behavior, which is resistive against the opposing team. For each role, behavior coordination is also based on fuzzy rule-base coordination. The exception is with the goalie, whose behavior is built on non-fuzzy if-else-if rules. The reason for doing so is that there are only a few rules governing the goalie's behavior. In addition, deciding whether the ball is moving to the home goal or not, is more or less a discrete decision. Hence, it is not necessary to use fuzzy logic for the goalie coordination.

Within each behavior, the method of action fusion and selection is based on the type of behavior. In general, the activity and action contribution method is used for

reactive behaviors like the avoid-wall behavior. The other deliberative behaviors such as shoot, pass and block, in principle, adopt fuzzy rule-base coordination. However, most of the deliberative behaviors in the proposed architecture are easily implemented based on one or two robot actions. The basic actions, like go-position and go-position-at-angle, are implemented using the fuzzy inference engine.

The rationale for merging the deliberative and reactive schools of thought is as follows. The idea of top-down behavior coordination is likened to a deliberate mode of decision-making. This process of thought applies more to behaviors like shooting, and ball passing that are displayed based on the game strategy held by the team. On the other hand, the bottom-up approach is more applicable for reactive behaviors, like obstacle avoidance.

As the behavioral architecture is combining a top-down and bottom-up approach, a compromise between the two decision-making processes needs to be made at the boundary where the two methods meet. At the meeting point, the activity value of the behavior received from the bottom-up approach is treated the same as the rule strength in top-down fuzzy rule-base approach. The activity value also represents the weight or significance of the sub-behavior/action to its superior behavior/role. For example, a role that contains the avoid-wall behavior can not activate that behavior based on the fuzzy rule. The avoid-wall behavior recommends itself when it is necessary. Under this circumstance, the role assesses the activity value of the avoid-wall behavior together with the strengths of the other rules in the rule-base to determine the activation of the overall behavior. In this way, reactive and deliberative behaviors/actions are seamlessly combined into the architecture.

5.3 Fuzzy Action Design and Implementation

Most of the actions in the behavior architecture are fuzzy actions, implemented using the fuzzy rule-base or inference engine. Depending on the type of action, different information are chosen as input to the different fuzzy rule-base. However,

the outputs from the fuzzy rule-base for all actions are the same: the speeds of the left and right wheels of the robot, which drive the robot to a desired posture. These basic fuzzy actions form the bedrock of a robot's behavior.

The fuzzy behavior based architecture results in a distributed fuzzy control system with small fuzzy sub-systems, instead of a single centralized one. Each fuzzy sub-system is simple to implement, especially those for the basic actions. Triangular and trapezoidal membership functions are widely used as they are effective enough for simple fuzzy actions. Small sized rule-bases are manually constructed and fine-tuned.

To illustrate the development of fuzzy actions, the go-position, go-position-at-angle and get-ball-at-angle actions are described as follows.

5.3.1 The go-position action

The go-position action is a movement the robot makes to bring itself to a desired position in the playground. The associated fuzzy rule-base takes in two inputs, the distance of the robot to the target position and the angular deviation of the robot's heading direction with respect to the target position. The rule-base then produces two outputs, the left and right wheel speeds.

The go-position action, as well as several other translation movements are implemented with three degrees of speed, namely slow, medium and fast. In the context of robot soccer, speed is a very critical issue when competition is involved. The rules of go-position are set up to provide only a single level of speed for each rule-base. Three different rule-bases are set up to cater for different levels of speed.

5.3.2 The go-position-at-angle action

Very often, a robot agent is required to approach a desired target with certain orientation. This is especially important for soccer-playing robots, which need to kick the ball in specific directions. The go-position-at-angle action achieves such a

motion.

A separate fuzzy rule-base is constructed for the robot to reach the target at a desired angle. This fuzzy rule-base is activated only when the robot moves within a certain region of the target, which is a circle area with pre-defined diameter.

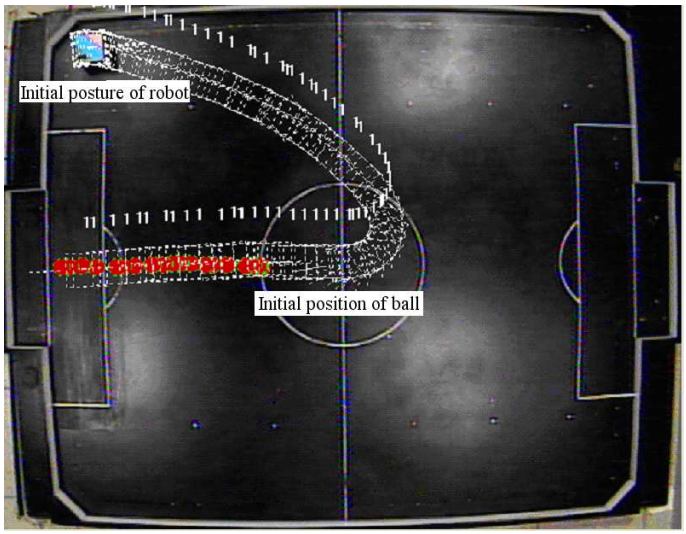
The two input variables to the fuzzy rule-base are the angles by which the current and the desired robot headings are away from the straight line between the centers of the robot and the ball. Essentially, the robot has to move in an approximately circular fashion around the target point when it is far away. When the robot closes in to the desired approaching angle, it makes a bigger turn to move towards the target.

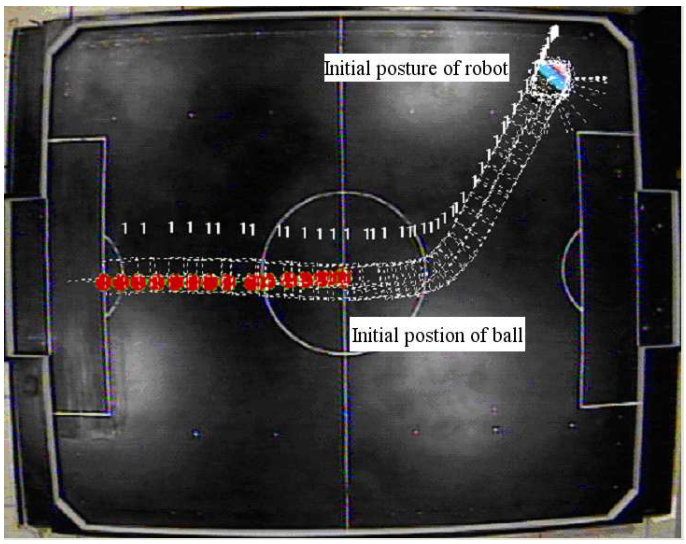
5.3.3 The get-ball-at-angle action

The get-ball-at-angle action is implemented based on the go-position-at-angle action, with the ball as the target position. This task is much more challenging as the ball is constantly on the move. It is important that the robot moves at an appropriate speed to reach the vicinity of the ball before kicking it at a desired angle.

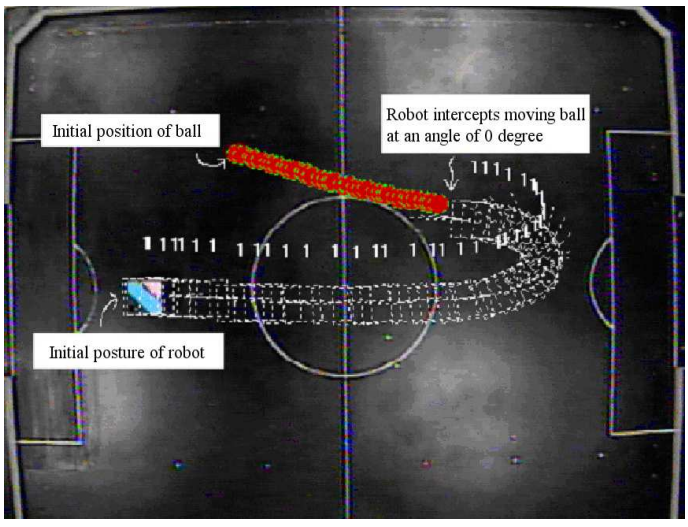
The implementations of the go-position-at-angle actions at different speed levels are activated under different scenarios to help the robot to reach the ball in the best possible manner. A fuzzy rule-base is used to determine the speed. The fuzzy input variables to the rule-base are the distance between the ball and the robot, the speed of the ball, and the direction of the ball relative to the robot's heading angle. The last variable determines whether the ball is currently moving towards the robot, or away from the robot, or in a direction perpendicular to the robot's heading direction. To further improve the performance, a fourth variable, the angle of approach to the target with respect to the direction of travel of ball, can be added to the fuzzy rule-base. However, this will be at the expense of additional computation time.

The performance of a robot displaying the get-ball-at-angle action is shown in





(a) Robot getting stationary ball at 0 degree



(b) Robot getting moving ball at 0 degree

Figure 5.4: Robot displaying the get-ball-at-angle action

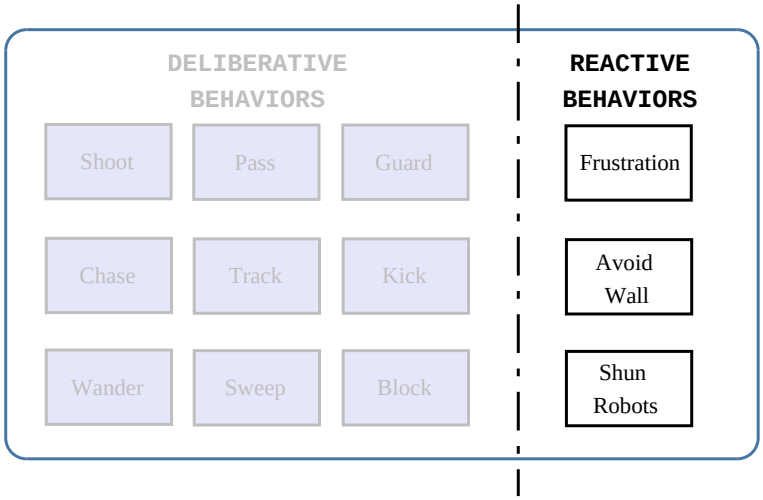


Figure 5.5: Robot behaviors with reactive behaviors highlighted

Figure 5.4. In this test, the robot is expected to approach the ball at 0°. When the ball is at the stationary state, the robot approaches the ball at the desired angle without any difficulty (Figure 5.4(a)). In the case of Figure 5.4(b), the ball is slightly in front of the robot and is traveling away from the robot. However, the robot moves at its fastest speed and manages to catch up with the ball and eventually intercepts it at the desired angle.

5.4 Reactive Behavior

The robot behaviors in the behavior architecture are divided into two classes: reactive and deliberative. Reactive behavior is defined as that displayed in direct response to environmental stimulation (Figure 5.5), as in the case of avoid-wall and shun-robots, or internal emotion (frustration). Deliberative behavior, on the other hand, involves some sort of intention of the robot agent. This section discusses the design and building of the reactive behaviors.

5.4.1 The avoid-wall behavior

There is a need for soccer robots to avoid the boundaries of the playground. The distances between the robot and the boundaries to the robot's left, left-corner

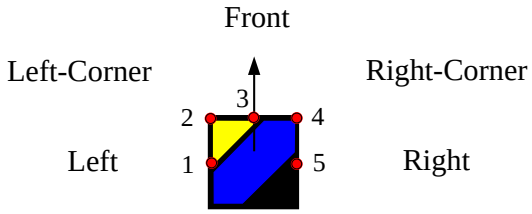


Figure 5.6: Five directions concerned in avoid-wall behavior

(45° left), front, right-corner (45° right) and right are evaluated (Figure 5.6). To monitor all the five variables together requires a very large fuzzy rule-base. The avoid-wall behavior is decomposed into five actions, namely, avoid-left, avoid-left-front, avoid-front, avoid-right-corner and avoid-right, with each action handling the obstacles in a particular direction. The coordination of these five actions is then accomplished using the activity and action contribution method. The outputs, the left and right wheel speeds, of each action form the action values. The activity values of the actions are based on the membership values of the associated input distance variables to the fuzzy rules.

The final left and right wheel speeds recommended by the avoid-wall behavior are computed using the CoA method. The activity value for the avoid-wall behavior is defined by the activity value of the avoid-front action which is the most crucial one.

5.4.2 The shun-robots behavior

Apart from avoiding the playground boundaries, robots also need to avoid the other robots in the playground. There are two steps in building the shun-robots behavior. The first is to build a fuzzy rule-base for shunning a single robot. The second is to take into account the other robots in the playground, and decide when to shun robots.

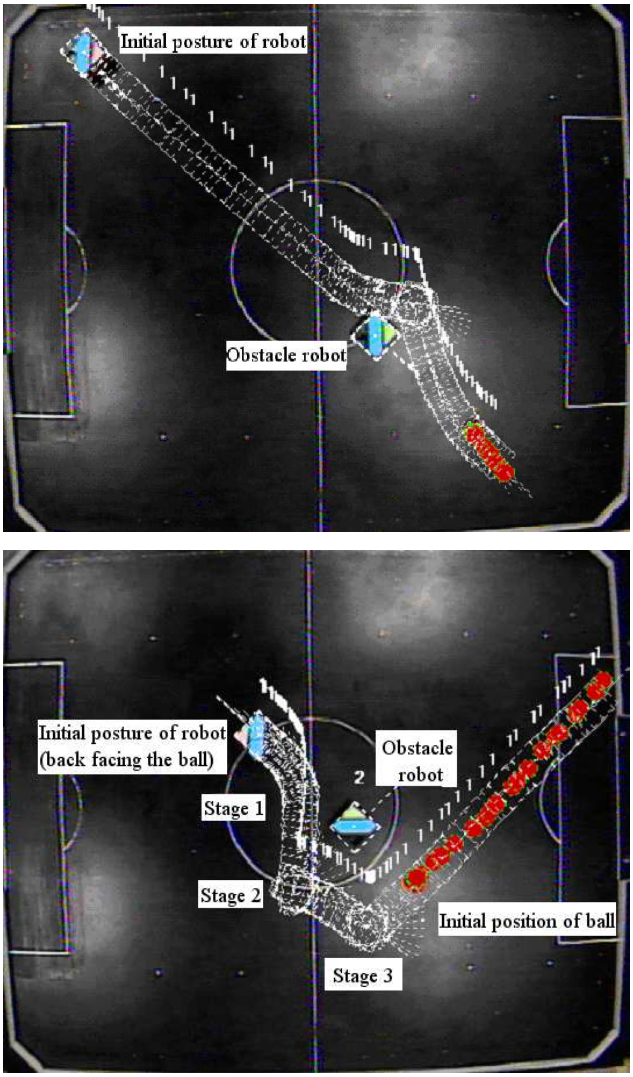
In the design of the fuzzy rule-base, the input variables are the distance to the obstacle robot and the angle by which the robot's heading differs from the direction towards the obstacle robot. When taking into account multiple obstacle

robots, the shun-robots fuzzy rule-base is applied to each of the obstacle robots to determine the speeds required to avoid each of them individually. This can be viewed as assigning a shun-robot action to monitor each of the other robots in the playground. Thus, the activity and action contribution method is applied to shun multiple robots.

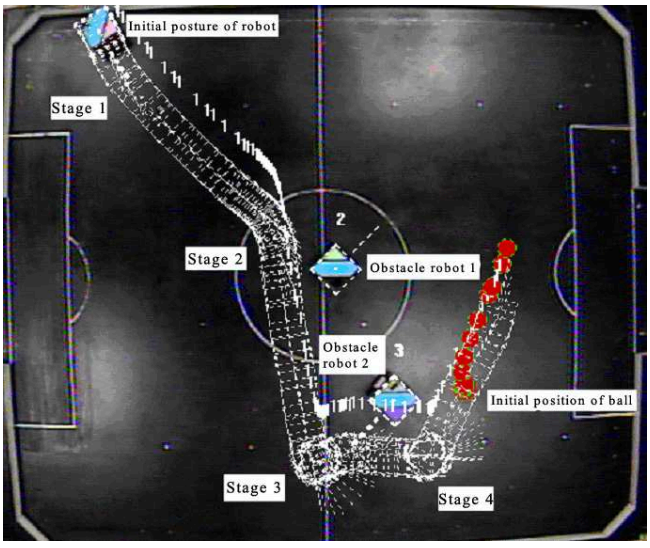
The activity value assists the robot to decide when to deploy the shun-robots behavior. This deals with the issue of perception. Fuzzy logic is employed to mimic this aspect of human behavior. The fuzzy notion of variable alert distance is introduced to determine the potential obstacles around a robot.

The variable alert distance helps to determine when surrounding robots are considered as near enough to be deemed as obstacles. The concept of the variable alert distance is derived from the fact that fast moving bodies need a longer time and distance to decelerate. Fast moving bodies, with large momentum, need to have greater anticipation of surrounding obstacles, as it needs the "space" to slow down and avoid the obstacles. For a fast moving body, the alert distance should be long, and the objects within this distance are considered as potential obstacles. The alert distance for a slow moving object is short. The slow moving robot only needs to beware of objects that are really close to it. Following this argument, a stationary object is not perceived to have any obstacles. The idea of a variable alert distance can assist to determine when obstacles are considered to be near, and it is used to derive the activity value. The activity value is computed through fuzzy inference, based on robot speed, or if possible, the relative speed between the robot and obstacle.

Figure 5.7 shows the performance of a robot that avoids two obstacle robots while moving towards the ball. The robot encounters no difficult in shunning the single obstacle robot (Figure 5.7(a)). When the robot is moving forward at a medium speed, it manages to bypass the obstacle within a very small margin. In the second case, the robot is moving backwards to the obstacle. It avoids the obstacle robot in the stage 1 and then swiftly adjusts its orientation to its desired direction (facing the ball) in stage 2 and 3. In the scenario of multiple obstacle robots



(a) Robot shunning obstacle robot while moving forward/backwards



(b) Robot shunning tow obstacle robots

Figure 5.7: Robot shunning obstacle robots

(Figure 5.7(b)), the robot begins with a high speed. At stage 1, the robot takes a slight turn to move towards the ball. It encounters obstacle-robot-1 at stage 2 and quickly turns towards the right. In the mean time, it bypasses obstacle-robot-2 as well. At stage 3, the robot turns left to move towards the ball again, but almost immediately finds obstacle-robot-2 in its path. It then quickly turns right to shun the obstacle-robot-2, and finally at stage 4, the robot turns and homes the target. It is observed that the robot has good ability in perceiving the presence of objects in the surroundings.

5.4.3 The frustration behavior

Strictly speaking, frustration should be studied as an emotion rather than a behavior. It is only the act of releasing the frustration that is considered as a behavior. Although this work does not include the study and modeling of robot emotions, the frustration behavior is implemented due to the need of the robot to resolve the many frustrating scenarios that is present in real world.

For instance, the robots used in the robot soccer system are cubic in shape. Such a geometric composition means that whenever robots are stuck together (Figure 5.8), it is very hard for them to release themselves. The frustration behavior is designed to monitor situations where the robot is stuck with another robot or when it is stuck against the walls. The frustration behavior monitors the period of time for which the robot is in such frustrating situations. This variable time is then fuzzified to obtain the activity value of the frustration behavior.

The behavior induced by frustration can be any haphazard reaction aimed at releasing the frustration. An example is the spinning action. The degree of this spinning motion is pegged against the period of frustration. If the period of frustration is long, the robot is having trouble with moving out of a tight situation by ordinary spinning and then it is required to spin at a higher velocity.

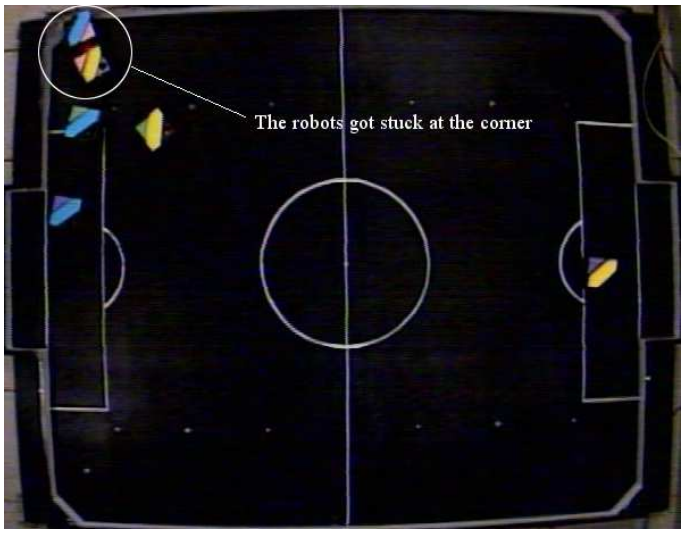


Figure 5.8: The situation to trigger frustration behavior

5.5 Deliberative Behavior

In contrast to reactive behavior, deliberative behavior is displayed not purely in response to environmental situations, but also due to some task that the agent is undertaking. For the soccer robot, this includes behavior for shooting, chasing, passing, wandering, guarding, sweeping, blocking and tracking (Figure 5.9). These are behaviors which deal with game strategy and are put on with the intention of winning a soccer game.

Most of the deliberative behaviors in this architecture are implemented directly based on the basic fuzzy actions. This is mainly due to the fact that the basic actions like get-ball-at-angle are very flexible when applied to ball seeking and manoeuvring. The emphasis on the building of deliberative behaviors falls more on the tactics considered in the design. The behaviors are designed to be simple modules, which are complementary to each other, so that after integration, coherent roles and logical group behavior of a soccer team can emerge.

Fuzzy logic is heavily used here for robot decision-making, such as determining the direction to dribble the ball, and the target position to shoot the ball. Fuzzy coordination is also used in speed control. As instances, the shoot and block behaviors are described.

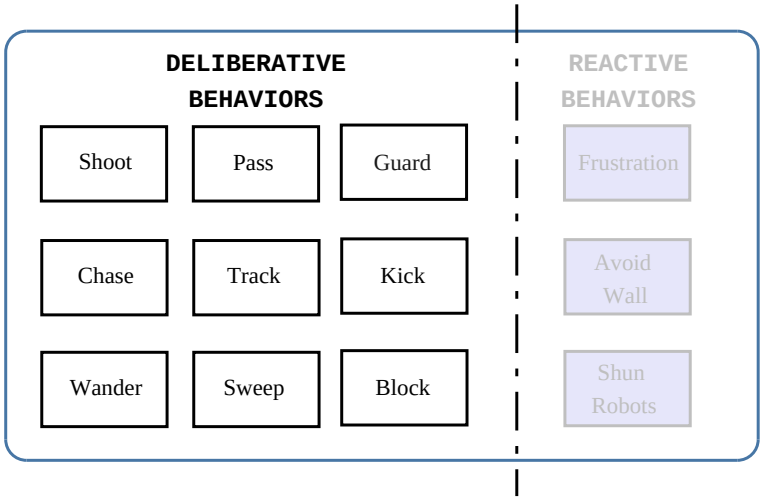


Figure 5.9: Robot behaviors with deliberative behaviors highlighted

5.5.1 The shoot behavior

The shoot behavior belongs to the attacker role and is vital in winning a game. This behavior is implemented by using the get-ball-at-angle action. The prime issue here is to decide the best angle to shoot, which in turn is determined by the target position that the robot is aiming at. To improve the chances of finding the target and eventually scoring, shooting should take into account the opponent goalie's position and the current ball position. Fuzzy reasoning is brought into determining a good target position to shoot at.

The input variables to the fuzzy rule-base of the shoot behavior are the ball position and the opponent goalie position. The output variable is the desired target position to shoot. Defuzzification provides the desired target position, which is used to compute the angle at which to shoot the ball. The resultant shoot angle is used to activate the get-ball-at-angle action.

5.5.2 The block behavior

The block behavior belongs to the goalie role. The block behavior's major task is to perceive where the ball is likely to hit the home goal, and to block it by moving to the perceived target position. This is achieved by prediction and extrapolation

on where the ball is heading for.

The goalie robot uses the goalie-go-position-FB (FB: forward and backward) action to move to the blocking position as accurately and fast as possible. The goalie robot should have the ability to decelerate quickly to stop at the desired position while moving at high speed. Furthermore, the goalie should try to maintain its motion along a straight line, in front of the goal.

There are three speed levels for goalie-go-position-FB actions. Experiments show that the fast version of goalie-go-position-FB enables the goalie to reach the blocking position very quickly. However, it turns out that the hardware could not undertake such high-speed requirements successfully all the time. It is observed that the robot sometimes skids and then overshoots the desired position. There is a need to damp the robot motion. It is also observed that the medium and slow versions of goalie-go-position-FB do not have skidding problems and achieved good performance when the ball is moving not so fast or when it is kicked from a far-away distance. The three versions of the goalie-go-position-FB actions are merged together to form the block behavior. Coordination of the three actions is carried out using the fuzzy rule-base coordination with the CoA defuzzification. The input variables for goalie-go-position-FB fuzzy rules are the distance of the ball to the home goal and its speed. The outputs are the left and right wheel speeds under the three speed levels. For the rules with positive membership grades, their respective output actions are activated, and the recommended output speeds are aggregated to obtain a desirable speed. It is observed that after this action coordination is implemented, occurrences of robot skidding have decreased considerably.

5.6 Behavior Coordination and Role Building

After each individual robot behaviors are built, the design involves building up the overall behavior of a robot, or the various robot roles.

5.6.1 Design approach

In this work, four roles are designed: the attacker, the midfielder, the defender and the goalie. The characteristics of each role and its compatibility with the other roles are kept in mind during the design. All behavior coordination is performed using fuzzy rule-base coordination except for the goalie, which is implemented using simple if-else-if rules. To illustrate the building of robot roles, the attacker, defender and goalie role are discussed.

Before designing the roles, a prior study on behavior coordination on a generic robot is made. Different roles are built based on the generic robot by specifically expanding the rule base for behavior coordination. Such a prior testing, which coordinates only a group of most basic behaviors, is important to detect early flaws. In the early testing of a generic robot, it is observed that when the ball is near or moving along the walls, the robot often oscillates between avoiding the wall and getting the ball. This reveals that either the avoid-wall behavior has not been designed well enough to allow trailing along the wall, or the robot's fuzzy rule-base is not sufficiently tuned so as to provide higher priority to get the ball. Such an outcome reiterates the importance of having individual behaviors well designed in the implementation of the behavior based architecture. It also reveals that improvements can be made by just changing rules or modifying membership functions of the fuzzy rule-base. The avoid-wall behavior has since then been tuned to better adapt to the wall trailing.

5.6.2 General behavior coordination

For a single robot playing by itself in the playground, the behaviors to be coordinated are kick, avoid-wall and shun-robot. The main purpose of the robot is to move to the ball and kick it, while avoiding the walls and other roaming robots. The fuzzy rule-base for behavior coordination for such a simple robot is as follows:

1. IF (Shun-robots behavior is active) AND (Ball is far away) THEN (Shun

robot)

2. IF (Avoid-wall behavior is active) THEN (Avoid wall)

3. IF (Shun-robots behavior is not active) AND (Avoid-wall behavior is not active) THEN (Kick ball).

The first two rules monitor the reactive behaviors in the architecture. The reactive behaviors calculate the associated activity values from lower level actions and these values form the grades of the antecedents or the rule strengths.

Experimentations show that the robot is able to move towards the ball in the best possible manner, that is, in a straight line, while successfully avoiding any roaming robot. Very often, the robot is found to slow down when the roaming robot lands on its pathway, and then picks up speed again when the roaming robot passes by. This observation is a desirable attribute, since most of the obstacles in the robot soccer system are moving robots. If an obstacle robot is moving head-on towards the robot, the robot should quickly shun that robot. Otherwise, it is better that the robot slows down and let the obstacle robot to pass by rather than quickly shunning it and ending up having to recalculate its direction. Collisions between robots are seldom observed except when robot speeds are very high. Similarly, the robot seldom crashes into walls unless it is traveling at a very high speed.

5.6.3 The attacker role

The attacker role is designed not just to perform the job of a striker. In fact, the attacker has the task of continuously pursuing and gaining possession of the ball. The fuzzy rule-base of the attacker is designed as follows:

1. IF (Frustration is active) THEN (Release frustration)

2. IF (Robot is heading towards wall) THEN (Avoid wall)

3. IF (Robot is blocked by other robots) AND (Destination is faraway) THEN (Shun obstacle robots)

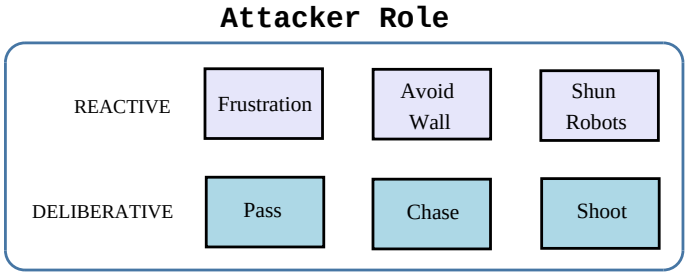


Figure 5.10: Behaviors of the attacker role

4. IF (Robot is surrounded by opponents) THEN (Pass ball to team-mate)

5. IF (Ball is within shooting range) AND (Robot is not constrained) THEN (Shoot ball)

6. IF (Ball is not within shooting range) AND (Robot is not constrained) THEN (Chase ball).

The first three rules are related to the environmental constraints on the robot, while the final three rules deal with game tactics. The strengths of the rules for reactive behavior are tied to their activity values, whereas those for deliberative behavior are determined by some of the antecedents (Figure 5.10). In particular, the two deliberative behaviors of shoot and chase can only be displayed if the robot is not constrained either by the wall or opponents. The grade of the antecedent "robot is not constrained" is obtained by taking the assessment of the maximum of the rule strengths of the first three rules.

5.6.4 The defender role

The defender is supposed to help the goalie in guarding the home goal and clearing the shots that are directed towards the home goal. The rule base of the defender is shown as follows.

1. IF (Frustration is active) THEN (Release frustration)

2. IF (Robot is heading towards wall) THEN (Avoid wall)

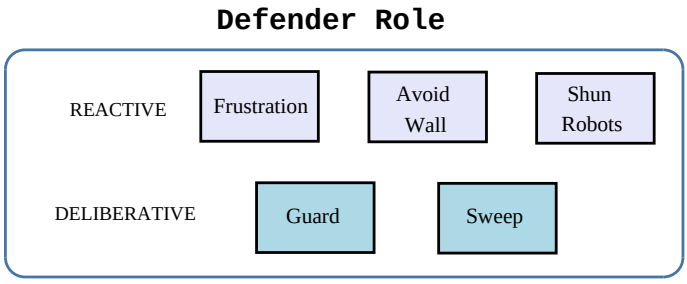


Figure 5.11: Behaviors of the defender role

3. IF (Robot is blocked by other robots) AND (Destination is faraway) THEN (Shun obstacle robots)

4. IF ((Ball is in danger zone) OR (Ball is moving towards home goal)) AND (Robot is not constrained) THEN (Sweep ball)

5. IF (Home side is not under attack) AND (Robot is not constrained) THEN (Guard home goal)

Once again, the first three rules handle reactive responses. The next two rules determine if the defender should stay on guard in front of goal or sweep the ball away (Figure 5.11). While designing the defender, it is kept in mind that there is an attacker which is responsible for winning the possession of the ball from the opponents. In order to avoid any clash with the attacker, the defender shall only sweep the ball when the ball is in the danger zone close to the goal or it is directed towards the goal. Otherwise, it is deemed that the home side is not under attack, and the defender should stay on guard.

5.6.5 The goalie role

The goalie is designed using straight-forward if-else-if rules instead of the fuzzy rule base. It consists of four behaviors (Figure 5.12) and they are organized as bellow.

- IF (Robot is frustrated) THEN (Release frustration)
- ELSE IF (Ball is at the sides of the goal) THEN (Kick ball away)

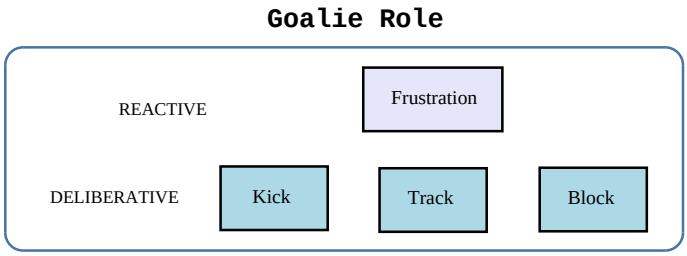


Figure 5.12: Behaviors of the goalie role

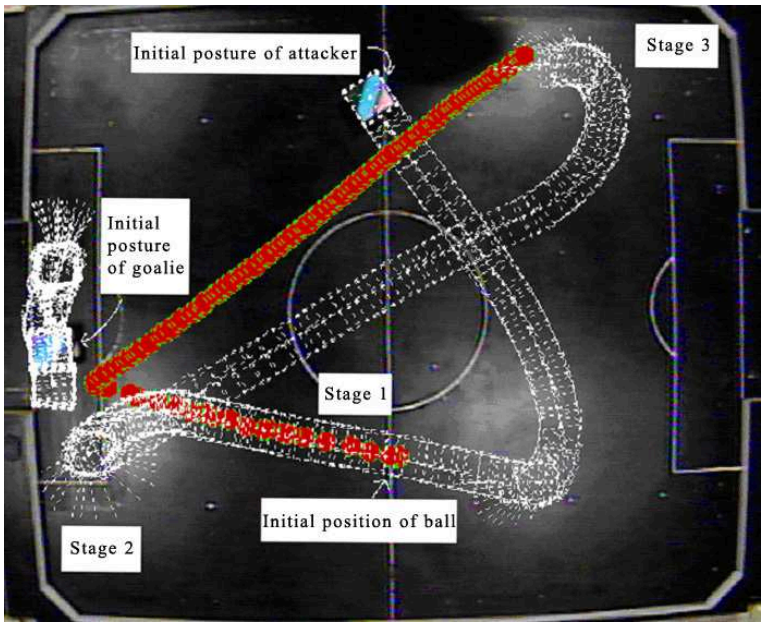


Figure 5.13: Performance of an attacker robot against a goalie

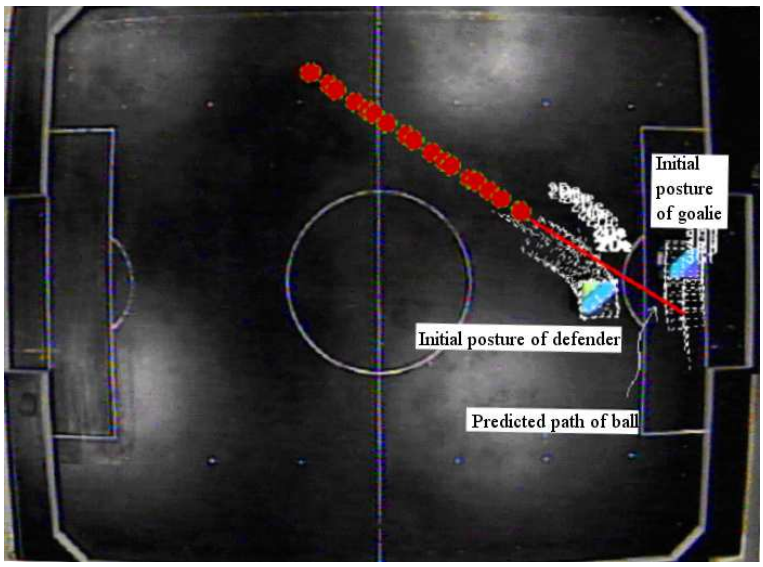


Figure 5.14: Performance of an defender assisting the goalie

- ELSE IF (Ball is moving towards home goal) THEN (Block ball)
- ELSE (Track ball)

The first reason for doing so is that there are only a few rules governing the goalie's behavior. Secondly, deciding whether the ball is moving to the home goal or not, is more or less a discrete decision. As such, it is believed that these few rules can be easily implemented without using fuzzy logic.

The performance of the attacker is tested by letting an attacker robot to play against a goalie. The overall paths taken by both robots are shown in Figure 5.13. In stage 1, the attacker moves from its initial position to shoot towards the goal, but the goalie manages to block. While shooting the ball towards goal, the attacker is traveling at high speed. However, it is observed that the attacker is able to slow down and avoid crashing into the goalie. The attacker, in fact, turned left to shun the goalie as shown at stage 2 (Figure 5.13). The attacker then prepares to turn around to chase after the ball. Meanwhile, the goalie continues to track the ball. At stage 3, the attacker, which is displaying its chase behavior, manages to intercept the ball.

The performance of the defender is illustrated in Figure 5.14. As the ball is travelling towards the goal, the goalie predicts where the ball is heading for (Figure 5.14). The goalie thus moves quickly to the perceived blocking position and would be able to block the ball. However, before the ball manages to reach the goal area, the defender has already emerged from its guarding position to intercept the ball and sweep it away.

5.7 Role Selection and Assignment

The final stage of the design involves the assignment of roles to the three robots in the team. The concerns in the assignment process include the characteristics of the roles and the game tactics deployed. In this case, only three of the four designed roles are displayed at one time. That is simply because there are only three robots

		Distance of Closest Opponent to Home Goal		
		Near	Medium	Far
Distance of Ball to Home Goal	Near	Defender	Defender	Midfielder
	Medium	Defender	Midfielder	Midfielder
	Far	Defender	Midfielder	Midfielder

Table 5.1: Fuzzy rule base for selection of the third robot role

in a team. Repeated roles are not allowed as conflict may arise between the two robots with the same role.

The assignment process is performed in two stages. The first stage is to select which three roles to be displayed at any time. The second stage is to assign these three roles to the three robots. This means that each robot may not perform the same role all the time and provision is made for role changing. Both role selection and assignment depend on the relative positions of the robots and ball in the playground.

5.7.1 Role selection

In the role selection, the presence of the goalie is always a must. In addition, the attacker is the only role which is actively pursuing the ball in the proposed designs. Hence, this role must be always present for the team to maintain possession of the ball. The last role will be either the midfielder or the defender, and that is decided based on the positions of the ball and the opponents in the playground.

The fuzzy logic is employed again to fulfill the selection of the third robot role. The input variables are the distance of the ball and the distance of the opponent robot which is closest to the home goal. The fuzzy rule base for role selection is shown in Table 5.1.

5.7.2 Role assignment

After the selection of the three roles is done, the next step involves assigning the selected roles to the robots in the team. As with a normal soccer game, the goalie role is always be performed by one particular robot.

The task then remains for the assignment of the two roles of attacker and midfielder, or attacker and defender to the two remaining robots. Since the attacker is designed to be the role that is pursuing the ball, the easiest method of role assignment is to let the robot which is closer to the ball be the attacker, and the other be the midfielder, or the defender.

5.8 Summary of Results

In view of all the experimentations performed, a summary of the results obtained is presented.

5.8.1 Fuzzy actions

The results show that the use of the fuzzy logic provides an easy way of implementing well-behaved actions. The ease of using the fuzzy rule-base is further appreciated considering the fact that the relationship between the inputs and outputs of the actions are highly nonlinear in nature. The versatility of fuzzy logic is observed from the fact that it is able to implement not only simple actions, like go-position and go-angle, but technically more skilful actions like get-ball-at-angle as well. The get-ball-at-angle action has proven to be a very important behavior as many of the deliberative behaviors like shoot, pass and chase rely heavily on it. In addition, fuzzy logic is also very useful in robot perception, decision-making and speed control.

Robots using fuzzy actions are found to move with better agility and greater purpose. In terms of target seeking, or ball seeking, the performance of the robots is

found to be very good, especially when the ball is stationary or moving at moderate speeds.

5.8.2 Robot behavior

For the reactive behaviors, the avoid-wall and frustration behavior are found to serve their purposes. The shun-robots behavior also worked well when obstacles are stationary or moving at moderate speeds. The main issues regarding the implementation of deliberative behavior is in the building of simple and yet competent behaviors. Results obtained from experimentations on the robot roles show that the deliberative behaviors worked effectively when combined together, and are instrumental in the successful implementation of the roles.

5.8.3 Robot roles

Among all the four roles designed, the goalie proves to be the most competent due to its high percentage of successful blocking. The good performance of the goalie mainly comes from the excellent performance of the block behavior. It can be seen that the actions and behaviors lower down in the behavior hierarchy greatly influence the performance of the roles and behaviors higher up in the hierarchy.

The coordination of the goalie's behavior does not use fuzzy techniques. However, this should not be interpreted to be that fuzzy coordination is not as useful as simple if-else-if rules. The simple reason for using if-else-if rules for the goalie is due to the fact that there is no compelling need to fuzzify the inputs to the goalie's rule-base, since they are mainly discrete decisions.

The attacker, midfielder and defender have performed up to expectations, bearing in mind that the capabilities and limits of the actions and behaviors that are used to implement them.

Within the limits of the hardware setup, it is found that the four different roles designed are complementary to each other, and result in robot behaviors which are

supportive among each other. Desirable role changes are also observed. Very often, when the attacker has possessed the ball and is dribbling it towards the opposition's side, the defender automatically moves up to midfield to support. Furthermore, when the opponent manages to resolve this attack and kicks the ball back to the home side, the midfielder often becomes the attacker and actively pursues the ball to intercept it.

5.8.4 Comparison with original system

The overall performance of the fuzzy behavior based robot soccer system is compared with the original in the competition.

In the original system, there is a goalie, defender and attacker. The attacker is always trying to pursue the ball and push the ball to the opponent goal area. The goalie guard the goal gate with the help of defender. The behavior of robot is decided by a simple strategy related non-fuzzy rule base. There are ten task-achieving robot behaviors to be performed by robot. To achieve the behavior's objective, the robot is supposed to reach certain destination point, sometimes with a desired orientational. For instance, the robot performing the "intercept-ball" behavior should reach the ball as soon as possible (Figure 5.15). Mathematic equations are set up based on the current velocities of the robot (V_r) and the ball (V_b), the current positions of the robot and the ball, as well as the direction that the balls velocity makes with the line adjoining the robot and the ball (α). The predicted position of interception and the time taken by robot (t) are resulted from the equations. Obviously, the position of interception is the robot's destination in the time of t. Based on these information, the trajectory of the robot is calculated by the trajectory planing module. From the trajectory, the desired position and heading of robot in the next time-step are decided and the speed control signals for the two wheels are generated. PID (Proportional-Integral-Derivative) controllers are used to ensure that the desired wheel speeds are achieved.

The fuzzy behavioral based robot soccer system and the original one are put

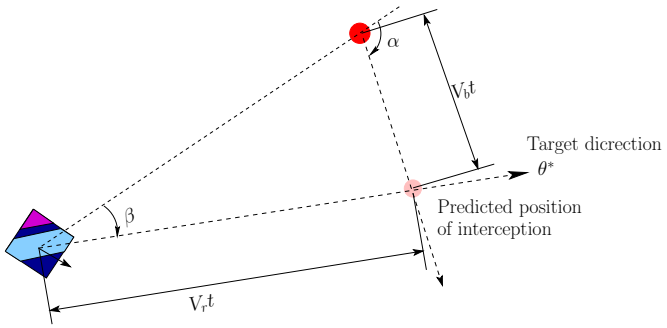


Figure 5.15: The "intercept-ball behavior in original system

into the match. Each match lasts for 10 minutes. The number of scores and shots are collected in 20 runs and the averaged values are tabulated in Table 5.2.

Performance Date	Original System	Fuzzy System
Scores	1	1.4
Number of shots	14	18.2

Table 5.2: Comparison of fuzzy and original robot soccer system

5.9 Conclusions

Behavior based robotics studies the desired types of behaviors for robots to display, as well as the algorithms and computational needs to achieve such behaviors. Each of these aspects has been dealt with to a certain degree in this work. A good portion of work has gone into designing behaviors and roles which are effective and beneficial for a team of soccer robots. The use of fuzzy logic as the tool to implement the behavior based architecture stems from the belief that the use of human reasoning can help robots attain behaviors that closely resembles animal and human behaviors. Performing this work on the actual hardware also allows the researchers to gain a better perspective on the actual interaction between the robots and their environment. The practical issues that arose from the hardware implementation also provided greater insights into the complexity of the problem at hand.

The results obtained have been very encouraging. The decomposition of a system into various simple behaviors has proven to be an effective method for implementing large control systems. The use of fuzzy logic together with the behavior based architecture has resulted in more agile robots, which are persistent in accomplishing their tasks. Results also show that multiple robots taking different and complementary roles have the tendency to display supportive behavior towards each other. This is due to the proper design of complementary behaviors within the architecture so as to build up a coherent system.

A major highlight in this work is that the whole study is performed directly on the actual robot soccer hardware. The act of bringing the robots out from the simulation platform into the real world allows the researcher to have a greater understanding of the environmental influences on the robots and the interaction between them. The algorithm used on the robots is also subjected to rigorous tests in the actual environment. Furthermore, implementation on actual hardware poses a lot of practical considerations and limitations. The results obtained under such conditions can be put into better perspective.

In conclusion, this work has established an extensive behavior architecture that can be easily adapted for further studies in related fields. In particular, the work discussed in this chapter seeks to contribute to the research in cooperative robotics and the development of social behavior in robotic systems, as well as the development of robot behaviors that parallel their biological counterparts.

Chapter 6

Adaptive Tuning in Fuzzy Behavior Based Robotic System

An extensive fuzzy behavior based architecture for a robot soccer system is outlined in Chapter 5. The various behaviors in the system were manually designed. Such a manual approach is inherently inefficient. In this chapter, an adaptive tuning methodology for the discussed fuzzy behavior based architecture is investigated. Two tuning methods are explored based on the manipulation of fuzzy membership functions and applied to the fuzzy behavioral architecture. The real world experimentation show improvements on the system's control performance.

6.1 Introduction

In Chapter 5, a fuzzy behavior based control architecture is developed for a robot soccer system. The behavior building, behavior coordination and robot decision-making blocks were realized through fuzzy logic. Such an approach has resulted in agile robots which have the tendency to exhibit supportive behaviors.

However, the fuzzy behavior based system is designed specifically to the current robot soccer system configuration. Furthermore, the fuzzy actions and behaviors are all static. Such a static architecture is incapable of handling all the unforeseen scenarios in the competitive and dynamic environment, which may degrade the system performance. Any changes in the system configuration features, including

playground's texture and robot's kinematic characteristic, require manual modification in the fuzzy controller. Since manual tuning is extremely tedious and time consuming, some kind of easier and faster tuning methods are needed to relieve the problem.

Two tuning methods, which change the position or shape of membership functions, are explored in this chapter. The tuning is applied to the fuzzy membership functions for different aspects of the fuzzy behavioral architecture: robot actions, robot roles and team strategy. The prime purpose of the tuning methodology is to provide adaptive fine-tuning ability to the system when notable environmental changes occur. Thus the behaviors and strategy can be suitably adjusted to cater for different situational needs, similar to an actual human soccer game.

Exploiting the adaptive tuning dispenses with the need for the re-formulation of the pre-defined fuzzy membership functions. Furthermore, the two tuning methods can form an off-line diagnostic tool which is useful in the formulation of new fuzzy membership functions and verifying the performance of existing ones. The diagnostic tool can replace the conventional "trial and error" approach. Real world experimentation are carried out on the robot soccer system to validate the proposed methodologies.

This chapter is organized as follows. Section 6.2 elaborates the tuning methodologies. Section 6.3 describes the implementation and experimentation while Section 6.4 summarizes the work. For convenience, the fuzzy behavior based system developed in Chapter 5 is referred as the original system in the rest of the chapter.

6.2 Tuning Methodologies

The adaptive tuning methods are focused on the fuzzy logic part while the behavioral hierarchy structure is kept intact. Since the fuzzy logic engine is the control kernel of the system, modifications on the fuzzy engine has direct influence on the system's performance. On the other hand, the system architecture remains stable

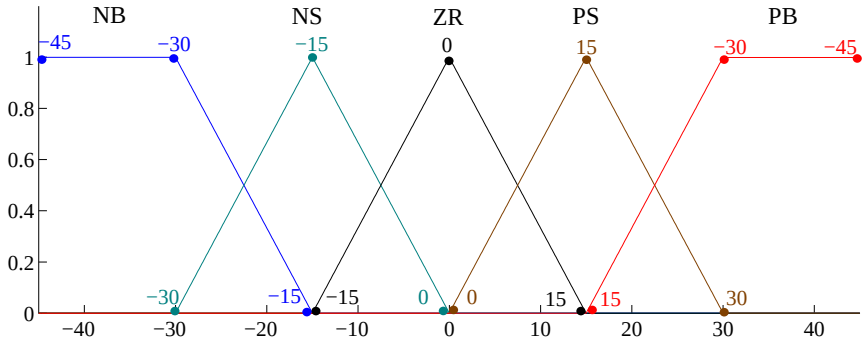


Figure 6.1: The parameterized fuzzy subsets

during the on-line tuning. That kind of stability is important in the middle of a real world match in which the tuning may happen.

The reforming of a fuzzy controller is usually concentrated on two parts: the fuzzy rule base and the fuzzy membership functions. On-line changes on the fuzzy rule base may greatly change the robot's behaviors and that is a bit risky in a real world match. Due to the importance of system stability, the membership functions are chosen as the subjects of adaptive tuning.

Triangular, trapezoidal, Gaussian bell curve and sigmoid are the commonly used fuzzy membership functions. Formed by straight lines, the triangular and trapezoidal membership functions have the advantage of simplicity, but lacking smoothness. Since one of the benefits of the behavior based architecture is the decomposition of complex fuzzy system into distributed and simpler fuzzy sub-systems, triangular and trapezoidal functions are sufficient for the sub-systems' fuzzy controllers and are prevalently adopted in the system.

The shape of the membership functions is usually parameterized by a set of values. For instance, there are three parameters for a triangular membership function (one for the peak point and the other two for the left and right bases) and four parameters for the trapezoidal one: the left and right shoulder points denote the positions of the associated fuzzy subsets in the universe of discourse, while the left and right base points represent the span of the function (Figure 6.1).

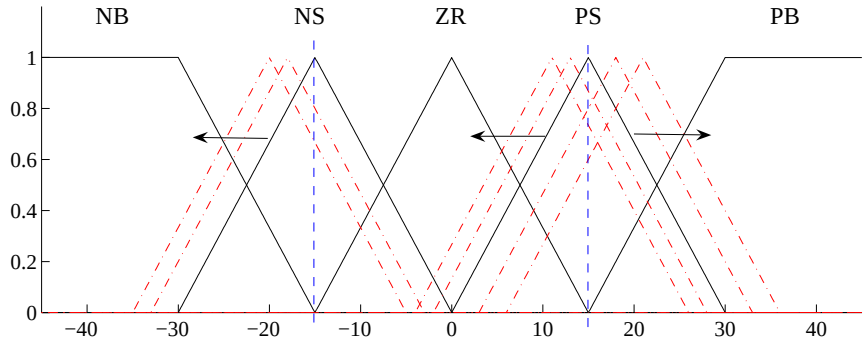


Figure 6.2: Translate-tuning and its imposed limits

One of the most commonly used defuzzification methods in the original system is center-of-area (CoA) method, whose mathematic form is defined in Equation 6.1 (for both the continuous and discrete cases):

$$\overline{y} = \frac{\int \mu(y) \cdot y dy}{\int \mu(y) dy} \quad or \quad \overline{y} = \frac{\sum_{i=1}^{n} \mu(y_i) \cdot y_i}{\sum_{i=1}^{n} \mu(y_i)}, \tag{6.1}$$

where $\overline{y}$ is the crisp output of the defuzzified variable y and, $\mu(y)$ is the degree of truth decided by the membership functions. Two tuning methodologies are developed according to the CoA method, resulting in either the shifting of the entire subset or the broadening/narrowing of the span of the subset. The first tuning method shifts a fuzzy subset along the universe of discourse in the entire output range (Figure 6.2). The crisp defuzzified output is manipulated by modifying the position of the subset. For example, if the positive subset for linguistic value "PS" is shifted to the left/right (Figure 6.2), the CoA of the subset is shifted. The overall CoA point is also shifted accordingly, that causes the output to decrease/increase. The y increases/decreases by shifting the CoA, while the related $\mu(y)$ remains the same. As a result, the output $\overline{y}$ is changed by shifting.

For smooth and gradual tuning, small shifting step is adopted so that the defuzzified output experiences no abrupt changes. Certain limits are imposed on the shifting to prevent the subsets from overlapping each other too much, which in fact reduces the number of subsets. Other limits are also needed to prevent the positive/negative subsets to overrun into the negative/positive side. There is no limit to the number of membership functions being tuned simultaneously. With the limits on the scope of tuning for each subset, simultaneous tuning on several

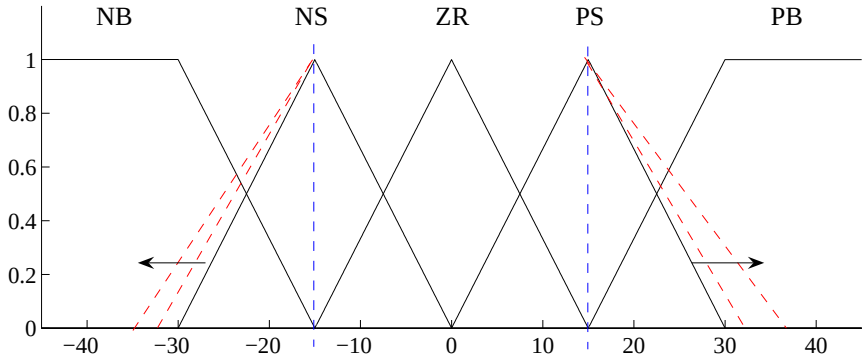


Figure 6.3: Base point tuning to increase output magnitude

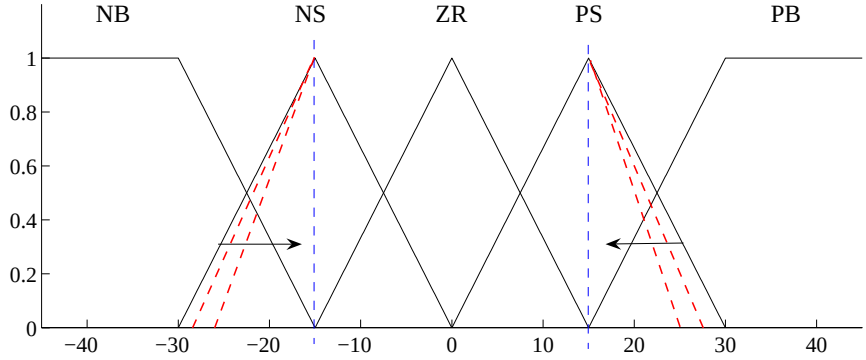


Figure 6.4: Base point tuning to decrease output magnitude

fuzzy subsets is indeed isolated so as not to affect each other.

Unlike shifting the whole membership function, the second tuning method alters the base points of the associated subsets and keeps the peak point fixed (Figures 6.3 and 6.4). By changing the shape of the membership functions, this method increases or decreases the output.

The modification of the base points makes the subset asymmetrical, thus changing the associated output. In Figure 6.3, the movement of the base points increases the positive/negative output's magnitude, while Figure 6.4 shows a contrary situation. The direction of the base point movement is related to whether the subset is positive or negative. The limits outlined for the shifting tuning method are imposed for the base point tuning as well.

It is noticed that the shifting tuning method usually has significant impact on

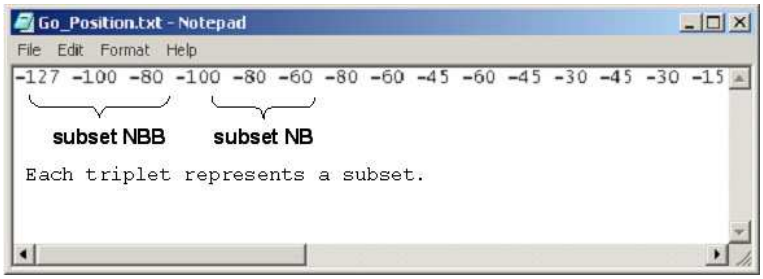


Figure 6.5: The parameter file

the membership functions while the base point tuning has a relatively moderate impact. The choice of which method to be used depends on the nature of the robot motion and the prevailing situation. For example, the shifting tuning can be used in fuzzy actions for straight motions in which more distinct increment or decrement of speed is allowed. Meanwhile, the base point tuning can be used to fine-tune a slight overshoot of an angle-turning action. The combined use of the two methods is also feasible.

The adaptive tuning is applied when the system needs to adapt to certain environmental changes. Certain criteria are set up based on the performance data. By checking the performance criteria from time to time, the system indirectly detects the environmental changes. Adjustments on fuzzy membership functions are triggered until the prescribed performance criteria are satisfied. The initialization of adaptive tuning can be manually carried out but it is the system itself adapts its behaviors to environmental changes. The process flow is depicted in Figure 6.6.

During the tuning process, the membership function's parameters are stored as a string of integers in text files named after their respective robot actions, behaviors or roles (Figure 6.5). For instance, each triplet represents a single triangular fuzzy subset. The prime purpose of the parameter files is to keep the resultant parameters of the tuned fuzzy subsets for future use. The parameters stored in the text files can be loaded into the system at any time, without the need to recompile the control program. By this way, the parameter files make the real-time alteration of the fuzzy subsets possible. Furthermore, various strategies can be activated by replacing the current parameter file(s) with those characterizing the different strategies, while the system is still running.

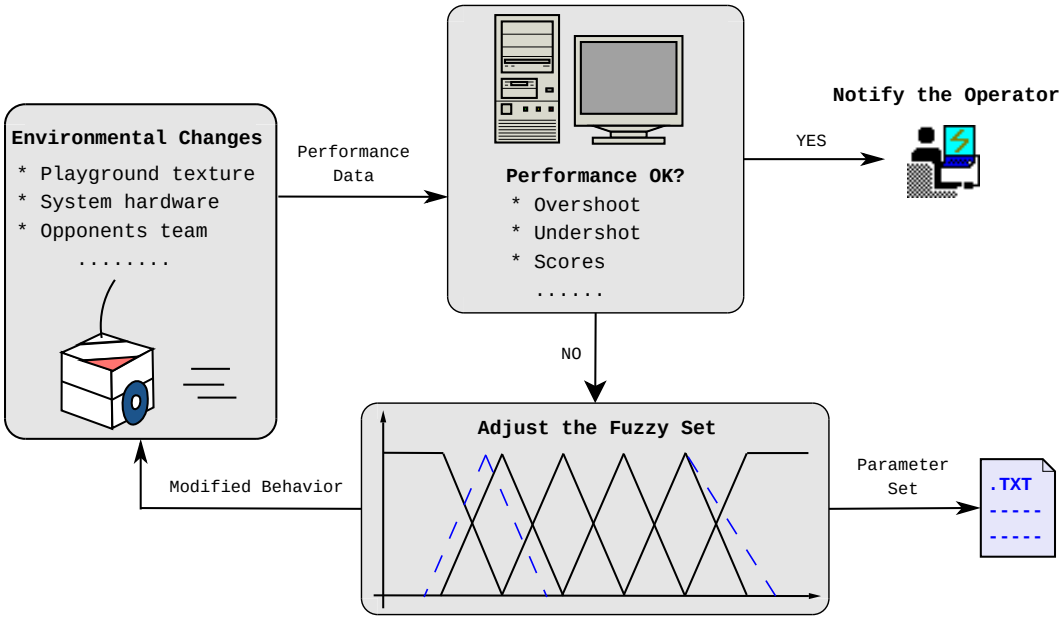


Figure 6.6: The adaptive tuning process flow

6.3 Experimental Implementation

The tuning methods discussed in Section 6.2 are implemented in a robot soccer system. For the fuzzy behavior based architecture, both the primitive actions at the base level and team strategy at the top level are selected to undergo the adaptive tuning.

The adaptive tuning on robot primitive actions aims to handle the environmental variations which cause undesirable control imprecision. For the tuning on basic fuzzy actions, only the output membership functions are subjected to tuning. As a result, the fuzzy inference engine is not affected.

6.3.1 Robot actions

There are always some changes occurring in the environment of the robot soccer system. The changes could be replacement of the playground with different texture, or changing the robot DC motors. The accuracy of primitive robot actions are affected due to such changes and fine-tuning is necessary. To carry out the adaptive tuning, the robot is scheduled to perform a specific primitive action repeatedly in

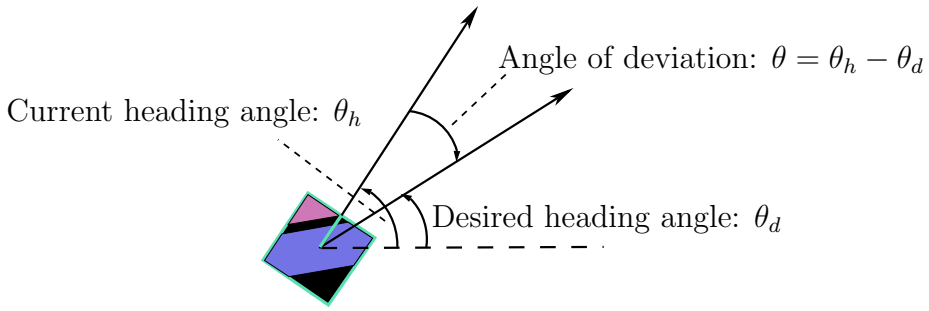


Figure 6.7: The go-angle action

the new environment. At each run, the system checks certain performance criteria, such as overshoot or undershoot. Accordingly, the output fuzzy membership functions related to the robot-speed are adjusted by one predefined step-size. The adjustments go on as the action is repeated, until the performance criteria are satisfied. A flag signal is set to indicate that enough number of runs are performed. In this way, the system successfully adapts its actions to the environmental changes.

One of the basic robot actions is the go-angle, whose purpose is to orient the robot to the desired angle - usually to face the direction of the ball. The input to the fuzzy engine is the angle of deviation θ, defined as the difference between the robot's current heading angle θ_h, and the desired heading angle θ_d (Figure 6.7). If the angle of deviation is negative, the robot performs a counter-clockwise turn. Since this is an on-the-spot rotation, the left and right wheel speeds should have the same value but of opposite directions.

The associated tuning process is straightforward. The robot is made to perform the go-angle action under different speed settings. The overshoot/undershoot is measured by the angular difference between the final heading angle and the desired one. If an overshoot/undershoot is detected, fuzzy subsets are adjusted to decrease/increase the speed. Depending on the magnitude of the overshoot, shift tuning or base points tuning is selected. The subset of the output fuzzy membership functions is shifted or broadened/narrowed by a predefined step-size. As the go-angel action being repeated, the angular error finally falls into the desired range and a suitable parameter set is reached.

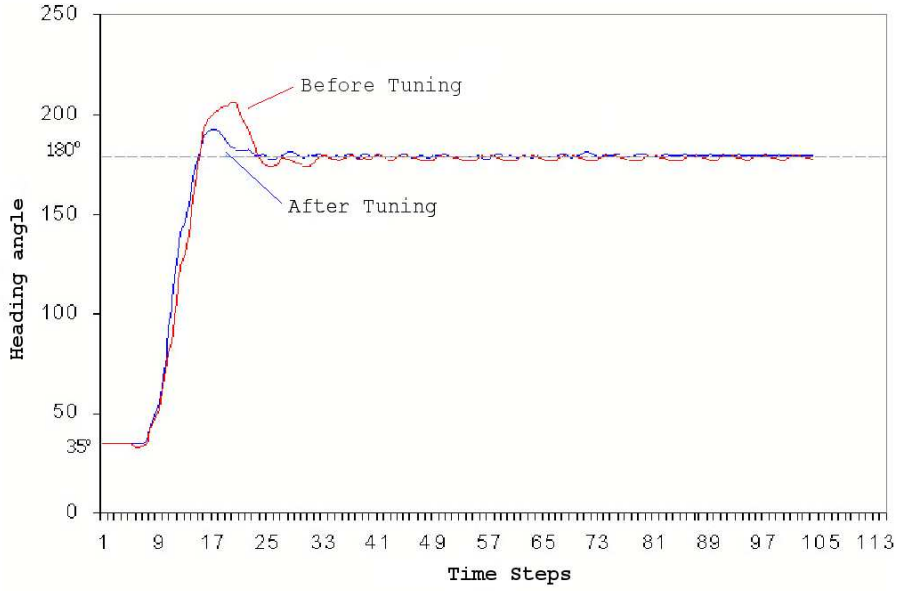


Figure 6.8: The effectiveness of adaptive tuning on go-angle action

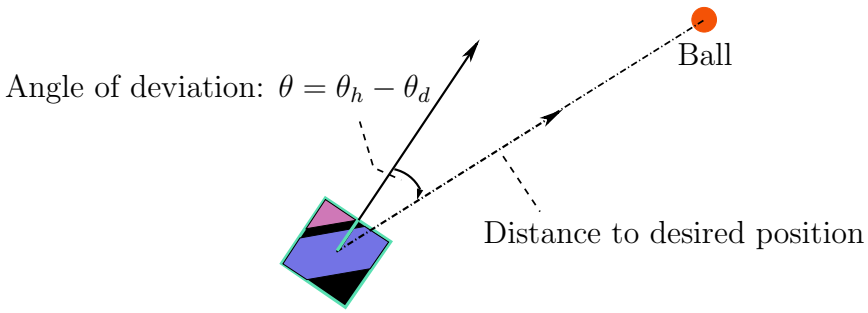


Figure 6.9: The go-position action

To verify the effectiveness of adaptive tuning, the performance of go-angle before and after the tuning is compared. The robot is made to orient itself from a specified heading angle of 35 degree to the desired heading angle of 180 degree. The trial is repeated for 25 times. At each time step in each run, the robot heading angle is recorded. The average heading angle at each time step is plotted in Figure 6.8. It is obvious that the control performance is improved with the application of parameter tuning. The overshoot is greatly decreased.

Another primitive action go-position (Figure 6.9) is implemented with adaptive tuning in the similar way. The go-position action is to move the robot to a desired position. The two inputs of this fuzzy action are the distance to destination and the angle of deviation (Figure 6.9). In the context of a robot-soccer game, the

destination is usually the position of the ball. Several other basic actions are realized on the basis of go-position, including get-position-at-angle and get-ball.

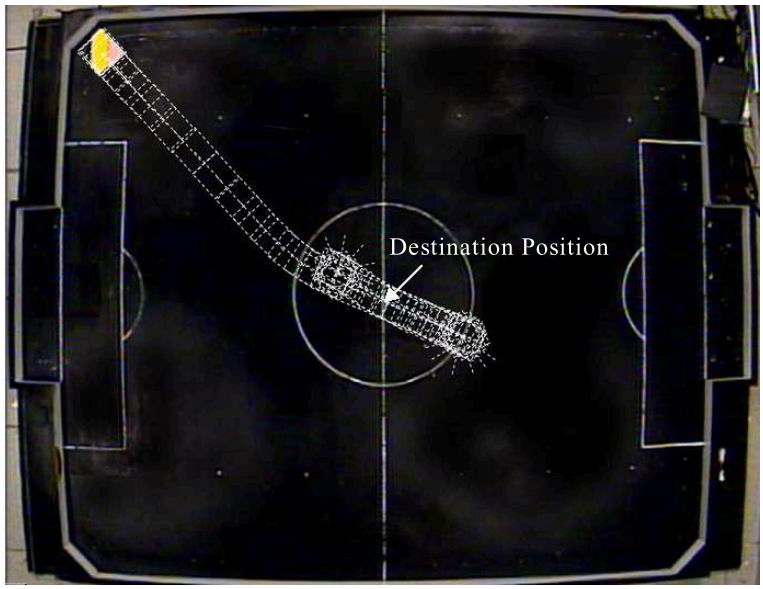
During the tuning process, the robot is commanded to perform the go-position action repeatedly to reach the center of playground from different starting positions. The go-position action's performance before and after tuning on a new playground are compared. Before the tuning, the robot overshoot the desired position, rotated and moved back towards the target, and missed the target again (Figure 6.10(a)). After the tuning is applied, the output fuzzy membership functions for speed are adjusted for every overshoot appeared. As the process goes on, a significant decrease in the magnitude of overshoot is observed (Figure 6.10(b)). Figure 6.11 shows the comparison in terms of the overshoot, rise-time, settling-time and the steady-state error, validating the higher precision and better performance achieved by adaptive tuning.

In the above experimentation, the tuning on go-angle and go-position actions are both triggered by the appearance of overshoot/undershoot. Other performance criteria can be set up to trigger the tuning catering for different scenarios. One good example is the tuning of get-ball action against an opponent. The purpose of get-ball action is to gain possession of the ball in a match. In the process of the get-ball action, the distance between the ball and robot is recorded at each step. Since the ball is a passive element, it is expected that the robot can always reach the ball provided that there is enough time. However, if the robot fails often to get the ball within a reasonable time, that implies that the ball is already snatched by an opponent robot at a faster speed. Under this circumstance, the tuning is triggered to adjust the associated fuzzy membership function. Gradually the robot's speed increases to match that of the opponent's speed. The whole idea is summarized as following:

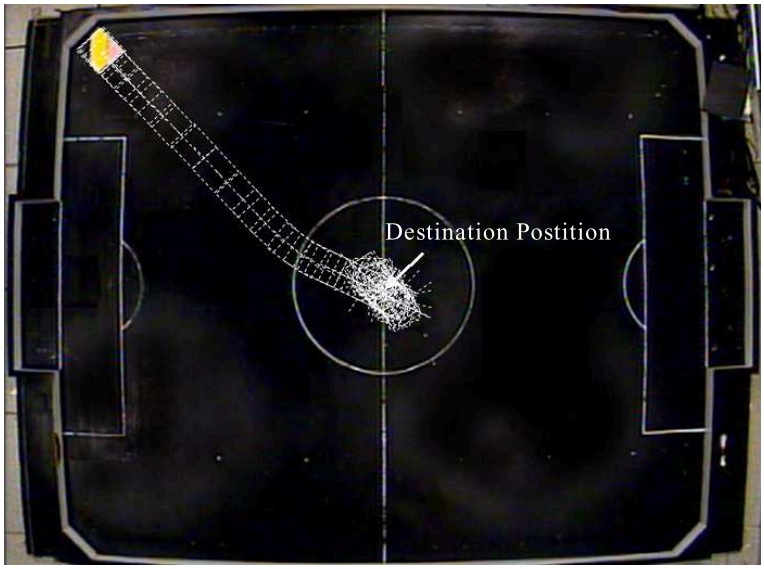
Scenario Opponent robot is faster and more likely to gain possession of the ball

Countermeasure Own robot increases speed

Principle IF *previous-robot-to-ball-dist* $<$ *current-robot-to-ball-dist*,



(a) Go-position before tuning



(b) Go-position after tuning

Figure 6.10: The performance comparison of go-position

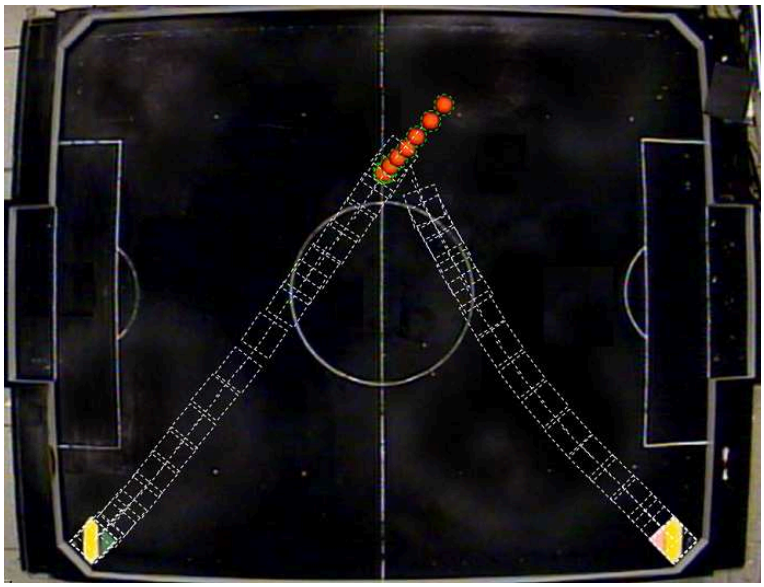
Figure 6.11: The effectiveness of adaptive tuning on go-position action

THEN *broaden the fuzzy subsets*

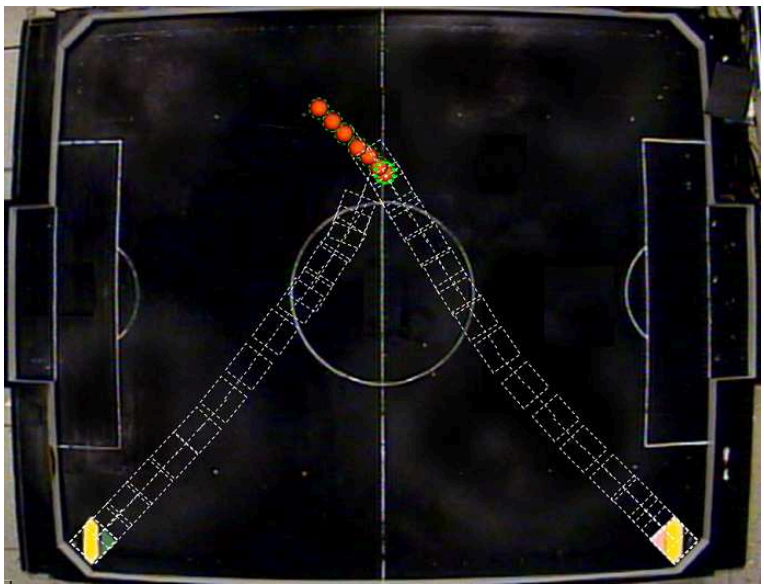
Figure 6.12 (a) and (b) demonstrate the robot's "speed-matching" capability developed by tuning. In the experimentation, two robots are made to go after the ball at equal distance. However, the robot at the left corner of the playground is inherently faster by a small margin (5% or so). At the end of the first run, the robot at the right corner is tuned as it failed to get the ball earlier than the left robot (Figure 6.12(a)). Consequently, after several runs, the right robot becomes quicker and finally gets the ball earlier than its competitor (Figure 6.12(b)).

6.3.2 Robot roles and team strategy

At the higher levels of the fuzzy behavior based architecture of the robot soccer system, the robot roles are defined according to the specific tasks of individual robots. The collective effects of the robot roles represent the team behavior. The team strategy is realized by role selection and assignment. There are four robot roles: the attacker, midfielder, defender and goalie. In the original system [117], a simple mechanism is adopted for role assignment. The attacker and goalie are always present, and the same role is not assigned to two robots simultaneously. As



(a) Get-ball before adaptive tuning



(b) Get-ball after adaptive tuning

Figure 6.12: The performance comparison of get-ball

a result, the fuzzy engine needs only to choose a role for the third robot between midfielder and defender. The adaptive tuning on the upper levels is aimed to adjust the robot's behaviors and the team strategy according to the soccer match situation.

The attacker has a crucial role as it is responsible of pursuing and maintaining the possession of the ball. The attacker takes all the attacking activities and also helps to reinforce the defence. For the attacker, the distance to the opponent goal area is evaluated by a fuzzy membership function. If the robot together with the ball are "near" enough and it is not obstructed, the shoot behavior is activated. In fact, the fuzzy membership function determines a fuzzy "shoot area". Adjusting of the fuzzy membership function changes the size of the "shoot area", and thus affects the attacker's tendency to shoot (Figure 6.13). In this way, the aggressiveness of the attacker is manipulated by adaptive tuning.

In the robot soccer match, the score difference and the match time can be used as the triggering condition for adaptive tuning. The adaptive tuning can take effect when the home-team's score is less than the opponent's. The attacker needs to become more offensive by taking a higher tendency to shoot. This is accomplished by changing the associated fuzzy subsets with the tuning method to expand the fuzzy "shoot area" (Figure 6.13). Basically, the increase in the degree of aggressiveness is roughly in proportion to the score difference. In addition, the attacker becomes even more offensive if its team is unable to catch up on the score despite of the earlier tuning. The time left in the match can also be taken into consideration. The less the time left, the more aggressive the attacker should be. Some triggering conditions for increasing the aggressiveness are listed as bellow:

- IF *Score-gap = Opponent-goal - Our-Goal* $\geq$ *1*
- IF *Our-Goal - Opponent-goal* $>$ *3* (means the opponent team is very weak)
- IF *Score-gap remains* AND *Aggressiveness already increased*

With the increasing aggressiveness, the attacker is observed to perform more shots.

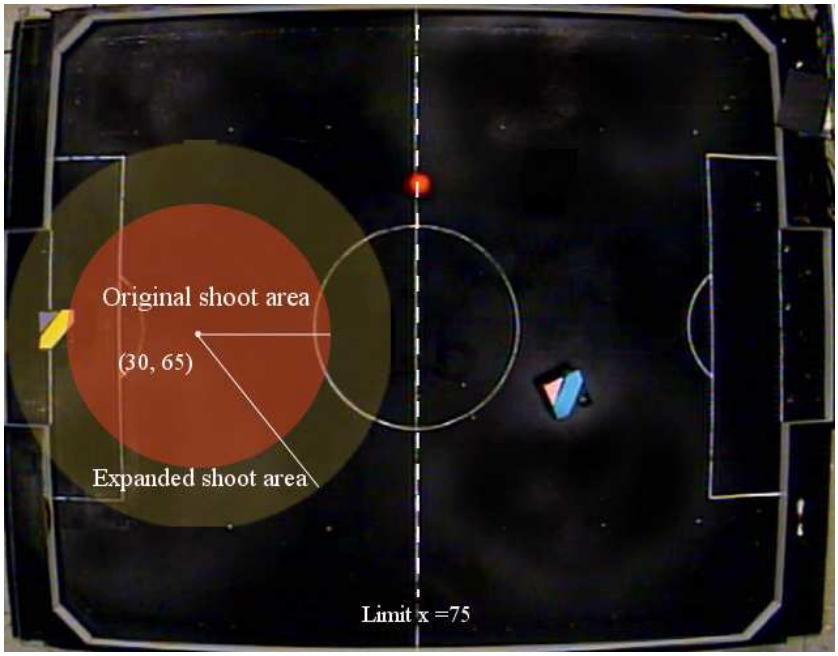


Figure 6.13: The fuzzy shoot area of attacker

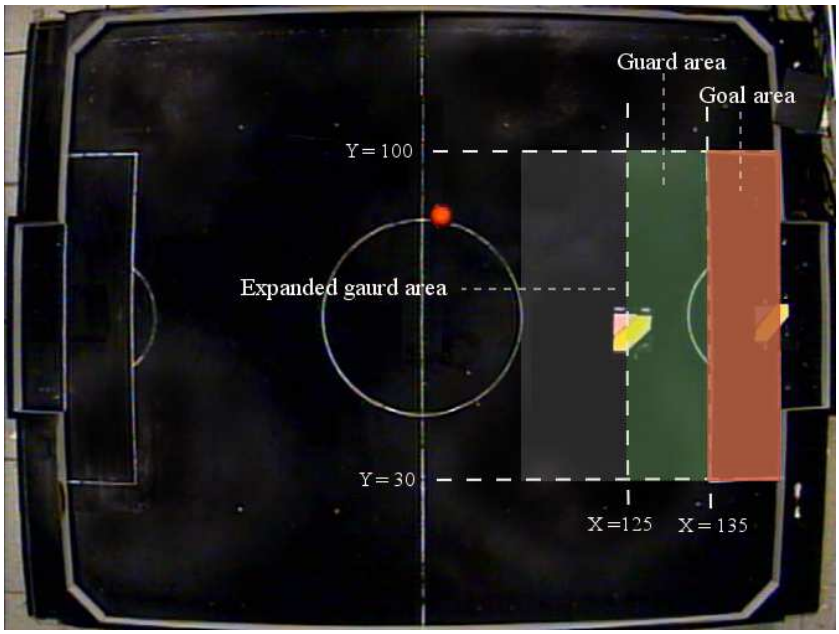


Figure 6.14: The fuzzy defence area of defender

Figure 6.15(a) shows some typical behaviors exhibited by the attacker after adaptive tuning. The robot shoots the ball even though it is still far away from the opponent goal area. After the ball is swept away by the opponent goalie, the attacker quickly performs another shot.

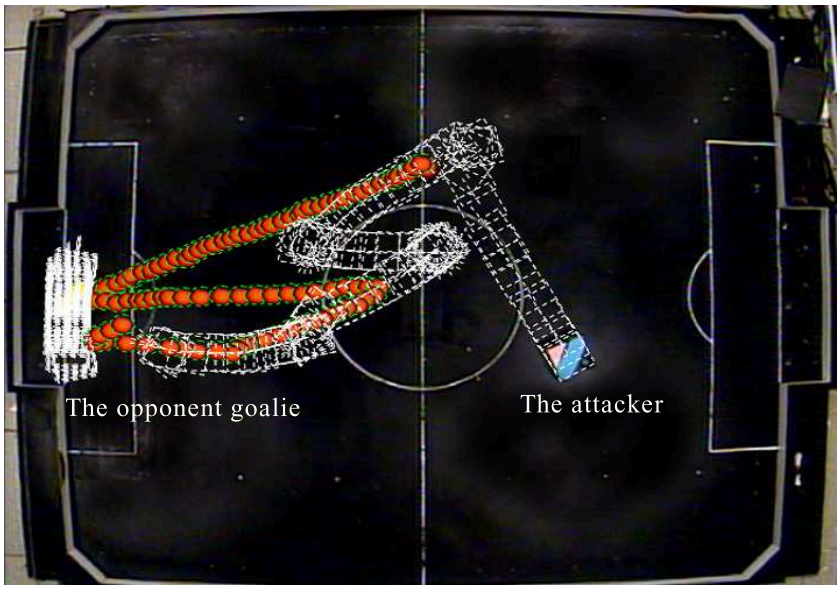
Another robot role subjected to adaptive tuning is the defender, which usually tries to block or sweep the ball which comes to its "guard area" (Figure 6.14). The "guard area" is also a fuzzy area defined by a set of fuzzy membership functions. The adaptive tuning can expand the "guard area" and increases the defensiveness of the defender (Figure 6.14), according to the following conditions:

- IF *Score-gap = Opponent-goal - Our-Goal* $\geq$ *3*
- IF *Score-gap remains* AND *defensiveness already increased*

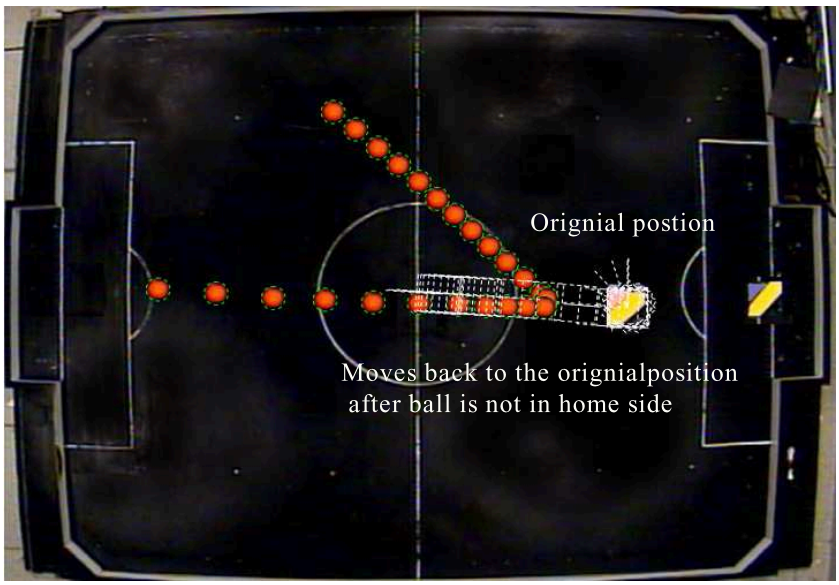
The result is an increased likelihood of defender to sweep the ball away from home goal area. It is also possible to use same conditions to trigger the tuning of attacker and defender simultaneously.

The defender's behavior after adaptive tuning is depicted in Figure 6.15.(b). After the tuning, the defender tries to sweep the ball from the home side of playground as soon as possible. Once the ball is out of the "guard area", the defender returns quickly to its original position within the "guard area" and leaves the task of pursuing the ball to the attacker.

The primary purpose of adaptive tuning on the team strategy level is to increase the team's offensiveness when the team is already in a disadvantageous position. The team strategy is embodied by selecting a role between midfielder and defender for the third robot other than the attacker robot and goalie robot. Since the midfielder is a supportive role to the attacker, the offensive strategy here means a bias towards midfielder during the role selection. The triggering condition for the team strategy is the same as those for the attacker and defender roles. The performance of the team strategy after tuning is not easy to evaluate as the robot roles' behaviors have also been tuned at the same time. However, as a whole,



(a) The attacker after adaptive tuning



(b) The defender after adaptive tuning

Figure 6.15: The performance of adaptive tuning on robot roles

the soccer team has become more aggressive as expected and sometimes greatly increased the chance of wining the match which had an unfavorable beginning.

It is noticed that the more aggressive robot behaviors or team strategy do not guarantee better performance in a match. However, the aggressive behaviors and strategy are in fact the only way out when a team cannot take the upper hand in the match with the standard behaviors and strategy. In the human soccer match, the soccer player's performance is always affected by his physical or mental states, which may fluctuate all the time. Compared to the human counterpart, a robot soccer player usually has a rather stable performance on the match. As a result, if a team is in an inferior position at the beginning, it has scarcely any chance to get back the upper hand if no alterations are made in the behavior or strategy. That is the reason why the adaptive tuning is important and useful although it cannot guarantee a winning strategy.

6.4 Summary of Results

An adaptive tuning method is applied to a fuzzy behavior based system. The performance of the tuned fuzzy actions is quite desirable. All the actions have achieved improvements in terms of less overshoot, and shorter settling-time as well as smaller steady-state error. The robots are observed to move with greater precision and agility. Specifically, the attacker and defender have shown the desired offensive and defensive behavior whenever necessary. Notably, despite the increase in the offensiveness whereby the attacker is made to shoot from a wider range, the accuracy of the shooting behavior is not adversely affected. Likewise, the defender has shown greater ability to sweep the ball out of the home ground. The team strategy, together with the adaptively tuned behaviors and actions, is capable of achieving better performance in a complicated match environment.

The proposed mechanism provides the adaptive tuning ability to the fuzzy behavior based system for handling environmental and system changes which degrade the performance. Compared to the manual tuning, this method saves a lot of efforts

and time. After the manual initialization, the system adapt the fuzzy behaviors to the changed environment by itself. Furthermore, the parameter files of behaviors are able to be loaded on-line. The different fuzzy behaviors' settings catering for different strategies can be pre-defined and be selected whenever necessary. However, it is hard to make the system fully self-tuning. The triggering conditions of the tuning for some of the fuzzy actions, especially those at the higher levels of the behavior hierarchy, are indeed manually set. The computational expenses to monitor and analyze all the triggering conditions for so many behaviors are very high and will seriously slow down the system's speed on vision data processing and robotic control, which are the more important tasks. Limited by the computation capacity of the current hardware setting, the partially manual initialized adaptive tuning is a better way out.

Chapter 7

Evolution of Fuzzy Behaviors in Multi-Robotic System

In the discussed behavior based architecture, complicated interactions of multiple robots are decomposed into modular behaviors in different complexity levels. The fuzzy logic approach brings in human-like reasoning to the behavior construction, selection and coordination. To facilitate the system's adaption to different changes, some adaptive tuning mechanism (Chapter 6) has been incorporated. However, there are limitations in those tuning methods. The heavy computational load prohibit it from a system-wide application. Furthermore, they are just tuning methods based on the already developed system, whose effectiveness is quite limited in the system developing stage.

In this chapter, the evolution of the fuzzy behavior based architecture is discussed. Genetic algorithm is used to evolve various behaviors in the fuzzy behavior based architecture. The behaviors at different levels of the architecture hierarchy are generated and improved through evolution. At the lowest level, the evolved fuzzy controllers enhanced the smoothness and accuracy of the primitive robot actions. At the higher level, the individual robot behaviors have become more skillful after the evolution. At the topmost level, the evolved group behaviors have resulted in aggressive competition strategy. The simulation and real-world experimentation on a robot soccer system justify the effectiveness of the approach.

7.1 Introduction

Fuzzy logic can be involved in almost every aspect of the behavior based system: from the implementation of robot actions and behaviors, the behavior coordination for individual robot, right up to the role building and role assignment. However, in the traditional way of developing a fuzzy system, the expert's knowledge of the system is necessary. The performance of such a fuzzy system is usually human dependent and sometimes far from optimum. To further improve and to make the system adaptive in nature, evolutionary methods are often used to complement the fuzzy behavior based system [64, 118, 119].

In this chapter, an evolutionary fuzzy behavior based approach for a multi-robotic system is explored. The genetic algorithm is used to evolve the fuzzy behavior based architecture. GA can work efficiently even without the comprehensive knowledge of the system model. This future makes it suitable to work with complicated systems like the multi-robotic system. For the developed hierarchical fuzzy behavior based architecture GA is utilized at various levels. The fuzzy behaviors are improved through the evolution process.

Different from the works that have been explained in the Chapters 5 and 6, the evolution process of fuzzy behavior based robot soccer system can not be performed on the real world set-up. A simulator platform for robot soccer system is carefully developed to provide a virtual environment as close to reality as possible. The evolution process is carried out and noticeable improvements in the performance of robot behaviors are observed in both the simulation and real-world experimentation. If there are internal changes (such as changes in robot kinematic features) or external changes (such as changes in environment characteristics), suitable behaviors can be reconstructed and evolved.

The fuzzy behavior based architecture for multi-robotic system is introduced in Section 7.2. The mechanism of evolving fuzzy behavioral system with GA is discussed in Section 7.3. The robot soccer system and the developed simulator are briefed in Section 7.4. Section 7.5 describes the simulation study and experimental

implementation. Conclusions and discussions are summarized in Section 7.6.

7.2 Fuzzy Behavior Based Architecture for Multi-Robotic System

The fuzzy behavior based architecture for multi-robotic system is discussed in the Chapter 5. However, it is worthwhile to provide a generalized overview in this section.

In the construction of behavior based architecture, a hierarchy of roles, behaviors and primitive actions is set up to start with. The multi-robotic system is decomposed according to such a hierarchy. How to build the hierarchy depends heavily on the nature of systems at hand. The general idea is to use the complexity of behaviors as a guideline. Complex roles and behaviors are put in the higher level of hierarchy. They are decomposed into simpler, modular and manageable sub-behaviors, from the top to the bottom of the hierarchy. The number of levels in the behavior architecture mainly depends on the complexity of the system. The objective and importance of each behavior might also affect the behavior's relative position in the hierarchy. The structure can be flexible as long as it facilitates the realization of the behavioral system.

Figure 7.1 shows a typical layout of behavioral architecture for multi-robotic system which has four levels. At the top of the hierarchy is the group behavior representing the team strategy. In a multi-robotic system, the "team" refers to a group of robots having cooperative relationship and serving a common purpose. As to one "team" of robots, other robots are treated as part of the environment, while they might be another "team". The group behavior of a team is constructed by the collective effects of individual robots' activities. Based on their objectives, individual robots are expected to display different behavior patterns. Roles are defined to decompose the group behavior and are located in the second level. At each time step, the fuzzy controller at the top level assigns a certain role to each

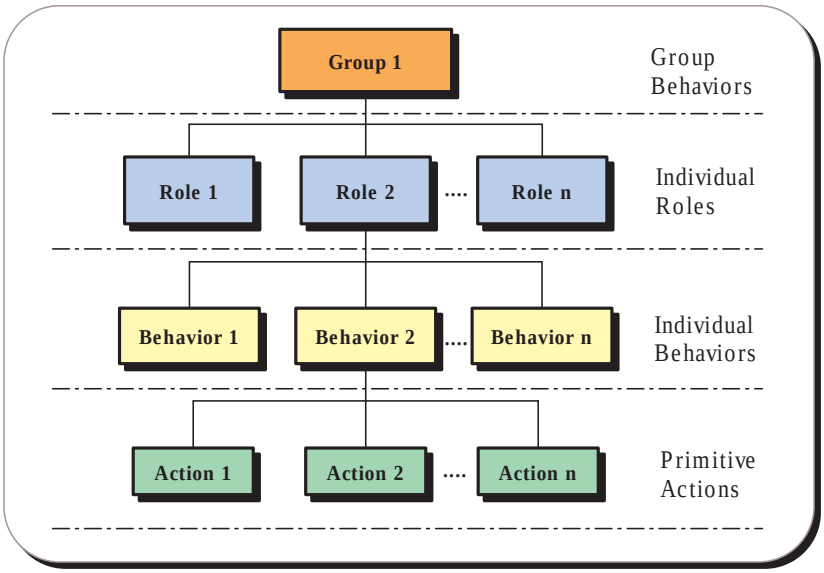


Figure 7.1: The behavior based architecture

robot. To fulfill its role, a robot needs to perform some task-specific behaviors. Those individual behaviors are defined in the third level. Behaviors in this level are designed to carry out simple tasks. Some behaviors are reactive in nature, which are in direct response to the environment stimulus. Other behaviors are strategy based, driven by the system's inherent strategy and objectives. The behaviors are further decomposed into primitive actions, which form the bottom level of the hierarchy. Primitive actions are usually the basic motions which are easy to realize. The primitive actions are components to construct complicated behaviors.

On the basis of the hierarchy structure, fuzzy logic is applied to realize the behaviors at each level. Normally there are different requirement for the controllers at different levels. For primitive actions, the associated fuzzy controllers are required to achieve accurate and smooth robot motion. The behaviors at higher levels are constructed based on the lower level behaviors. The fuzzy controllers at the higher level are mainly used for decision making and behavior coordination. Based on the environmental information and the relevant objectives assigned by the upper level, the fuzzy controllers activate suitable behaviors at the lower level.

Obviously, behavior coordination is one of the main objectives of the developed system. Different actions/sub-behaviors are combined into higher level behaviors,

or behaviors into roles. Two general fuzzy behavior coordination mechanisms are utilized in the behavior based architectures: the fuzzy rule base coordination [54] and, the activity and action contribution [55].

The fuzzy rule base coordination method uses a fuzzy rule base to select a suitable role, behavior or action. For the robot soccer system, the rule base contains a series of fuzzy rules, which assesses the conditions within the playground and chooses a suitable action to serve the team's game strategy. Basically the rule base takes the role of an decision maker to resolve the behavior conflicts and achieve coordination. This method adopts a top-down decision making approach. The fuzzy rule base coordination method is used in role assignment, behavior coordination for roles and action coordination for deliberative behaviors.

Contrary to the rule base coordination method, the activity and action contribution method uses a bottom-up approach in behavior building. The fuzzy inference engine of each sub-behavior provides a pair of activity and action values. Action value represents the output from the fuzzy inference engine and activity value represents the degree of the sub-behavior's contribution to the upper-level behavior. The upper-level arbitrator simply reviews the activity values proposed from the lower level and calculates the output behavior with a pre-defined algorithm, such as the center–of–area (CoA) method. Generally, the activity and action contribution method is used for action fusion and action selection within the reactive behaviors.

The hierarchy architecture is flexible and modular. Other control methods can be utilized along with fuzzy logic to form a hybrid system. The option of setting up a hybrid system is quite open and is not discussed in this thesis. The focus of this work is to improve the fuzzy behavior based system with the incorporation of the evolutionary algorithms.

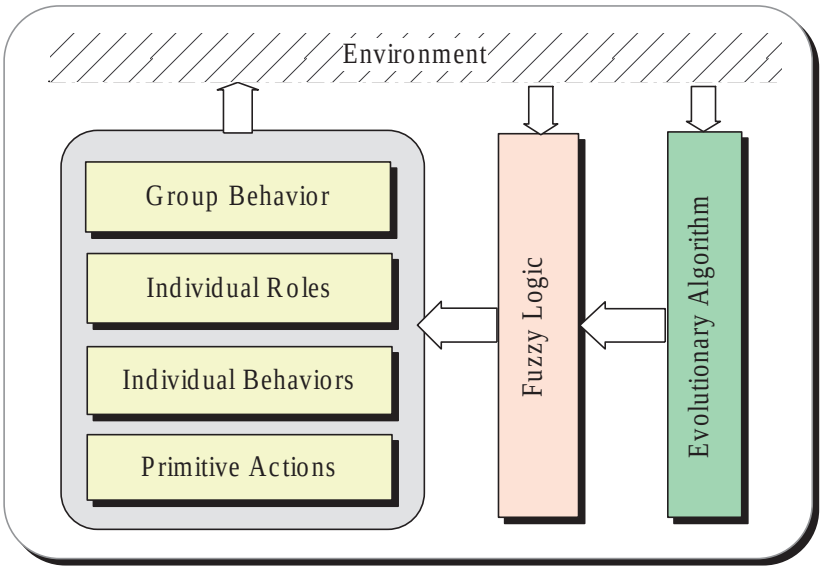


Figure 7.2: The evolution of fuzzy behavior based architecture

7.3 Evolution of the Fuzzy Behavior Based System

In the behavior based architecture discussed in Section 7.2, fuzzy logic is applied in almost every aspects, from the primitive actions' realization and behavior coordination to the overall role building and assignment. Thus, it is clear that the fuzzy logic controller's quality is crucial to the performance of the system.

On the other hand, the design of fuzzy logic controller is usually an experience based work. Although the time-consuming "trial and error" approach can be taken to fine-tune the system, the human dependent fuzzy controllers still have space for further optimization. Meanwhile, any change in the system configuration, such as the difference in environment features or robot's kinematic characteristic, requires relative modification/tuning in the fuzzy controller. Incorporating the evolutionary method into the development or modification of fuzzy controller for behavior based system seems to be a suitable way out. The evolutionary mechanism paves the way for both improving the performance and reducing the human workload.

The genetic algorithm is an optimum searching algorithm which is useful in knowledge acquisition. The genetic algorithm for fuzzy controller development and

optimization usually focuses on two aspects: the rule base and the membership functions reforms [74, 120, 121, 122, 123], and these two approaches are explored in this work.

For the primitive actions at the bottom level of hierarchy, the fuzzy rules are often simple and straightforward. The possible improvement on fuzzy rules is quite limited. Under such circumstances, the evolution of membership function seems to be more promising at the bottom level of the architecture.

As mentioned in Chapter 6, the triangular membership function is prevalently adopted in the system. The triangular membership functions can be tuned in two ways. Shifting the entire triangle can modify the position of the associated fuzzy subset in the universe of discourse. Depending on the necessity, restrictions can be imposed, such as limiting the range, step-size of shifting or specifying the subsets under modification. In the second method, changing the base points of the triangle modifies the span of the functions. In fact, both the methods move the "center of gravity" of the associated membership function. According to the extent of movement caused, the first tuning method seems to have significant impact on the membership function while base point tuning has a relatively moderate impact. These two methods are compatible with each other and can be used together.

On the other hand, the evaluation of the performance of fuzzy membership function is meaningful only when the function is associated with fuzzy rules. The rule base is the kernel of a fuzzy controller. For the fuzzy behavior architecture, the rule base optimization is usually applied to higher level robot behaviors and role assignment.

A typical fuzzy rule often consists of antecedents (or premise), consequents (or conclusion) and fuzzy relations. All parts of a fuzzy rule are suitable for evolution and, the choice is made based on the requirement and characteristic of the problem under consideration.

Since fuzzy logic is applied everywhere in the behavior structure, the evolution of the fuzzy controllers can make the whole system evolvable (Figure 7.2). However,

it is reasonable to focus on those fuzzy behaviors which are most important to the system's performance and those which have greater margin for improvement. Such criteria serve as guidelines in the case study.

To apply the genetic algorithm, parameters of the membership functions and/or the fuzzy rules are encoded as individual chromosomes. Various genetic operations and fitness functions are designed for different fuzzy controllers under evolution. In this work, GA evolution is performed in a robot soccer simulator software and the results are validated by real world experimentations on a robotic soccer setup.

7.4 The Robot Soccer System

7.4.1 Fuzzy behavior based architecture of robot soccer system

The case study of evolution of fuzzy behavior based multi-robotic system is performed on a robot soccer system. The fuzzy behavior architecture has been set up for the robot soccer system in Chapter 5 [117]. Figure 7.3 shows the hierarchy of fuzzy behaviors of different complexity, as well as the coordination mechanisms. The fuzzy rule base coordination method is widely used in each level. The activity and action contribution method is used only for reactive behaviors, such as "avoid wall" and "shun robot". The goalie adopts a straightforward "if–else–if logic" coordination method for simple but quick reactive motions.

The above fuzzy behavioral architecture is realized on the real world set up in Chapter 5. Fuzzy techniques are used almost everywhere in this system. The only exception is the coordination of the goalie's behavior which uses the simple if-else-if rules. Most of the primitive actions and behaviors show acceptable performance. However, not all of them are fully optimized, especially for the higher level behaviors, which has a prominent role in decision making.

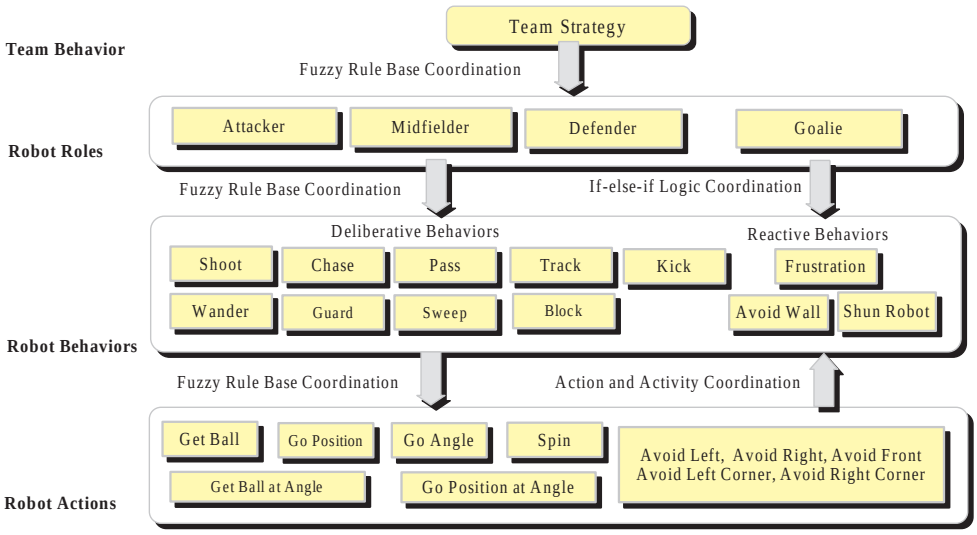


Figure 7.3: The behavior architecture of the team of soccer robots

7.4.2 Robot soccer system simulator

The evolutionary computations are performed on a robot-soccer simulation system (Figure 7.4) developed as a test-bed for multi-agent system research. The use of a simulation environment is necessary since performing GA evolution directly on an actual robot soccer system is impractical. In particular, the evaluation of the fitness of each individual in every generation requires the robots to perform a set of benchmark actions repeatedly. Even with the moderate population size and number of generations to run, the process is too time-consuming, not to mention the strain on the robot hardware and limitations on the longevity of the robots' batteries.

Based on the mathematical model of soccer robot discussed in Section 4.4, the simulator for robot soccer system is developed in Microsoft Visual C++, using the OpenGL library for visualization. The simulator models many of the environmental conditions of a real robot soccer game, such as the interactions between the robots, the ball, and the boundaries of the soccer field. Robots are modeled as block masses with two-wheel differential drive, and the ball as a circular sliding mass, subject to rolling friction. Collisions between robots and the ball comply with the relative orientations of their colliding surfaces. The effects of collisions are calculated by

Figure 7.4: The robot soccer simulator

solving the vector equations for the conservation of momentum, giving rise to accurate and visually-realistic characteristics. In addition, uncertainty in positions of objects due to noise in visual perception are simulated by some random factors in position data. To achieve the desirable verisimilitude, all these characteristics are adjustable by changing the parameter settings in the simulator. Such features are especially useful when systems with different configurations (such as ball and robot masses) are to be simulated.

The simulator menus are activated with a right click on the graphic. They provide convenient access to different working modes. The "game mode" submenu includes all the default scenarios in a robot soccer competition, such as "Kick off", "Free-ball" and "Penalty". The "Train mode" submenu is defined particularly for the evolution of behaviors in different levels. The "Demo mode" menu consists of demonstrations of some basic actions. Simulation settings such as toggling on and off the path tracing in simulator are set in the "Options" menu. The basic functions, including "Start" and "Quit", are listed in the main menu.

The fuzzy behavior based architecture which is used in the real-world robot soccer system in Chapter 5 [117] is migrated into the simulator to control the virtual robots. For the evolution purpose, the proposed genetic algorithms are also incorporated in the simulator as an extension. The evolved system is compared

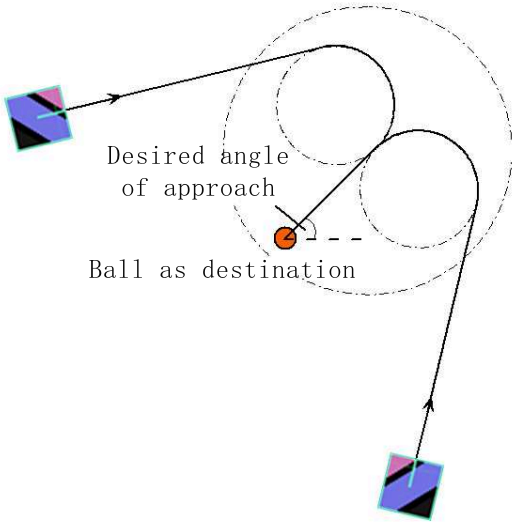


Figure 7.5: Go-position-at-angle action

with the original fuzzy logic control system in Chapter 5 [117] and verified on the real-world platform.

7.5 Simulation and Experimentation

7.5.1 Evolution at the primitive behavioral level

The primitive actions are the most simple actions in the system (Figure 7.3). For the basic components of all the complicated behaviors, the accuracy and smoothness of the primitive actions are obviously important. As discussed in Section 7.3, evolution at this level is focused on membership function tuning. Due to the simplicity of the rule bases for primitive fuzzy actions, it is more useful to evolve the membership functions alone.

Among the primitive actions (Figure 7.3), the "go" actions are most basic. The "get–ball" actions are derived from the "go" actions. Among the "go" actions, "go–angle" and "go–position" are simpler than the "go–position–at–angle". The "go–angle" is for on-the-spot turning, which the robot makes to orientate itself to a certain angle. The "go–position" is a generic action the robot takes to reach a desired position with the shortest path. The robot turns towards the destination,

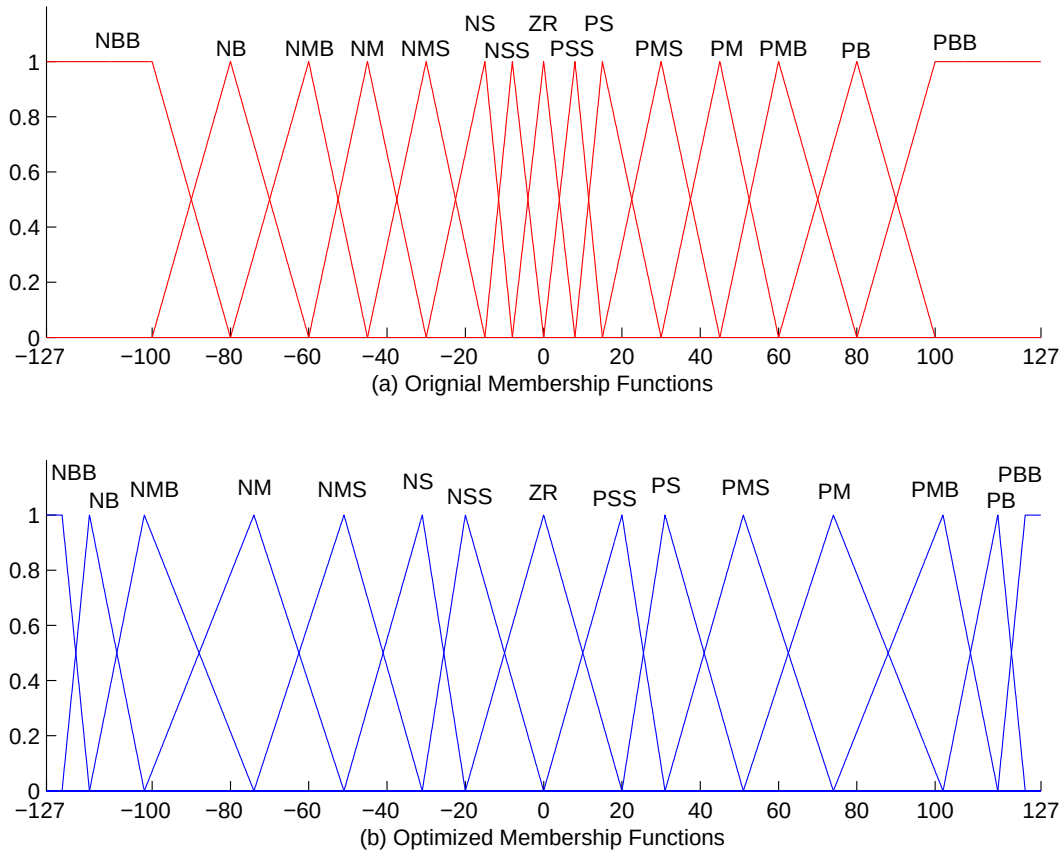
and then moves directly towards the target point. The "go–position–at–angle", whose goal is to reach a desired position with a desired orientation (Figure 7.5), requires more skill and has more influence on robot performance than the first two. The accuracy of "go–position–at–angle" affects the robot's ability to control the ball and is important for behaviors like "shoot" and "pass". As a result, the "go–position–at–angle" action is selected to evolve.

For the "go–position–at–angle" action, fifteen linguistic values are used for the output of fuzzy controller while only three linguistic values are used for the input. As a result, the output membership function set is more complicated than the input set. The improvement space for the rather simple input membership functions are quite limited. The evolution processing is focused on the tuning of output membership functions for the "go–position–at–angle" action.

Fully overlapped triangular membership functions are used for the fifteen fuzzy linguistic values which represent the output of the fuzzy controller for "go–position–at–angle" (Figure 7.6). Due to the fully overlapped membership functions, tuning the peak point is equivalent to changing the base points of the nearby membership functions. A set of fifteen parameters (i.e. the peak points) is enough to define the shapes and positions of the fifteen membership functions. Integer coding method is used to encode one parameter set into one individual chromosome.

To evaluate different parameter settings, the robot is made to perform the "go–position–at–angle" action to reach the destination from five different starting positions and orientations. For each starting point, three runs are performed with three speed settings (fast, medium and slow). Two factors are considered in performance evaluation: the difference between the robot's final and desired orientations at the destination point (angle error $\Delta\alpha$) and the number of steps (n) used in each run to reach the destination. The angle error is a measure of the accuracy of robot action. The steps taken in each run is a measure of smoothness and swiftness associated with the robot actions. The fitness function F_{go} used in simulation is defined by Equation (7.1).

$$F_{go} = u \cdot \frac{v}{Sum(\Delta\alpha) + v} \cdot \frac{w}{Sum(n) + w} \tag{7.1}$$



Linguistic values: NBB(negative big big), NB(negative big), NMB(negative medium big), NM(negative medium), NMS(negative medium small), NS(negative small), NSS(negative small small), ZR(zero), PSS(positive small small), PS(positive small), PMS(positive medium small), PM(positive medium), PMB(positive medium big), PB(positive big), PBB(positive big big)

Figure 7.6: The membership functions for "go–position–at–angle"

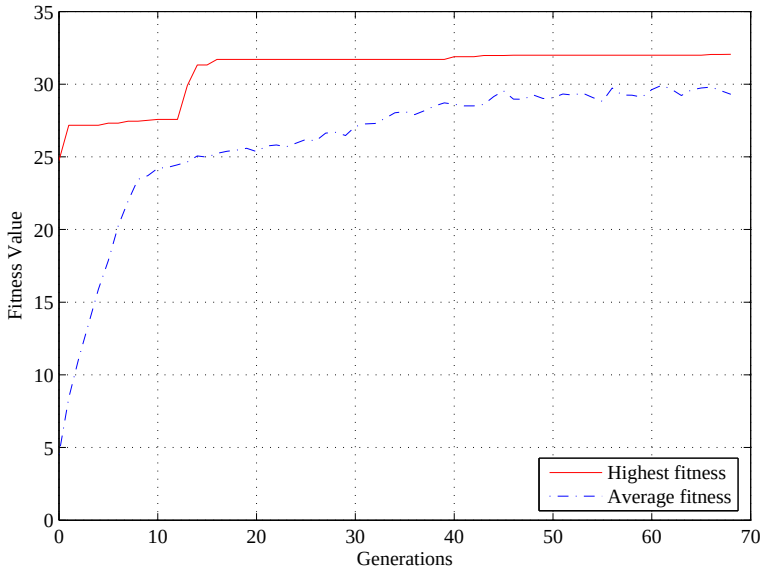
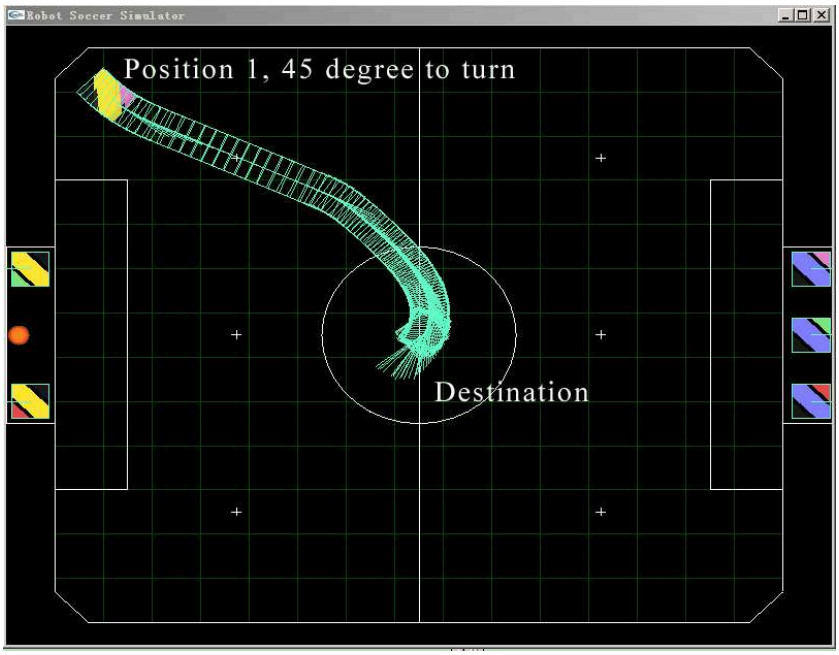


Figure 7.7: The GA process for "go–position–at–angle"

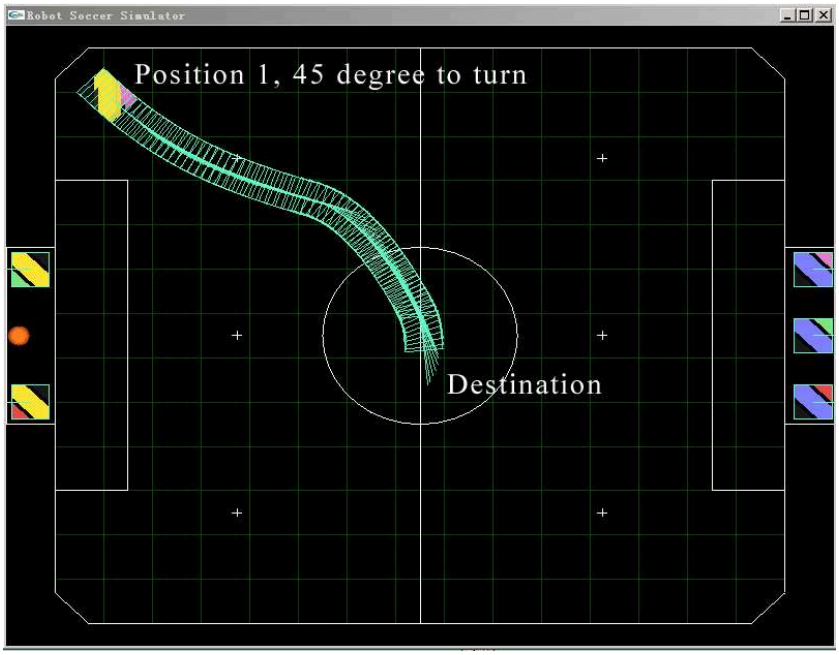
where u, v and w are integer constants, $Sum(\Delta\alpha)$ is the sum of the angle errors and, $Sum(n)$ is the sum of the steps in all the runs with different starting points and speed settings. u is merely a scaler to regulate the fitness range. v and w adjust the weights of $Sum(\Delta\alpha)$ and $Sum(n)$ in F_{go}. The values of u, v and w are heuristically decided. In this work, u, v and w are considered as 100, 10 and 100 respectively. Due to a smaller v value, the angle error has more influence than the "steps" in F_{go}. The accuracy of the action is more important and has higher priority for consideration in the optimization process.

As to the GA process, the population size is fixed at 300 individuals. Stochastic universal sampling (SUS) [124] with elitism selection is adopted. Only one elite individual is selected and preserved for the following generation. Uniform crossover [125] is used with a crossover probability P_c of 0.8. Mutation takes place at two points with a probability P_m of 0.05. The evolution process usually converged around 60 generations (Figure 7.7).

The membership functions using the original parameter setting [117] are plotted in Figure 7.6(a). The original membership functions have a fitness value less than 10 in the simulation, while the fitness value after evolution is around 30 (Figure

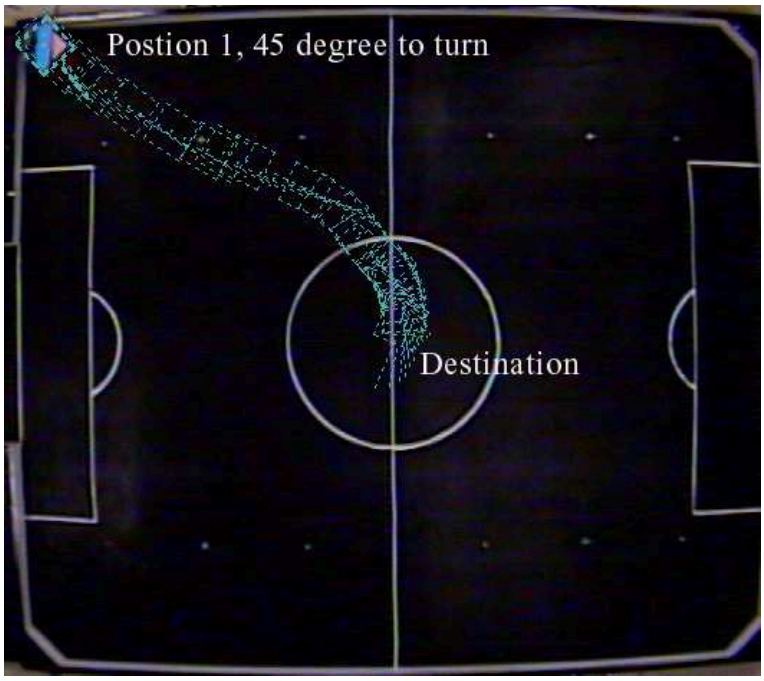


(a) Trajectory of original system

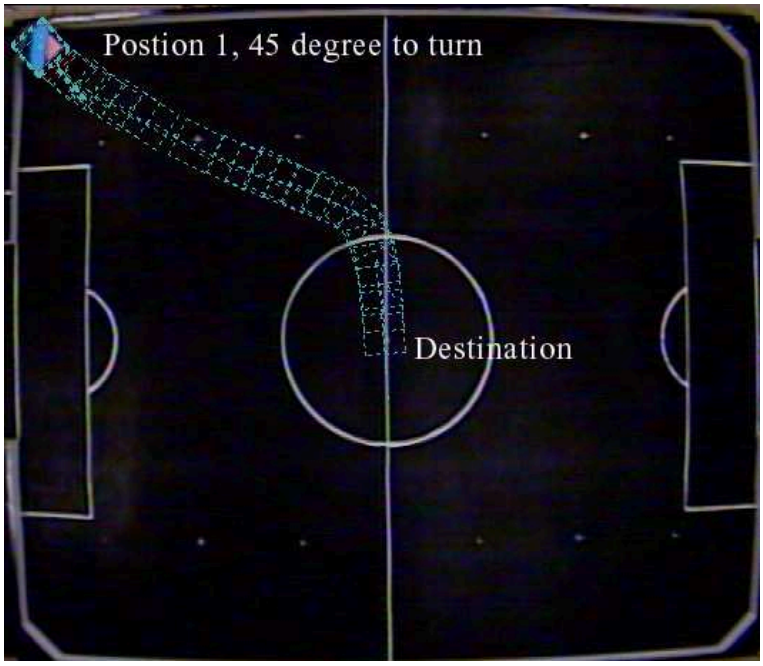


(b) Trajectory of evolved system

Figure 7.8: Simulation performance of "go–position–at–angle" (position No. 1)

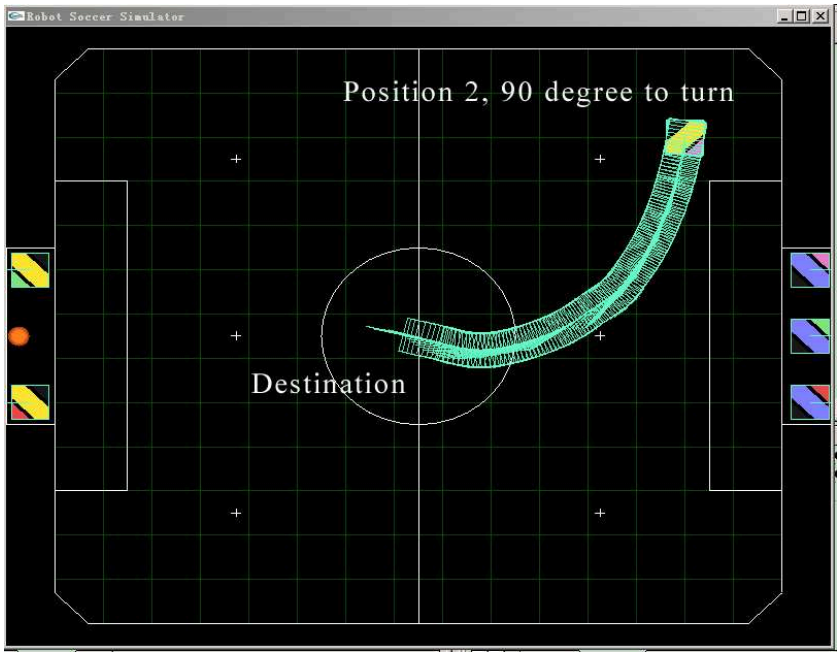


(a) Trajectory of original system

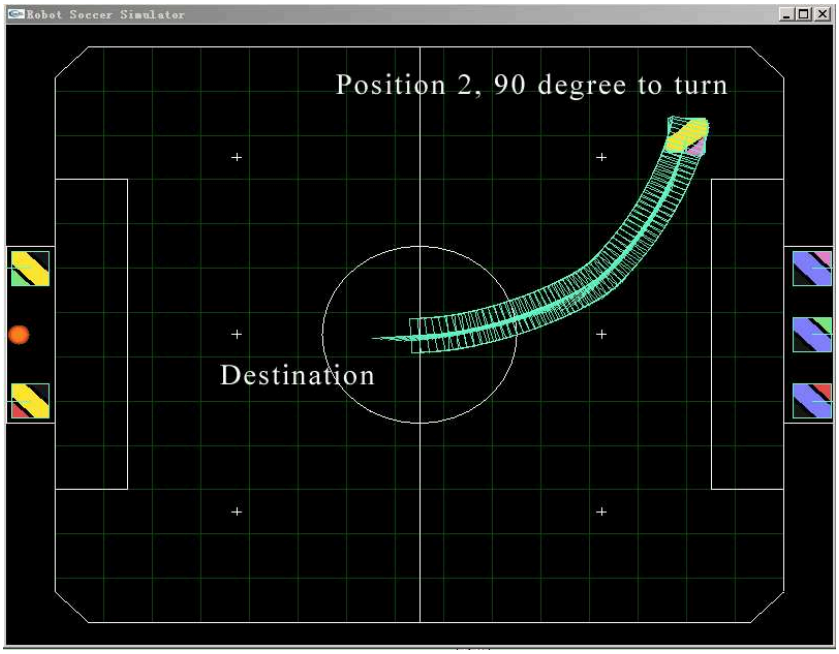


(b) Trajectory of evolved system

Figure 7.9: Real world performance of "go–position–at–angle" (position No. 1)

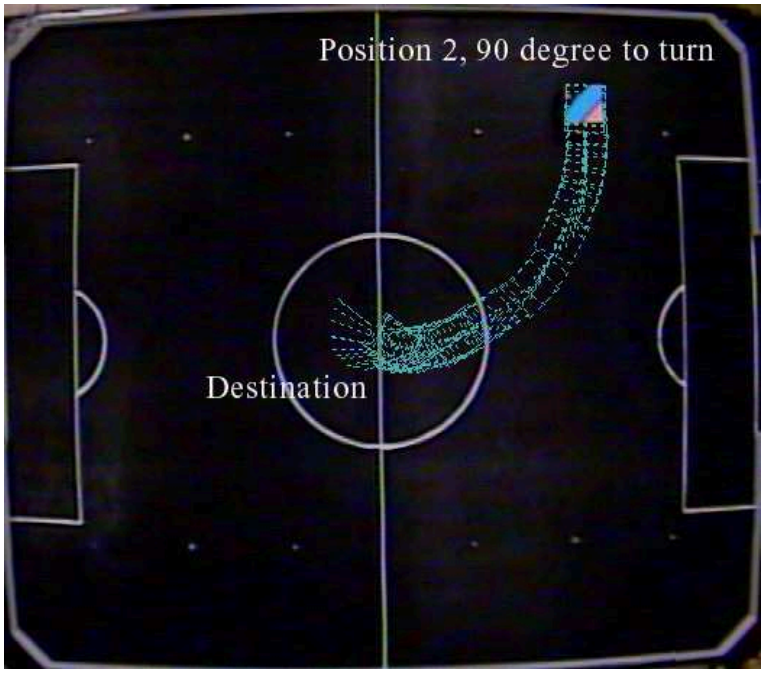


(a) Trajectory of original system

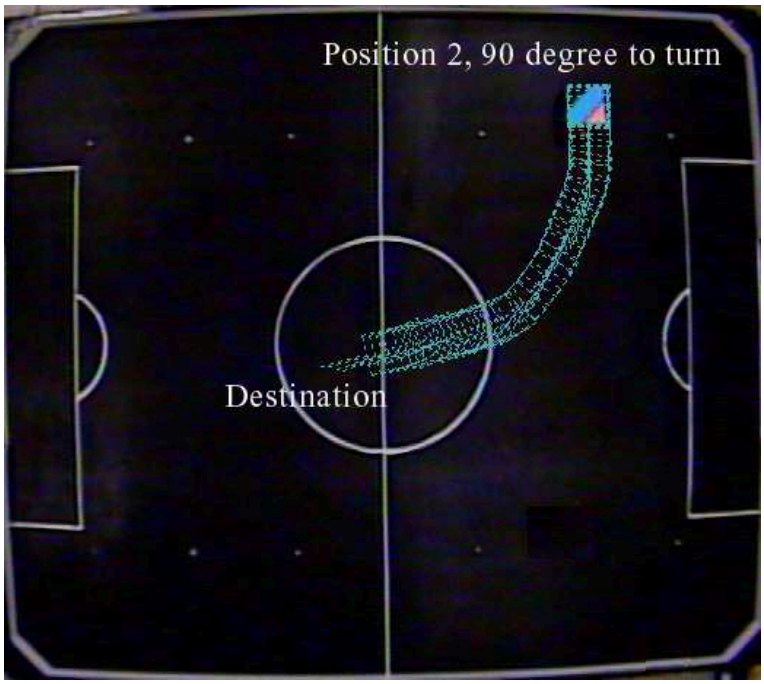


(b) Trajectory of evolved system

Figure 7.10: Simulation performance of "go–position–at–angle" (position No. 2)

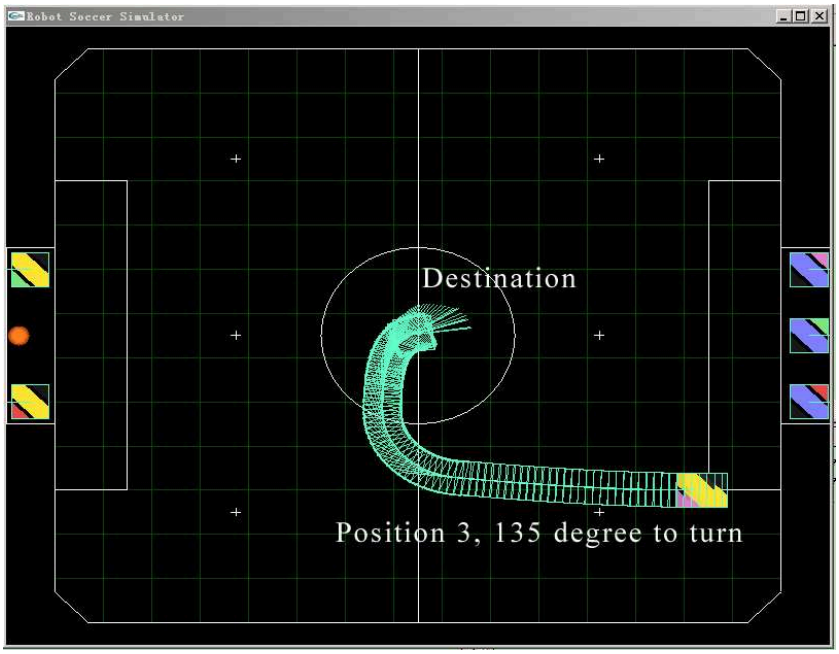


(a) Trajectory of original system

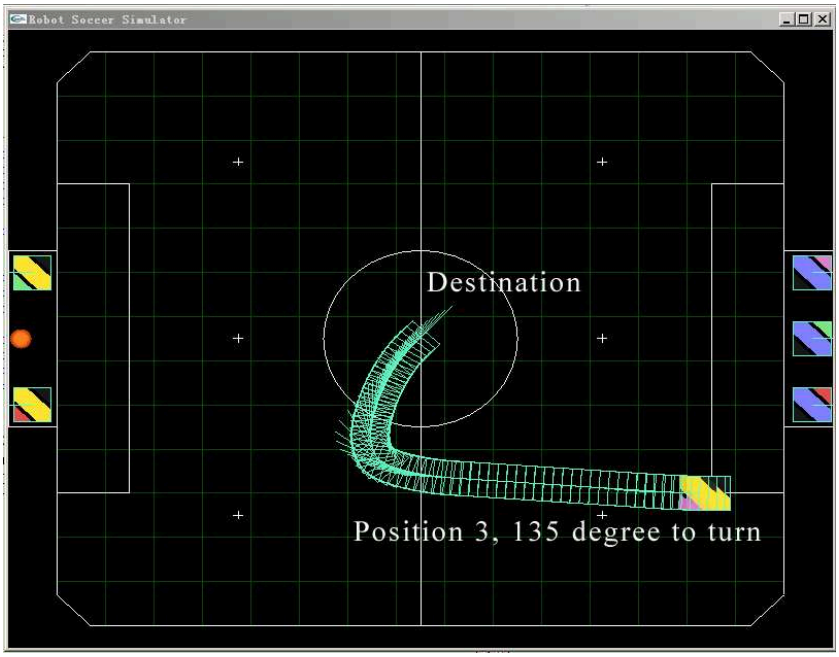


(b) Trajectory of evolved system

Figure 7.11: Real world performance of "go–position–at–angle" (position No. 2)

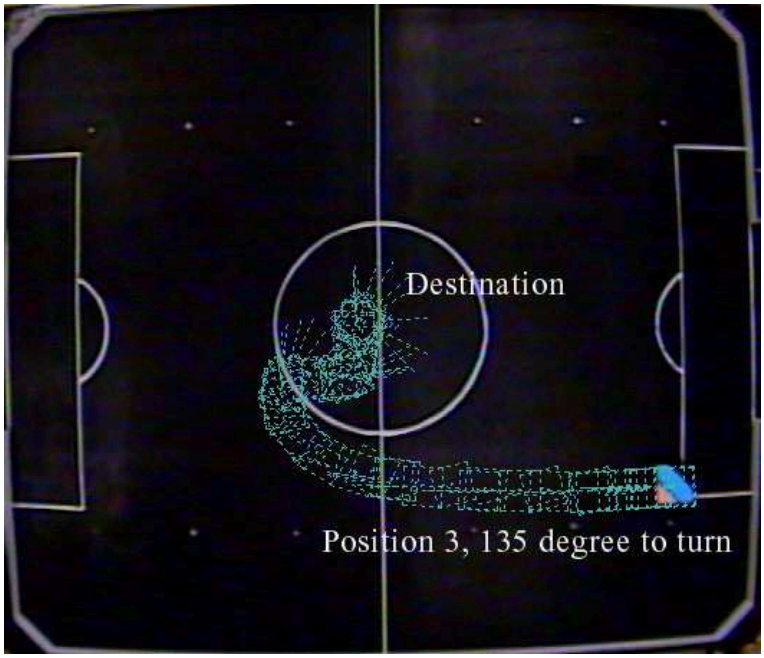


(a) Trajectory of original system

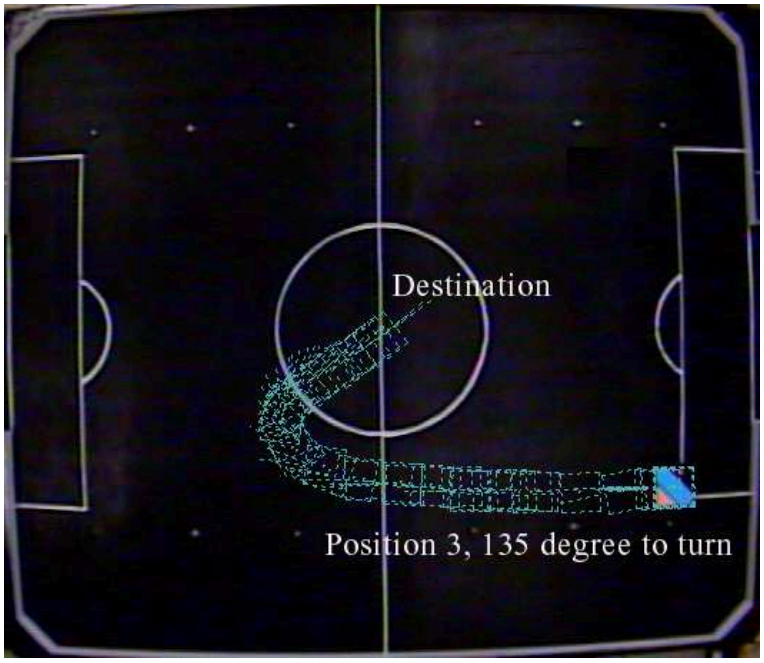


(b) Trajectory of evolved system

Figure 7.12: Simulation performance of "go–position–at–angle" (position No. 3)

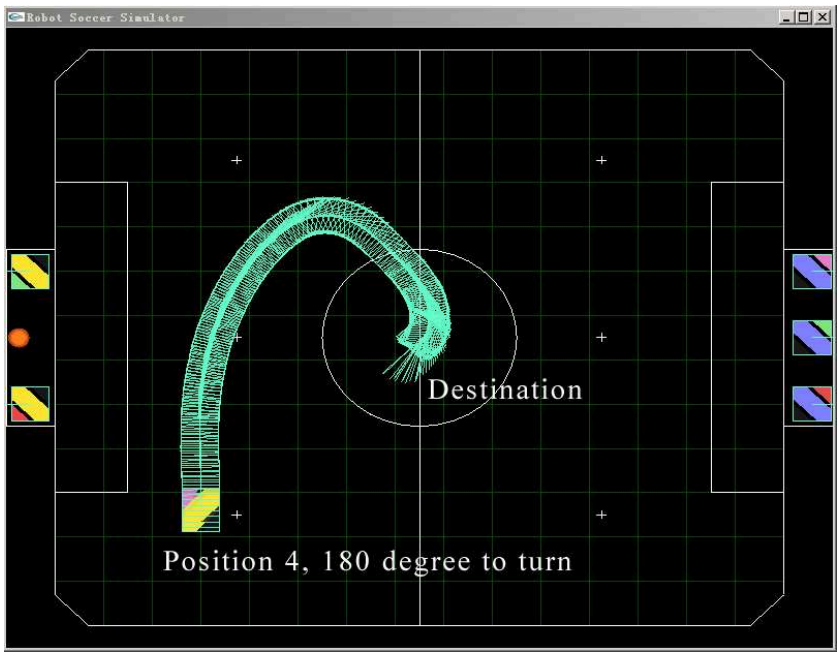


(a) Trajectory of original system

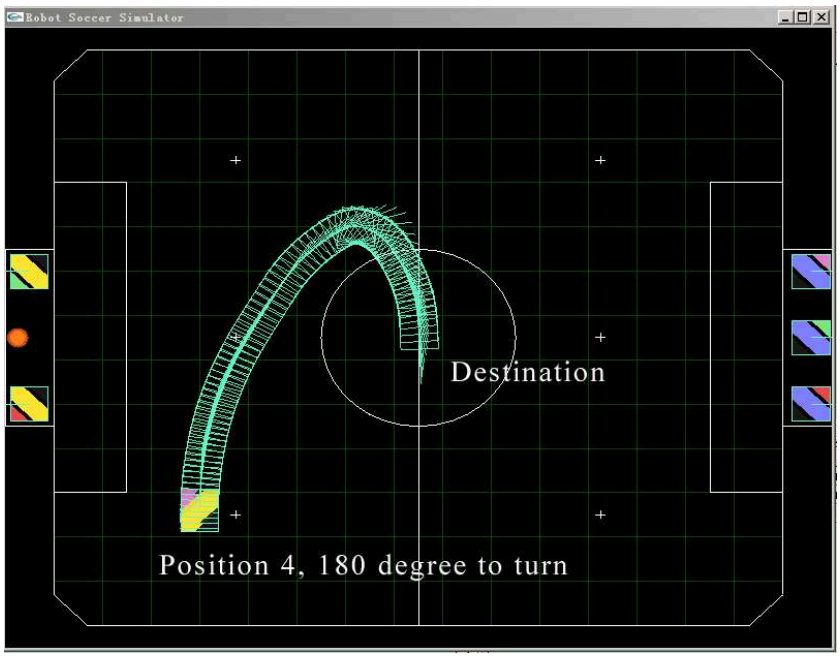


(b) Trajectory of evolved system

Figure 7.13: Real world performance of "go–position–at–angle" (position No. 3)

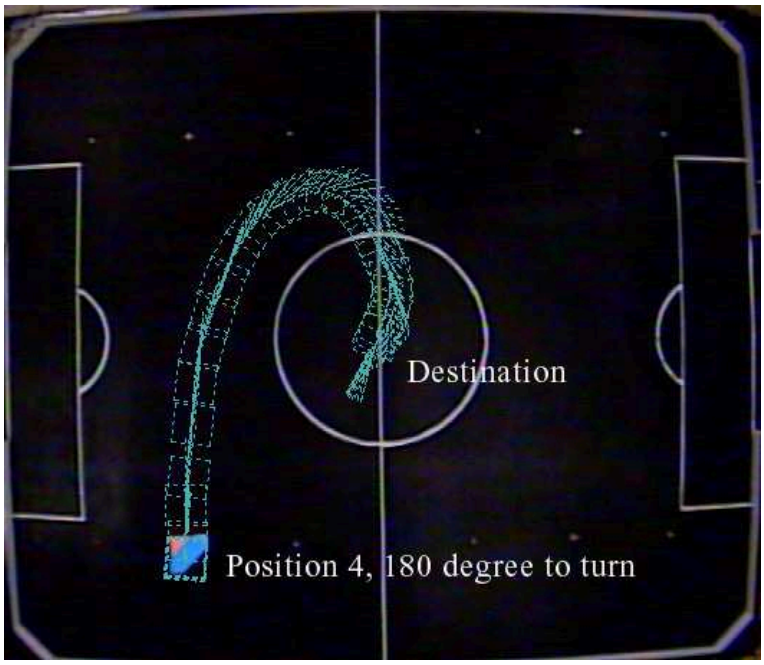


(a) Trajectory of original system

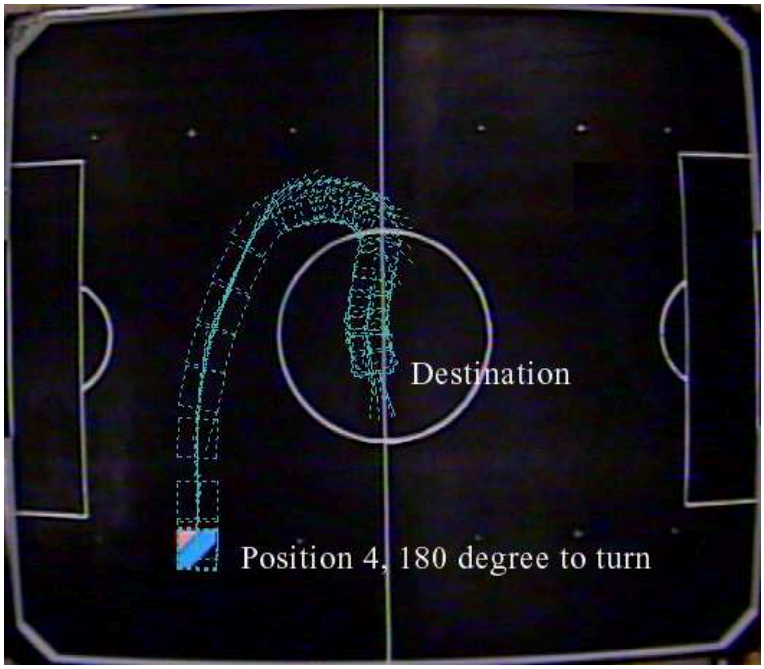


(b) Trajectory of evolved system

Figure 7.14: Simulation performance of "go–position–at–angle" (position No. 4)



(a) Trajectory of original system



(b) Trajectory of evolved system

Figure 7.15: Real world performance of "go–position–at–angle" (position No. 4)

7.7).

The performance of the optimized membership functions (Figure 7.6(b)) is validated in simulation and real world experimentations. The evolved system is compared to the system with original membership functions (Figures 7.8, 7.10, 7.12, 7.14, 7.9, 7.11, 7.13, 7.15). In these experimentations, robots perform "go–position–at–angle" from different starting points numbered from 1 to 4. The destination is the center of the field. The differences between the robot starting orientation and the desired final orientation are 45, 90, 135 and 180 degrees respectively in each case. In Figures 7.8, 7.10, 7.12, 7.14, it is clear that compared to the original system, the evolved system achieved significant improvements on the performance in simulation experiments. The original system's final angle errors are quite obvious, while the evolved system decreased the angle errors close to zero. Furthermore, the trajectories of the evolved system are smoother than that of the original system. The same conclusion can be drawn from the real world experimentations (Figure 7.9, 7.11, 7.13, 7.15), although the trajectories are slightly different from those observed in the simulation.

On the other hand, though the performance of the evolved system is found better than the original system, the results of real world experimentations are not perfect yet. Starting from the starting position 4 with 180 degree to turn (Figure 7.15(b)), the evolved system overshoots the angle difference before reaching the destination and makes the trajectory not so smooth while adjusting the orientation. Since this is not observed in the simulation, it is reasonable to say that those flaws in performance are caused by the difference between the virtual system in simulation and the real one. However, it is safe to say that the evolved system does outperform the original one. The flaws are very small and have less effects on the performance of the upper level behaviors which are based on "go–position–at–angle" action. The same is experimentally verified in Section 7.5.2 for the "shoot" behaviors.

7.5.2 Evolution at the robot behavioral level

The robot behaviors are constructed from primitive behaviors. At this level, GA is applied to the rule bases. The "shoot" behavior, which is the trickiest and most important one at the robot behavior level, is selected as a test case. The "shoot" behavior belongs to the attacker role and is vital in winning a game. This behavior is implemented by the "get–ball–at–angle" actions, which is a modification of the "go–position–at–angle". To perform the "shoot" behavior, the fuzzy controller of the attacker analyzes the positions of the opponent goalie and the ball, decides the best shooting angle and activates "get–ball–at–angle" actions to fulfill the shooting. The ball position is gauged with the angle λ of the ball with respect to the center of the goal. λ is calculated as follows:

$$\lambda = tan^{-1} \frac{BallY - GoalY}{BallX - GoalX} \quad , \tag{7.2}$$

where $BallY$ and $BallX$ are the coordinates of the ball, and, $GoalY$ and $GoalX$ are the coordinates of the center of goal (Figure 7.16). The opponent goalie's position is determined by its y-coordinate. The output variable of the fuzzy rule base is the y-coordinate of the desired target position to shoot ($TargetY$ in Figure 7.16). Table 7.1(a) is the original rule base for "shoot" behavior [117].

In Table 7.1(a), the inputs to the fuzzy inference engine are classified into 20 states. Verified in the early experimentations and matches, this number of states are suitable and enough to represent the input space. Thus the antecedent part of fuzzy rules are not evolved in this system. The evolutionary optimization is focused on the consequent parts of fuzzy rules.

In the simulation, the attacker robot is fielded against the opponent goalie, whose behaviors are pre-programmed as in [117] and have demonstrated good performance in real world experimentations. The ball is kept at ten different positions. Five of the positions are predefined and fixed throughout the evolution and, the other five positions are randomly chosen and changed in every generation. The fixed shooting positions are benchmark positions to provide a universal standard through all the generations. Without the fixed positions, the fitness of the same

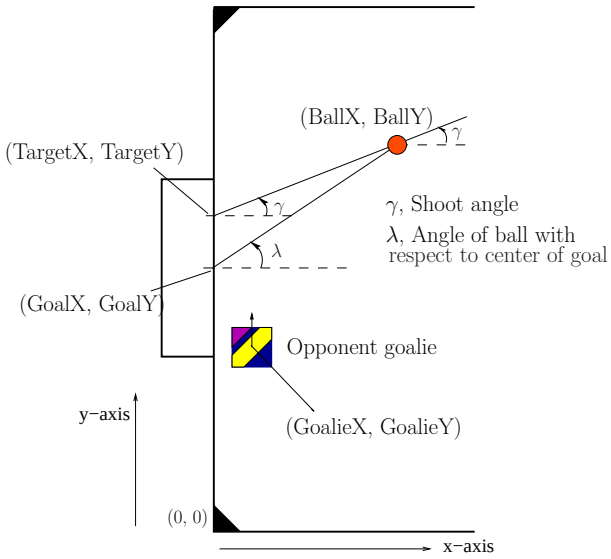


Figure 7.16: The inputs and outputs of shoot behavior

		Y-coordination of opponent goalie				
		FL	CL	C	CR	FR
Angular	FL	R	L	L	CL	CL
Ball	L	CR	CR	L	CL	CL
Position	R	CR	CR	R	CL	CL
	FR	CR	CR	R	R	L

(a)The original rule base

		Y-coordination of opponent goalie				
		FL	CL	C	CR	FR
Angular	FL	CR	L	R	C	CL
Ball	L	R	R	L	L	L
Position	R	R	R	R	L	L
	FR	CR	C	L	R	CL

(b)The evolved rule base

Linguistic values: L(left), R(right), C(center), FL/R(far left/right), CL/R(central left/right)

Table 7.1: Rule bases for shoot ball

chromosome may change too much in different generations and become meaningless. Meanwhile, fixed positions are not enough to represent the whole testing space, and that is why the random positions are necessary.

The goal score "*Goal*" achieved by the robot is an apparent guideline to the performance of "shoot" behavior. However, only the goal score itself cannot reveal the performance difference between chromosomes with the same goal score. Furthermore, the goal score of the same chromosome is a bit unstable due to the randomly selected shooting positions. Thus, another performance index ΔY is introduced as a measure of the quality of the "shoot" behavior. It is defined as the difference (denoted as ΔY) between $TargetY$ and the Y-coordinate of the goalie at the moment when the robot kicks the ball. The bigger ΔY means the higher chance to score a goal. Since the width of the goal is 40cm and the goalie seldom leaves its goal area, the maximum ΔY value is 40. The fitness function F_{shoot} is thus defined as follows:

$$F_{shoot} = Sum(\Delta Y) + 40 \cdot Goal \ , \tag{7.3}$$

where $Sum(\Delta Y)$ is the sum of ΔY obtained by shooting actions from the ten starting positions. When a goal is scored, the maximum ΔY (40) is granted to that run as a bonus.

The objective of GA is to maximize the chance of scoring a goal by evolving the rule base. Due to the symmetry of the rule base (Table 7.1), 10 of the 20 fuzzy rule consequences (the outputs) are encoded into one chromosome. The population size is set to 180. The GA operations (selection, crossover and mutation) are the same as those for the "go–position–at–angle" action, with different parameter setting. The elite size is taken as 10, the crossover probability P_c as 0.6 and the mutation probability P_m as 0.05.

A typical GA evolution for the "shoot" behavior is plotted in Figure 7.17. The convergence is usually reached around 80 generations, after that the best individual dominates the population. Due to the random starting positions and the existence of the goalie robot, the fitness value of the same individual is not consistent. That is the cause of the fluctuations in the average fitness and highest fitness curves

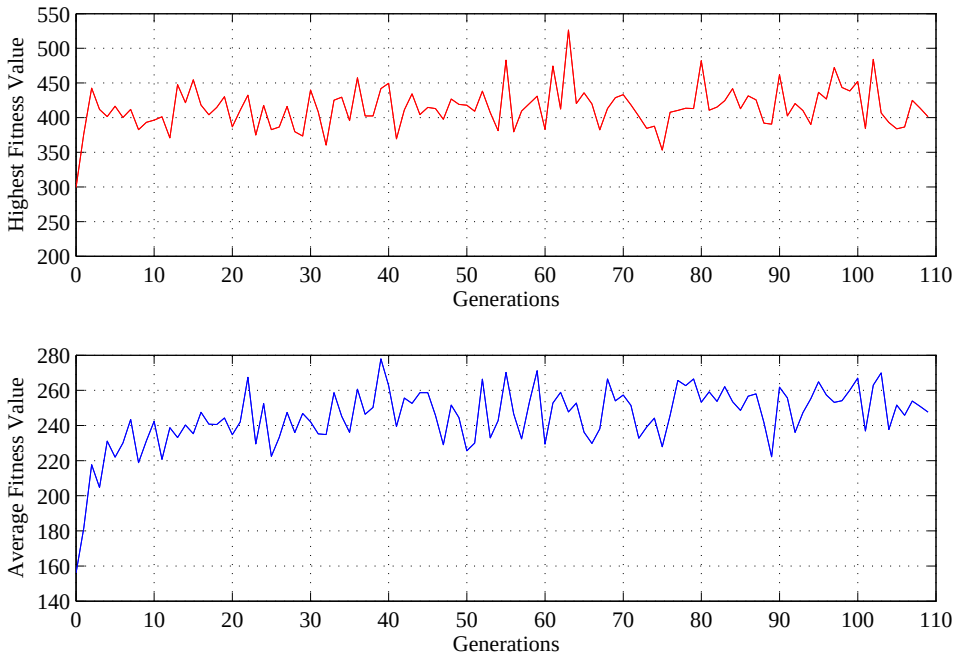
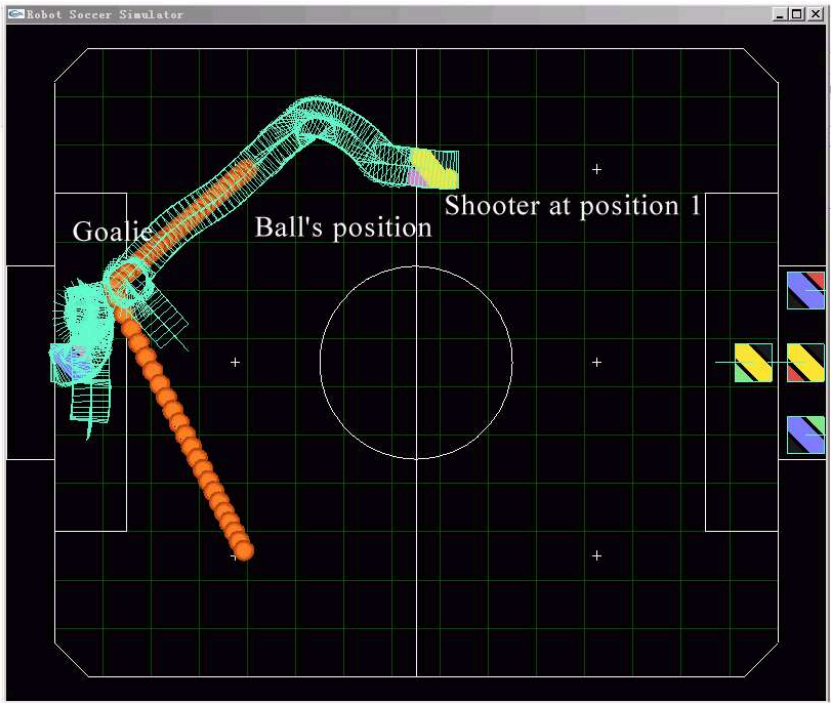


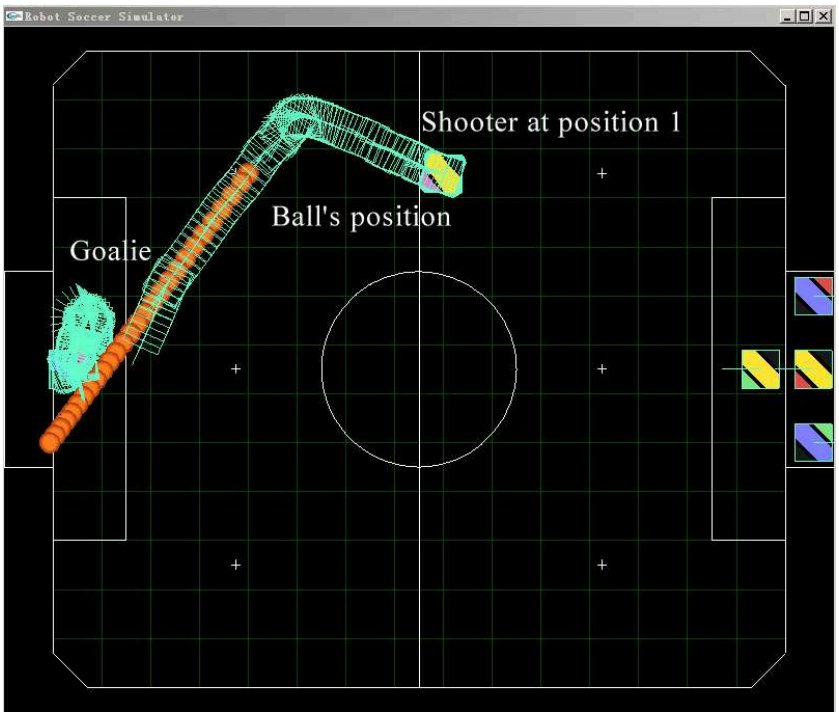
Figure 7.17: The GA process for the shoot behavior

in Figure 7.17. However, the long term improvement of average fitness is still observable. Although the highest fitness value in each generation changes from time to time, the genotypes of best individuals in different generations are in fact the same or quite similar. That means the GA process is stabler than its appearance in Figure 7.17.

The performance of the evolved rule-base (Table 7.1(b)) is compared with the original system in simulation (Fig. 7.18, 7.20, 7.22, 7.24), as well as through real world experimentation (Fig. 7.19, 7.21, 7.23, 7.25). Four shooting positions are selected from the areas in which the "shoot" behavior is most commonly performed. The scoring percentages of the original and evolved systems are summarized in Table 7.2 with 30 shots for each shooting position. It is worthy of mention that attacking is always much harder than defending in a robot-soccer competition. Compared to the width of the goal (40cm), the goalie robot is quite large an obstacle in size (7.5cm). Furthermore, this "obstacle" is as agile as any other robot player and its task is to block the ball. It is quite normal that the goalie may block most of the shots which is observed with the original rule-base. The attacker failed

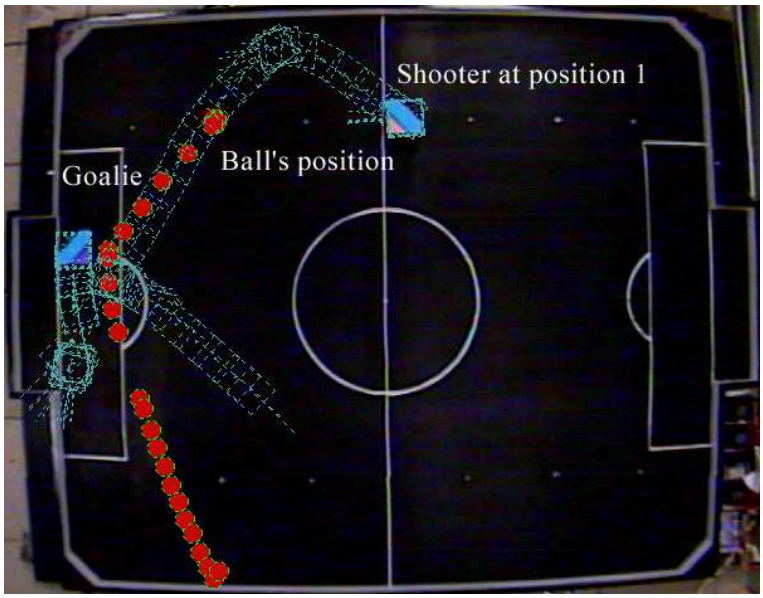


(a) Trajectory of original system

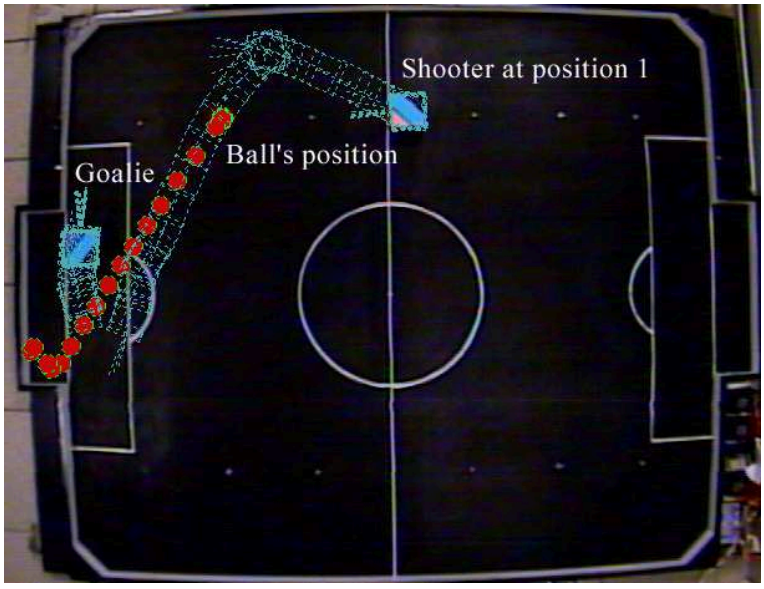


(b) Trajectory of evolved system

Figure 7.18: Simulation performance of "shoot" behavior (position No. 1)



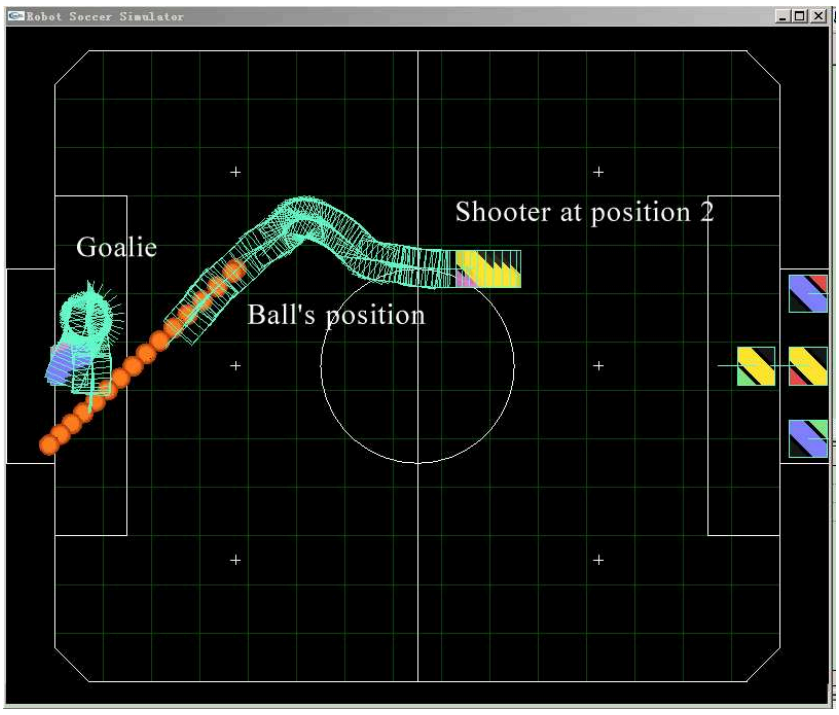
(a) Trajectory of original system



(b) Trajectory of evolved system

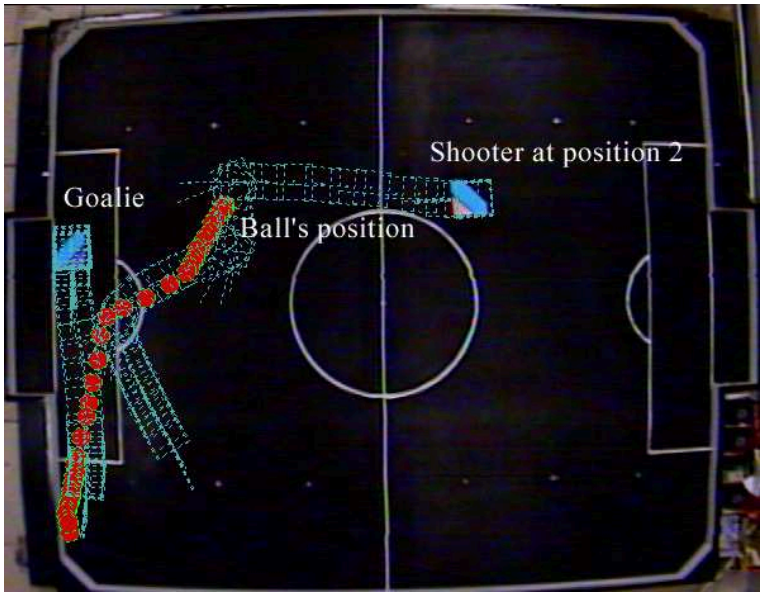
Figure 7.19: Real world performance of "shoot" behavior (position No. 1)

(a) Trajectory of original system

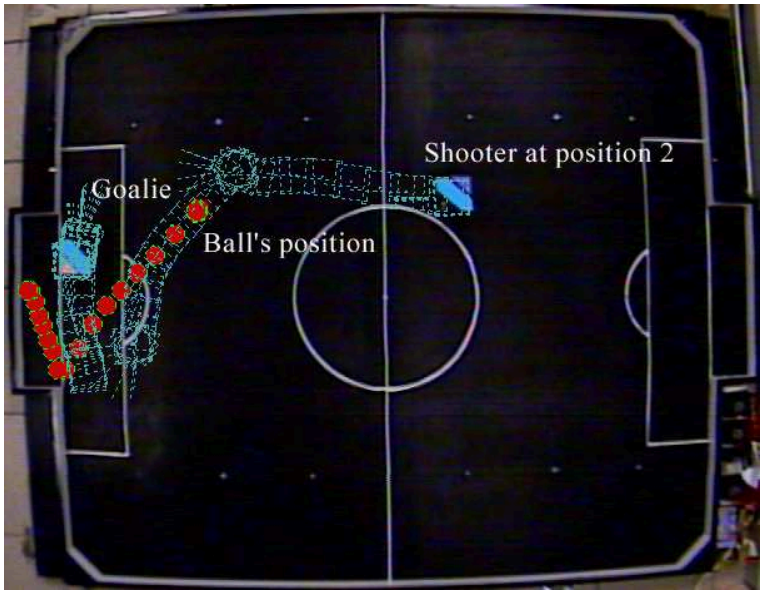


(b) Trajectory of evolved system

Figure 7.20: Simulation performance of "shoot" behavior (position No. 2)

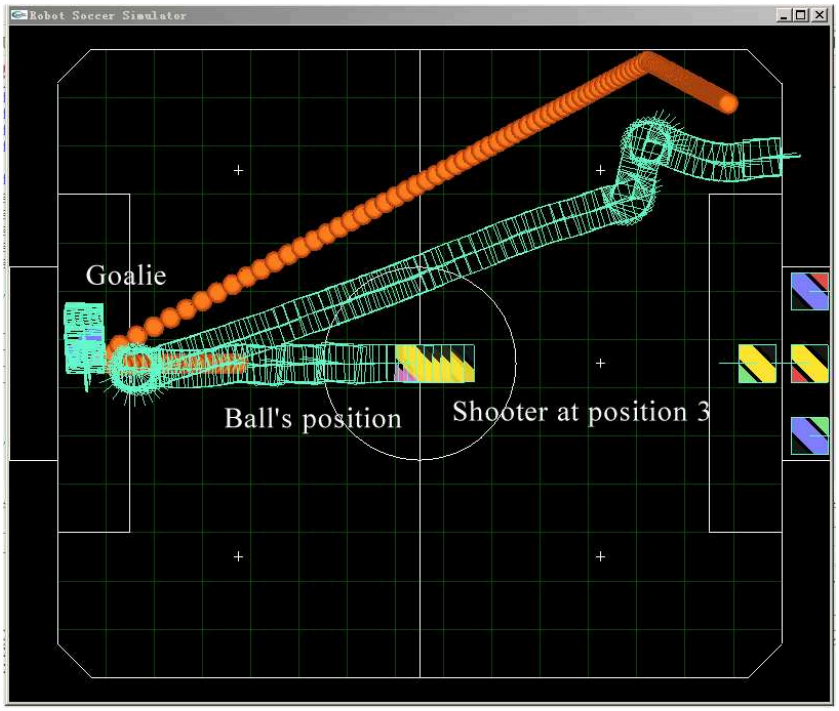


(a) Trajectory of original system

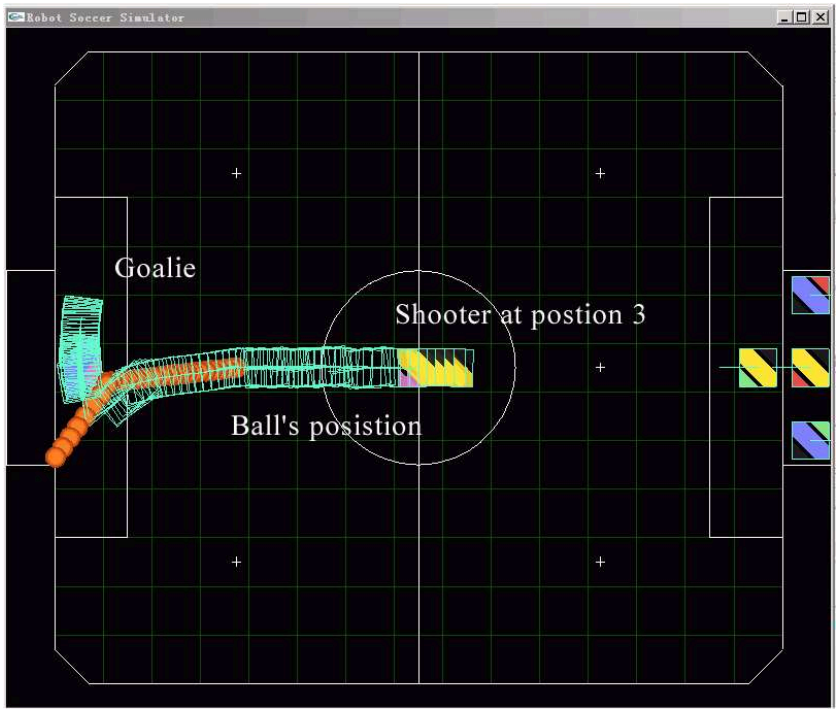


(b) Trajectory of evolved system

Figure 7.21: Real world performance of "shoot" behavior (position No. 2)

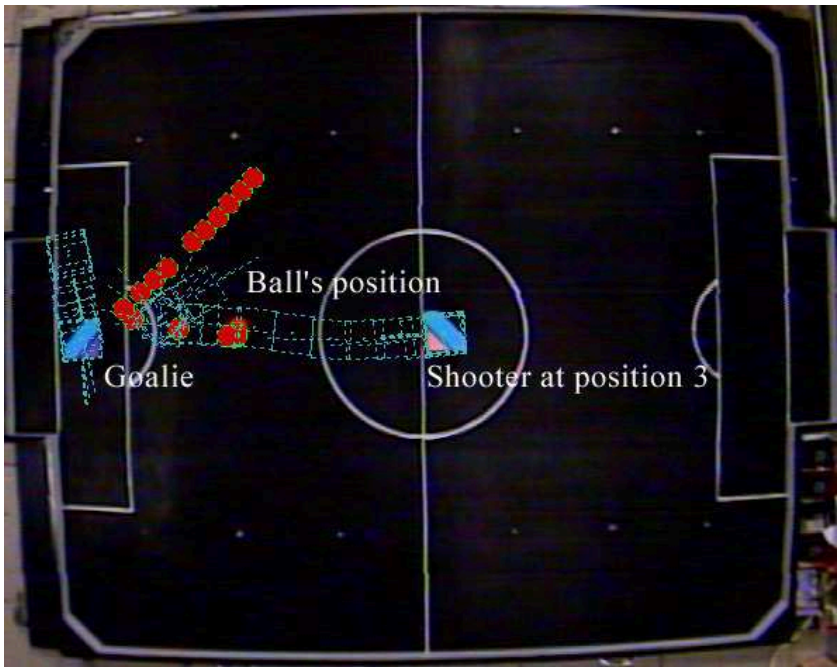


(a) Trajectory of original system

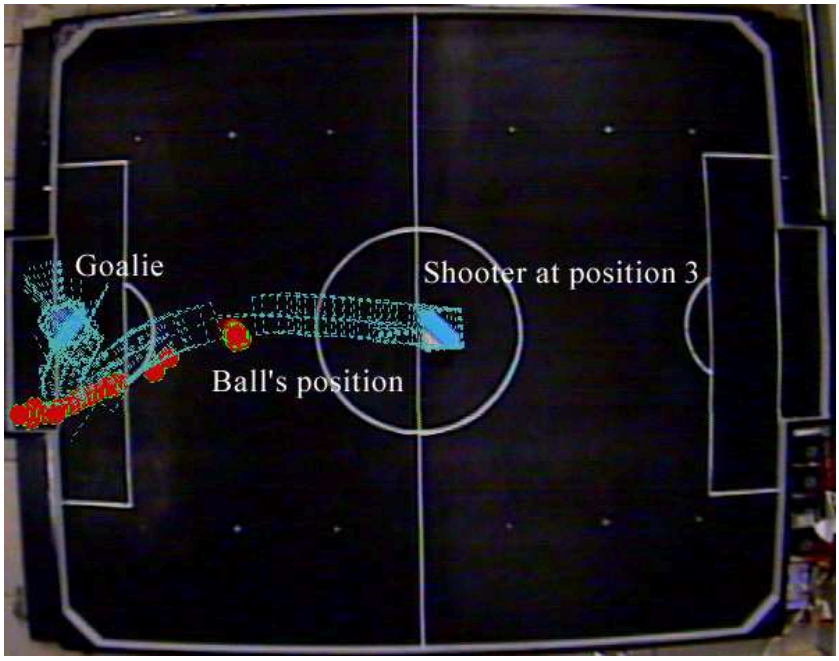


(b) Trajectory of evolved system

Figure 7.22: Simulation performance of "shoot" behavior (position No. 3)

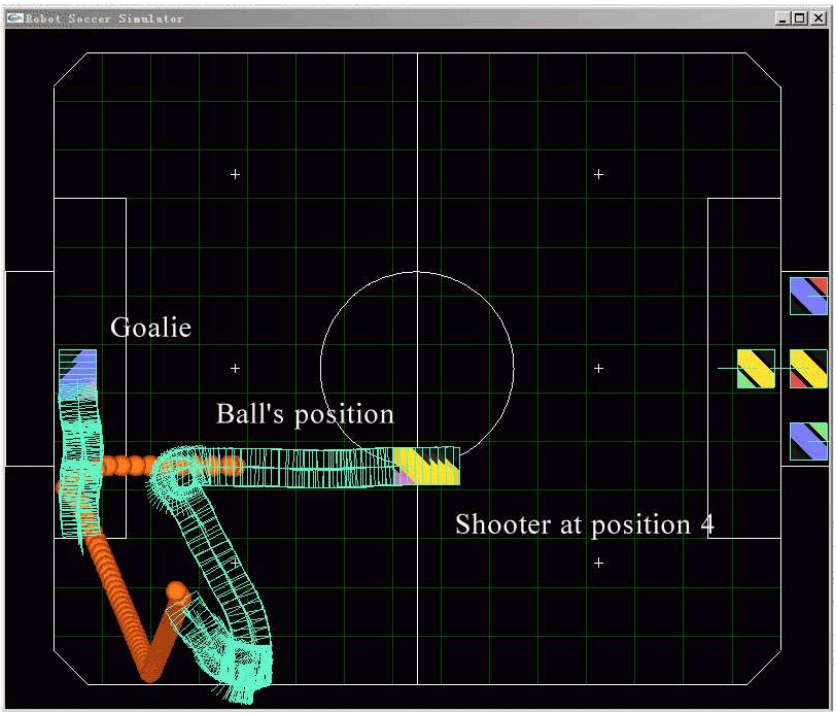


(a) Trajectory of original system

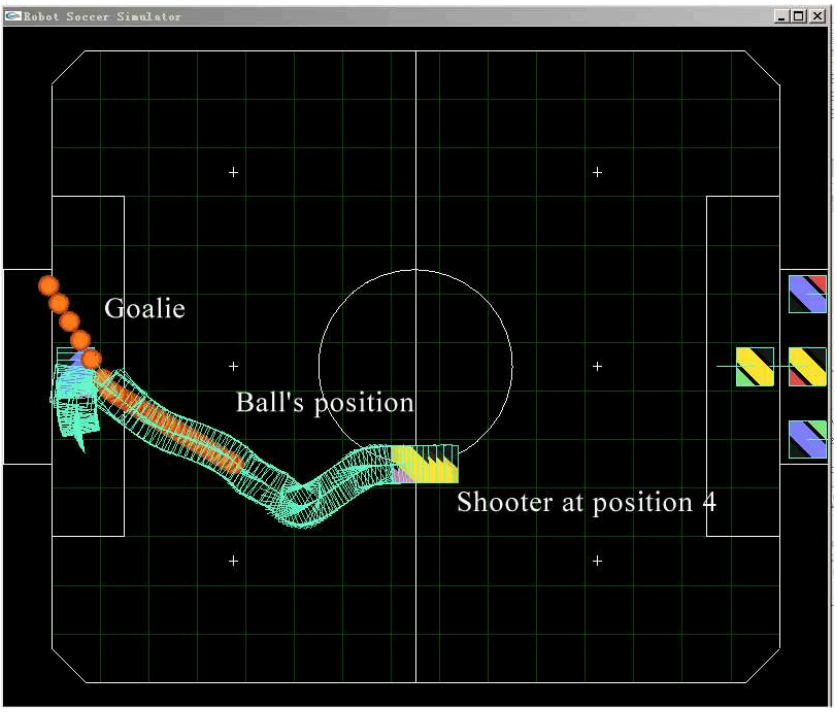


(b) Trajectory of evolved system

Figure 7.23: Real world performance of "shoot" behavior (position No. 3)



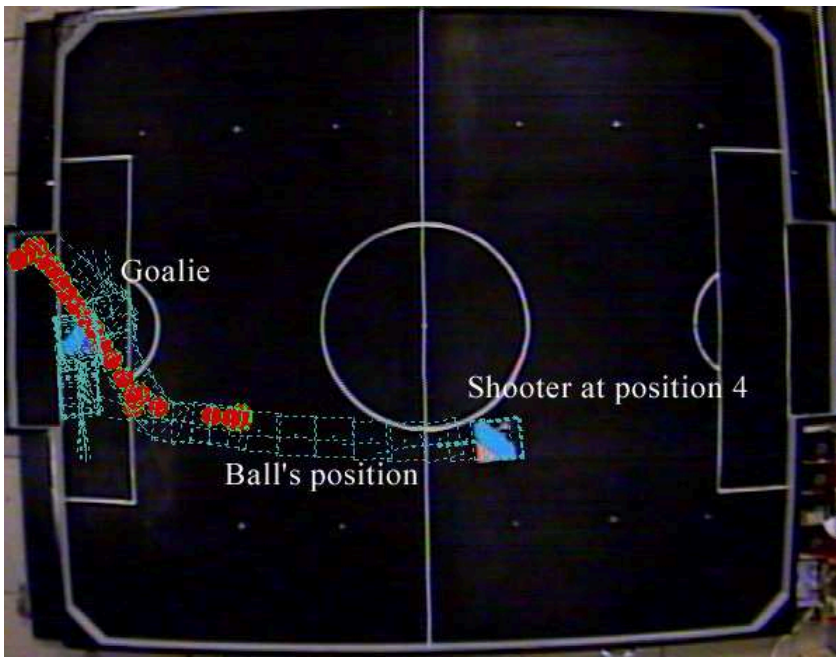
(a) Trajectory of original system



(b) Trajectory of evolved system

Figure 7.24: Simulation performance of "shoot" behavior (position No. 4)

(a) Trajectory of original system



(b) Trajectory of evolved system

Figure 7.25: Real world performance of "shoot" behavior (position No. 4)

to score any goals at most of the times and even never succeeded from certain positions (i.e. position No. 2 and 4) (Table 7.2). On the contrary, the evolved rule-base highly improved the scoring percentage and the attacker successfully scored goals from all of the four positions (Table 7.2).

Shooting Positions	Original System		Evolved System	
	Simulation	Real World	Simulation	Real World
Position 1	10.0%	3.3%	23.3%	13.3%
Position 2	3.3%	0%	16.7%	10.0%
Position 3	13.3%	13.3%	16.7%	20.0%
Position 4	0%	0%	23.3%	16.7%

Note: scoring percentage = (goal scored / number of shots) × 100%

Table 7.2: Comparison of scoring percentage

From both the simulation (Figures 7.18, 7.20, 7.22, 7.24) and real world results (Figures 7.19, 7.21, 7.23, 7.25), it is observed that the robot in the evolved system always tries to shoot the ball into the far end of the goal with respect to the reactions of the goalie robot. One good instance in real world experimentation is the scenario at position 4 (Figure 7.25(b)). At the beginning, the ball is on the left side of the goal and the goalie is at the center. When the goalie moves to the left side where the ball is nearby, the robot makes a turn and shoots the ball to the right end of the goal. Although the goalie quickly changed its direction, it barely missed the ball. That is a good indication of the skill developed from GA evolution. It should be mentioned that the accuracy of the "shoot" behavior is determined by the primitive actions from which it is constructed ("get–ball–at–angle", "go–position–at–angle", etc.). The performance of the evolved "shoot" behavior implies that the optimized "go–position–at–angle" is quite satisfactory.

7.5.3 Evolution at the group behavioral level

After some of the important individual robot behaviors are optimized, the evolution is carried out at the higher layer which is more strategy related. At the role

assignment level, the robot soccer system needs to evaluate the current competition situation, carry out the team strategy by the intelligent selection and assignment of four roles (attacker, midfielder, defender and goalie) to three robots. Since the presence of the goalie is a must, the assignment is in fact to select roles from attacker, midfielder and defender for the other two robots.

Role assignment mechanism revised

The input space of the fuzzy rule base at this level is the whole play-field. With the need of handling complicated environmental information, the antecedent parts of fuzzy rules become very important at this level. Both the antecedent and consequent parts of fuzzy rules need to be evolved.

In the original system, the role assignment adopts a quite simple mechanism. At first, the goalie role is fixed with one robot. The attacker role is always present because it is the only role equipped with offensive behaviors. Between the two robots other than the goalie, the one which is currently closer to the ball is chosen as the attacker. Due to the possible conflict, the same role is not assigned to two robots at the same time. As a result, the fuzzy engine needs only to chose a role for the third robot between midfielder and defender (Table 7.3(a)). The inputs to fuzzy engine are the distance of the closest opponent robot to home goal and the distance of the ball to home goal. The outputs are the role of the midfielder or defender. This mechanism is too simple to handle the complicated and dynamic system activities. Its inability often causes inappropriate group behaviors. For instance, it is often found in the competition [115, 126] that: when a robot is stuck with opponent robots, its role cannot be taken up by another robot which is in idle status then.

In this work, no limits are set in the role selection of the two robots other than the goalie robot. The two robots are allowed to perform the same role at the same time. Both the antecedent and consequent components of fuzzy rules are evolved. There are three linguistic values (near, medium and far) for the two inputs: distance of the closest opponent robot to home goal and the distance of the ball to home

goal. The output is a pair of roles chosen from three roles (attacker, midfielder and defender). The six possible role combinations are the possible rule consequents. The antecedents are randomly combined with the consequents to generate a fuzzy rule. Every individual of GA represents a rule base which contains a set of fuzzy rules. The size of rule base is defined as 18, twice the number of the possible antecedents. It is natural to find rules with the same antecedents in one rule set. Under this situation, only the first rule encountered in the string is considered. Meanwhile, it is also normal that sometimes not all the nine antecedents are present in one individual rule base. The missing of antecedents implies that the relative input states are omitted in the fuzzy controller. The fuzzy rule base can still work with such kind of omission although the output may be different. The effects of the missing antecedents on the performance are left to the evolutionary process to handle. If an antecedent is crucial to the fuzzy controller's performance, any individual rule base without it, receives a low fitness value and cannot survive the evolution. As a result, only the unimportant or unnecessary antecedents will be in the risk of omission and that kind of omission is indeed making the rule base more efficient.

Evolution process

The performance of the new rule base is evaluated by competing with a team using the original rule base. Two teams of robots are pitched for a match for a certain period (a certain number of steps). The match is considered over when the total goals scored reaches the limit (set as 3 here) or the time (set as 1200 steps) is up. Besides the scores, the time duration for which the ball is each which side of the field is also recorded. The fitness function F_{role} is defined as follows:

$$F_{role} = u + v \cdot (Goal - Lost) + w \cdot \frac{T_{opp}}{T_{total}} \ , \tag{7.4}$$

where u, v and w are integer constants, $Goal$ and $Lost$ are scores of the home and opponent teams, T_{opp} is the time for which the ball is in the opponent half of the field, and T_{total} is the total match time. The ratio of T_{opp} to T_{total} is an indication of which team takes the upper hand in a game and the associated weight of this

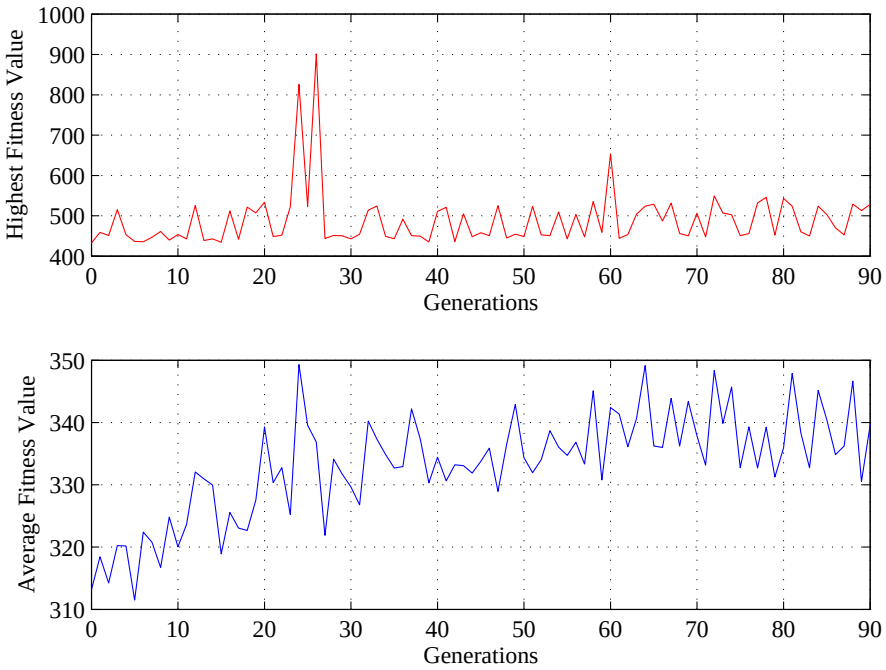


Figure 7.26: The GA process for role selection and assignment

ratio (w) is set to 100. The weight attached to the score difference (v) is set to 100. Since the team may lose up to 3 goals, the second part of F_{role} can be negative. u is utilized to keep F_{role} non-negative and is set to 300.

The population size of GA is fixed at 100 while the elite size is set to 8. The crossover probability P_c and the mutation probability P_m are set to 0.65 and 0.05 respectively. In the evolution process (Figure 7.26), it is noticed that the fluctuations in the fitness value curves are a bit large. This is not strange considering the rather random nature of a soccer match. The dynamic situations in the competition made the fitness of the same individual inconsistent. However, despite the fluctuations, the observable improvement in the average fitness shows the effectiveness of GA. Although the best individuals keep changing at each generation, they are found similar to one another at later stages of evolution. It is also noticed that not all the best individuals contain all the nine possible antecedents, which means that the antecedents have different degrees of importance. Some of the antecedents can be omitted and replaced by others with similar inputs. Such a feature is quite helpful in situations where the increase in the number of input variables results

		Distance of closest opponent robot to home goal		
		Near	Medium	Far
Ball distance to home goal	Near	Defender	Defender	Midfielder
	Medium	Defender	Midfielder	Midfielder
	Far	Defender	Midfielder	Midfielder

(a) The original rule base

		Distance of closest opponent robot to home goal		
		Near	Medium	Far
Ball distance to home goal	Near	AA	MD	DD
	Medium	AM	AD	AA
	Far	AD	AA	AA

(b) The evolved rule base

Linguistic values: AA(two attackers), DD(two defenders), MM(two midfielders), AM(attacker & midfielder), AD(attacker & defender), MD(midfielder & defender)

Table 7.3: Rule bases for role assignment

in an exponential increase in the number of the rules. Under such circumstances, finding the most important antecedents can help to decrease the size of the rule base and to increase computational efficiency. The evolved rule base is displayed in Table 7.3(b).

Performance Data	Real World		Simulation	
	Original	Evolved	Original	Evolved
Scores	1	1	1.2	1.4
Number of shots	14	22	15.4	18.2
(T_{opp}/T_{Total})	—	—	36.3%	63.7%

Note: In simulation, the average values from 5 matches are used

Table 7.4: Comparison of match performances

The performance of the evolved rule-base is evaluated in simulation and real world against the original rule-base. For a real world match of 20 minutes and several simulation trials, the score and number of shots are recorded. Additionally, the ratio of T_{opp} to T_{total} in simulation is also noted. The recorded data is compared in Table 7.4. Though the score difference is small, the scores do not necessarily speak

of or reflect the competition strategy in place. The differences in number of shots and ratio of T_{opp} to T_{total} (Table 7.4) indicate that the evolved team has performed better and gained dominance in the matches compared with the original system. The evolved team is observed to be aggressive in attacking and in defending. It is noteworthy that the original team has won several prizes at the international level robot-soccer competitions [115, 126].

7.6 Conclusion and Discussion

The decomposition of a complicated system into various simple behaviors is proven to be an effective method for realization of complex control systems. The GA's development and optimization at different levels of fuzzy behavioral architecture resulted in more agile and intelligent behaviors. The GA is applied to different aspects of the fuzzy controller at different levels of the behavioral architecture. Based on the original system which is already well designed, improvements ware achieved at different levels of the architecture. At the lowest level, the accuracy of primitive behaviors are obviously enhanced. At the higher levels, more skillful behaviors (like shoot to the far-end corner of the goalie) are developed, as well as a more effective role assignment mechanism which embodies the group behavior. All of these are verified through the simulation and real-world experimentations.

In general, the optimized behaviors enable the robots to carry out the tasks more accurately and the evolved group behavior results in a better game strategy. Meanwhile, the evolutionary algorithm provides an efficient way to develop and optimize a complex multi-robotic system. Utilizing the simulator, the optimization process is much faster than the manual "trial and error" method, especially when there is not enough knowledge of the system to guide the manual tuning.

There is still improvement space for the team strategy part of the architecture. For example, a more feasible and comprehensive performance index for the evaluation of the strategy is necessary for the evolutionary methods. The mechanism used here can be extended to other multi-robotic systems as well.

Chapter 8

DNA Coded GA for Fuzzy Robot-Role Assignment

In the evolution of team strategy for fuzzy behavior based robot soccer system, all the antecedents and consequents of fuzzy rules are evolved through genetic algorithm. The coding method determines the meaning of a character in the individual chromosome by the characters surrounding it. In another words, the meaning of each character is context dependant, not position dependant. Despite the fact that position dependant coding is most commonly used in GA, a context dependant coding formation is in fact much closer to the natural DNA chromosome. The coding strings can have variable length and are still compatible to the normal genetic operations. This chapter project the DNA coding in a more general scheme. The specific features of DNA coding methods and their influence on the genetic algorithms are analyzed through the robot soccer role assignment problem.

8.1 Introduction

Humans have always looked towards nature for solutions to everyday problems. Evolutionary computation is a collection of computation algorithms derived from observing nature, in particular evolution of population through natural selection [127, 128, 129]. Nature, through constraints in environment, has been able to select the offspring from the individuals with the most suitable attributes. In the same

way, an optimum solution to computation problems can be selected by applying constraints to a population over many generations.

While studies and theories of Charles Darwin and Gregor Mendel had been the basis in the science of selection, a great breakthrough in genetics came in the year 1953 when James Watson and Francis Crick unveiled the secret of life, the Deoxyribonucleic Acid (DNA) [130]. Henceforth, researchers have begun to understand how genetic information is coded and passed on from one generation to the next.

Motivated by the gene expression which involves translation of nucleotide sequences of DNA into amino acid sequences, the DNA like coding method has been proposed for evolutionary computing [131, 132, 133, 134]. DNA coded strings are used to represent the fuzzy "if-then" rule bases [131, 132, 134, 135, 136] or the production rules of the L-system [133] as the individuals of genetic algorithm. The DNA coding method allows flexible representation, overlapped and redundant coding, variable string length and no restrictions on the crossover point. The simulation results suggested that the redundancy and overlapping in DNA coding worked well for fuzzy rule discovery [131]. However, the reported works lack in explicit explanation.

The main objective of this chapter is to project the DNA coding in a more general scheme, which is characterized by the specific features: the context dependency, intron parts, redundancy and variable string length in evolutionary algorithms. It also aims to analyze the influence of these features and explain their influence on the performance, via the simulation study in the context of role assignment in a robot soccer system.

8.2 Coding Methods for Genetic Algorithm

Genetic algorithm is the optimization algorithm based on Darwinism and Mendelism. In genetic algorithm, a set of parameters is selected to define a solution to the problem. Depending on the problem, various coding methods have been used in the implementation on genetic algorithm. Different problems call for different representations as well as search methods for maximum efficiency. The most popular coding methods for genetic algorithm include binary coding, real value coding and permutation coding.

Binary coding is simple and most widely used. It is also easy to implement various types of genetic operators when binary coding is used. For example, mutation can be done by randomly inverting bits in a chromosome. Binary coding is well suited for problems where solutions are pseudo-Boolean as in knapsack problem and other combinatorial problems. Integer or real-valued variables can also be represented by binary strings of specific length depending on the required accuracy. An alternative to binary coding is the gray coding, which is used by some researchers to eliminate the Hamming cliff problem associated with binary coding [137]. In gray coding, any two consecutive strings differ only by one bit, whereas in binary coding this is not the case. However, in a binary coded string or even a gray coded string, a bit change in any arbitrary position may cause a large change in the represented integer or real number. Furthermore, both binary and gray codings may cause representation space to be much more complicated than the searching space.

The real value coding method was initially used in evolution strategies and evolutionary programming, which is characterized by the direct operation on the real-valued solution vector. Since there is no empirical evidence indicating that binary coding results in greater efficiency in solving real-valued problems, there has been a trend away from binary coding in GA research [138, 139, 140]. For real-valued numerical optimization problems, the real value coding outperforms binary coding method because it is more convenient, consistent and concise in representation [141].

Permutation based coding method is usually used to represent logical solutions for scheduling problems and classic combination problems such as the traveling salesman problem. Genetic algorithms using permutation coding method are referred as ordering GAs [142]. One obvious attribute of permutation coding is that genetic operators such as simple crossover and mutation may fail to generate valid offsprings. Specialized recombination operators for GA have been proposed, including order crossover 1 [143, 144], order crossover 2 [145], position crossover [145], partially map crossover (PMX) [144] and maximal preservative crossover (MPX) [146].

There are other coding methods, such as mixed-integer coding [147], intron coding [148, 149] and parse tree coding [150]. These coding methods are specialized to cater for different kind of problems. One class of interesting coding method which is discussed in depth in the this chapter is the DNA like coding method, which is motivated by the transcription of DNA to mRNA and the translation of mRNA to proteins [131, 132, 133, 134].

8.3 DNA Like Coding Method

8.3.1 Protein, DNA and messenger RNA

Before discussing the DNA like coding method for GA, it is worthwhile to introduce the natural DNA coding mechanism, which are very important in every life form. It is well known that proteins are the fundamental agents of life; every human being contains something like ten thousand different proteins. Their properties and interactions determine the way human beings are. The information that defines the primary structure* of every protein is encoded in the DNA (deoxyribonucleic acid). In fact, the protein (in its primary structure) is a linear sequence of a combination of twenty different amino acids which are decided by DNA.

*The first product of the protein synthesis. The final structure of the protein (that is the one determines its function) is the result of the interactions between the primary structure and its environment. Understanding this process in detail is an open problem faced by the biologists.

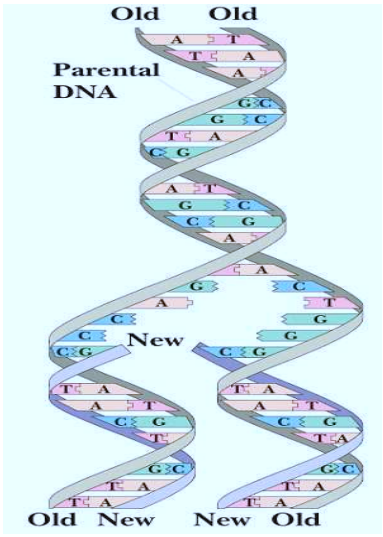


Figure 8.1: The chemical structure of DNA

DNA is a double stranded sequence of four nucleotides which are adenine (A), guanine (G), cytosine (C) and thymine (T). A portion of DNA sequence (a gene) encodes the information that determines the sequence of amino acids of the protein. It is for this reason that scientists use the expression *genetic code*. In a word, DNA contains the genetic information that defines the proteins, the *engines* of life.

The four nucleotides that compose a strand of DNA are often called bases. The chemical structure of DNA, the famous double helix (Figure 8.1), was discovered by James Watson and Francis Crick in 1953. It consists of a particular bond of two linear sequences of bases. This bond follows a property of complementarity: adenine bonds with thymine and viceversa, and cytosine bonds with guanine and viceversa. This is known as *Watson-Crick* complementarity and it is also denoted as follows:

$$\overline{A} = T, \quad \overline{T} = A, \quad \overline{C} = G, \quad \overline{G} = C.$$

One of the strands which constitute a DNA molecule holds the information that codes various genes. This strand is often known as the *template strand* or *antisense strand*. The complementary strand is the *coding strand* or *sense strand*. In the synthesis of proteins, the messenger Ribonucleic acid (mRNA) is constructed at first. As being constructed from the *template strand*, the mRNA strand has the same information as the *coding strand*. The genetic code for the mRNA is identical

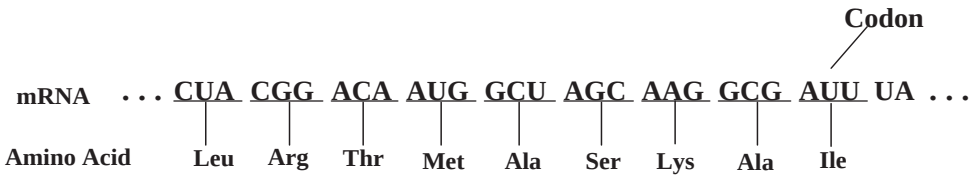


Figure 8.2: Codons in mRNA and corresponding Amino Acids

to the *coding strand* but for the fact that RNA contains the U (uracil) base rather than the T (thymine) base. The RNA contains many unused parts after the first synthesization. Splicing is performed to remove the unused parts to create the final mRNA.

In the mRNA, three successive bases called codons are allocated sequentially. The codons in mRNA are shown in Figure 8.2. The Leu, Arg, Thr and etc, are abbreviations for amino acids (Table 8.1). Based on the three bases of codon, the corresponding amino acid can be identified. The "AUG" along the left-top-right direction in the Table 8.1 corresponds to "Met". The special *Terminator* codon (Ter) is a stop codon. The transcription, which is the process of synthesizing the RNA based on the DNA strand, is terminated when the "Ter" is present.

There are twenty types of amino acids and *terminator* codons, represented by totally sixty-four three-letter codons (Table 8.1). That results in a certain extent of redundancy. It is important to note that this redundancy helps to relax the accuracy requirement or increase the flexibility. For instance, a point mutation on the last letter is not likely to produce a malfunctioning protein. Similarly, this property can be exploited in DNA coding method.

Another fact worthy of mention is the existence of many non-coding sequences in the RNA after the first synthesization. These non-coding sequences are usually referred as introns while the coding sequences are named as exon. All the introns are precisely spliced to produce the final mRNA (Figure 8.3). The reason for the existence of introns is still being debated. There is a strong belief that introns play a regulatory role in the cell. It is possible that introns of DNA contain sequences that control gene activity in someway or other in the splicing process of the mRNA

<table>
<tr><td></td><td>U</td><td>C</td><td>A</td><td>G</td><td></td></tr>
<tr><td rowspan="4">U</td><td rowspan="2">Phenylalaine
(Phe)</td><td rowspan="4">Serine
(Ser)</td><td>Tyrosine
(Tyr)</td><td>Cysteine
(Cys)</td><td>U</td></tr>
<tr><td></td><td></td><td>C</td></tr>
<tr><td rowspan="2">Leucine
(Leu)</td><td>Terminator
(Ter)</td><td>Ter</td><td>A</td></tr>
<tr><td></td><td>Tryptophan(Trp)</td><td>G</td></tr>
<tr><td rowspan="4">C</td><td rowspan="4">Leu</td><td rowspan="4">Proline
(Pro)</td><td>Histidine
(His)</td><td rowspan="4">Arginine
(Arg)</td><td>U</td></tr>
<tr><td></td><td></td><td>C</td></tr>
<tr><td rowspan="2">Glutamine
(Gln)</td><td></td><td>A</td></tr>
<tr><td></td><td>G</td></tr>
<tr><td rowspan="4">A</td><td rowspan="2">Isoleucine
(Ile)</td><td rowspan="4">Threonine
(Thr)</td><td>Asparagine
(Asn)</td><td rowspan="2">Ser</td><td>U</td></tr>
<tr><td></td><td></td><td>C</td></tr>
<tr><td>Lysine
(Lys)</td><td rowspan="2">Arg</td><td>A</td></tr>
<tr><td>Methionine(Met)</td><td></td><td>G</td></tr>
<tr><td rowspan="4">G</td><td rowspan="4">Valine
(Val)</td><td rowspan="4">Alanine
(Ala)</td><td>Aspartic acid
(Asp)</td><td rowspan="4">Glycine
(Gly)</td><td>U</td></tr>
<tr><td></td><td></td><td>C</td></tr>
<tr><td>Glutamic acid
(Glu)</td><td></td><td>A</td></tr>
<tr><td></td><td></td><td>G</td></tr>
</table>

*U(uracil) replaces T(thymines) in RNA

Table 8.1: The genetic code of amino acids

[151, 152, 153]. The different ways of splicing of introns increase the variation of the gene. Coding methods utilizing introns have already been proposed [148, 154, 149]. In this chapter, the intron's effects on the coding for GA is also explored.

8.3.2 The basics of encoding

Motivated by the gene expression which involves translation of nucleotide sequences of DNA into amino acid sequences, the DNA coding method has been proposed to encode fuzzy if-then rules and neural networks [131, 133].

To illustrate the basic idea of DNA coding for GA, the fuzzy system is taken as example. DNA chromosome is assigned to each fuzzy inference system. The parameters and variables of fuzzy system values are mapped to the protein codons. The strand of DNA is manipulated by genetic operators and then decoded. Unlike

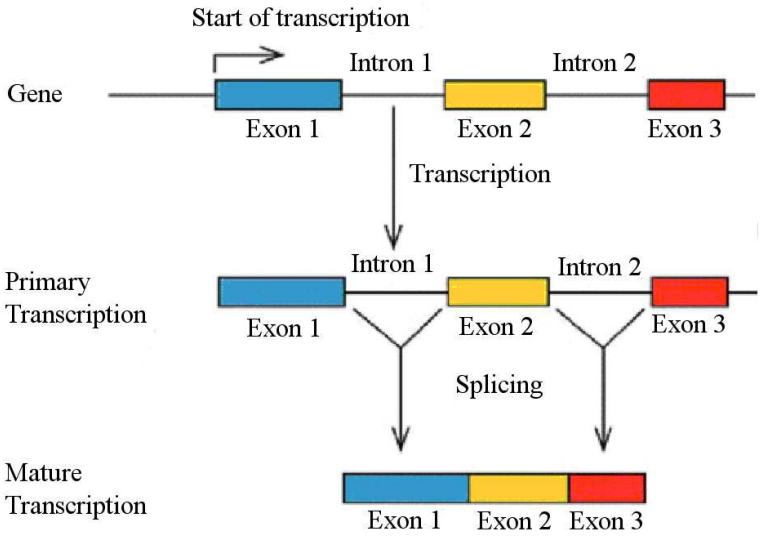


Figure 8.3: The exon and intron

Figure 8.4: Reading and translation of genes

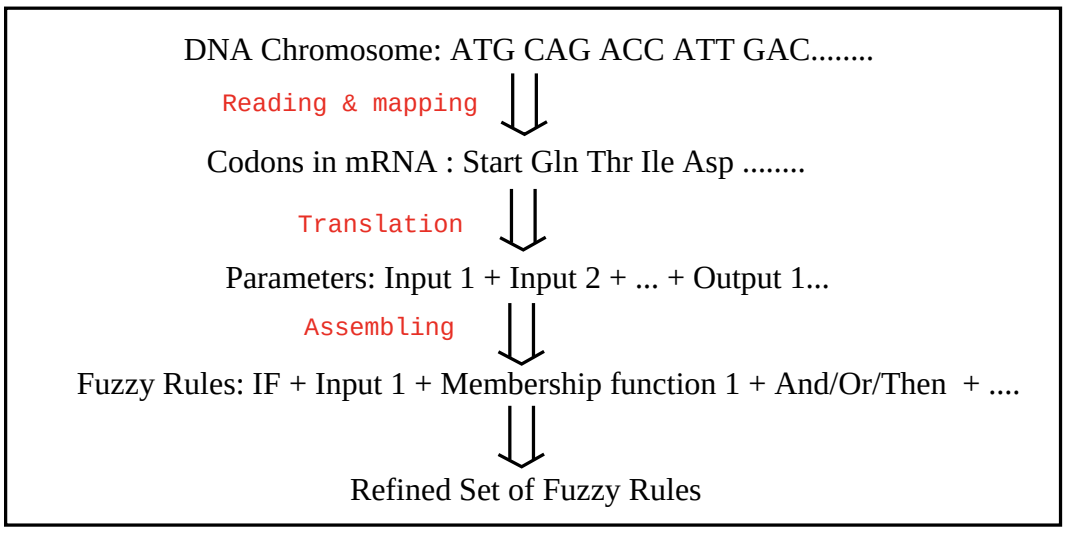


Figure 8.5: Translation from DNA to fuzzy rules

<table>
<tr><td rowspan="2"></td><td rowspan="2">Input</td><td colspan="2">Membership Function Setting</td><td rowspan="2">And/Or/Then</td><td rowspan="2">Output</td><td rowspan="2">Weight</td></tr>
<tr><td>Central Position</td><td>Discourse Spread</td></tr>
<tr><td>Phe</td><td rowspan="4">Input 1</td><td rowspan="2">Position 1</td><td rowspan="4">Width 1</td><td rowspan="6">AND</td><td rowspan="6">Output 1</td><td>0.1</td></tr>
<tr><td>Leu</td><td>0.2</td></tr>
<tr><td>Ile</td><td rowspan="2">Position 2</td><td rowspan="2">0.3</td></tr>
<tr><td>Val</td></tr>
<tr><td>Ser</td><td rowspan="4">Input 2</td><td rowspan="2">0.4</td></tr>
<tr><td>Pro</td></tr>
<tr><td>Thr</td><td rowspan="2">Position 3</td><td rowspan="5">Width 2</td><td rowspan="4">Output 2</td><td rowspan="2">0.5</td></tr>
<tr><td>Tyr</td></tr>
<tr><td>His</td><td rowspan="3">OR</td><td>0.6</td></tr>
<tr><td>Gln</td><td rowspan="4">Input 3</td><td rowspan="3">Position 4</td><td rowspan="3">0.7</td></tr>
<tr><td>Asn</td></tr>
<tr><td>Lsy</td></tr>
<tr><td>Asp</td><td rowspan="2">Position 5</td><td rowspan="5">Width 3</td><td rowspan="5">THEN</td><td rowspan="5">Output 3</td></tr>
<tr><td>Glu</td><td rowspan="4">Input 4</td><td rowspan="2">0.8</td></tr>
<tr><td>Cys</td></tr>
<tr><td>Trp</td><td rowspan="3">Position 6</td><td rowspan="2">0.9</td></tr>
<tr><td>Arg</td></tr>
<tr><td>Gly</td><td>1.0</td></tr>
</table>

Table 8.2: A possible sample translation table

biological DNA, it is unnecessary to transcript the DNA chromosome into mRNA. Decoding can be done by reading the DNA string starting from the beginning or a predefined start point, such as an "ATG" codon (Figure 8.4). Every triplet codon is translated into a corresponding parameter or logic operation according to the predefined coding table (Table 8.2). The whole procedure is depicted in Figure 8.5. It should be noticed that in Table 8.2, every codon has different possible meanings. The exact meaning of a codon depends on the meaning of the codon just before it. For instance, the codon "His" represents the central position of the membership function if it follows a codon representing the input and the codon after it is translated as the discourse spread of the membership functions. Moreover, if "His" follows a codon representing the fuzzy relation "AND", it is decoded as another input. If the codon before it means the "THEN", clause in a typical fuzzy "IF ... THEN ..." statement, it will be an output. In general, the meaning of a codon is context depended.

In this work, a more general DNA coding mechanism is explored, which is

characterized by the features of context dependency, intron parts, redundancy and variable string length. Under such a scheme, total flexibility in codons' definitions and translation rules are allowed. Even the codons themselves can be redefined as 2-letter couplets or any n-letter combinations from the alphabet {A, C, G, T}, other than the natural amino acid triplets. Some codons may share the same meaning, resulting in redundancies as in natural chromosomes. Some others may stand for nothing in particular, just like introns in RNA. The encoded individual string can be of variable length. Since there is no restriction on the formation of the individual string, DNA coded strings are compatible to simple genetic algorithm (SGA) operators. The features of the DNA like coding are further analyzed with a case study.

8.4 DNA Coded GA for Robot-Role Assignment

To test out the DNA coding method, the fuzzy behavior for role assignment in robot soccer system is evolved by DNA coded genetic algorithm.

In Section 7.5.3, the evolution of the group behavior of the behavior based robot soccer system is discussed. The major function of the group behavior is to assign roles to soccer robots. In the original system (Chapter 5), the goalie role is fixed onto robot. For the rest of the two robots, the existence of an attacker is always necessary and it is assigned to that robot closer to the ball. Since any role is restricted to only one robot, the fuzzy system needs only to select a role between defender and midfielder for the last robot, based on the positions of the ball and the opponents in the playground. In Section 7.5.3, the limit of "one role for one robot" is eased. More role combinations are allowed and the fuzzy system is evolved by GA to make the choice. Since only the distance of the closest opponent robot to home goal and the distance of the ball to home goal are taken as the inputs, the fuzzy system utilized is still a rather simple one. However, a more complicated rule base increases the computation load and slow down the system which is definitely unacceptable in a real-world robot soccer match.

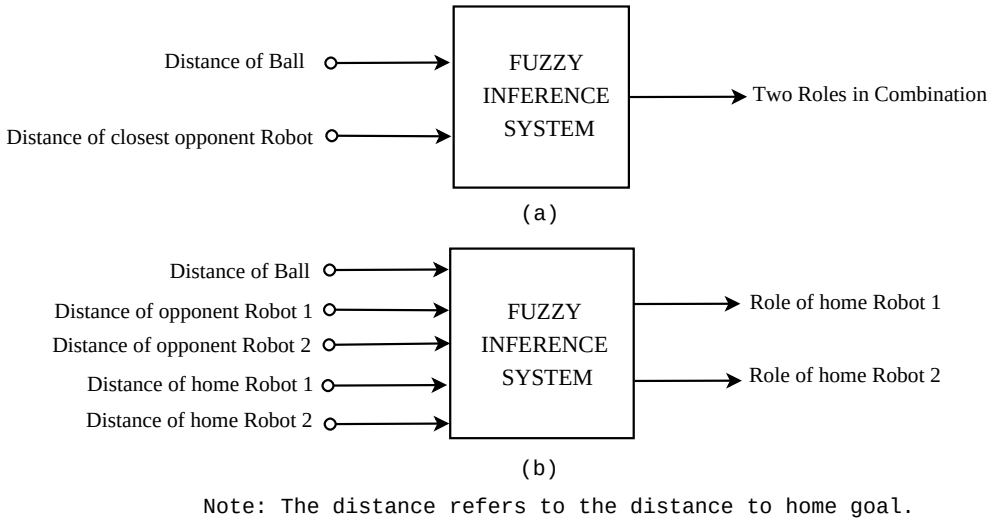


Figure 8.6: The two-input and five-input systems

In this section, the fuzzy role assignment problem for robot soccer system is considered again. As the major purpose here is to study the performance of the DNA coding method, a more complex fuzzy rule base is developed for role assignment and it is evolved by genetic algorithm in simulation, while the real world performance of the robot soccer system is not looked into.

8.4.1 Coding mechanisms

Compared to the original two-input system from Section 7.5.3 (Figure 8.6.(a)), five input variables are considered for the new fuzzy system (Figure 8.6.(b)). The five inputs are the position of the ball to the home goal, the positions of the two opponents and the positions of the two home robots. Each variable has three linguistic values: near, medium and far. Two output variables are the roles of the two robots other than the goalie, which may be the attacker, midfielder and defender. Consequently, there are $3^5 = 243$ input states and totally $3^7 = 2187$ possible rules. The evolution is focused on the rule base of the fuzzy system while the membership functions are kept intact.

Three coding methods are used for the comparison. The first one is the standard

integer coding. The others are two DNA coding methods, named as DNA coding 1 and 2. The difference between the two DNA codings is that DNA coding 1 contains no intron while the DNA coding 2 does. The four characters representing the DNA base – A, C, G and T – are employed to build the chromosome string. The biological term "codon" is borrowed to indicate the three-letter triplets and/or two-letter couplets in the DNA coding method.

The start/stop codon is not used in the DNA coding methods proposed here, which marks an apparent departure from the DNA coding methods in [131, 132, 133]. The using of start/stop codon separates the individual strings into segments of useful genes and meaningless introns. Each segment of gene is mapped to a fuzzy rule. If the structure of rule is fixed, whose number of parameters are already known (just like the case of robot soccer system here), the length of gene segment needs to match the rule. Too short a gene results in an invalid rule while too long a gene wastes a lot of codons. Both the cases result in some kind of inefficiency in decoding useful rules from individual chromosomes. However, it is difficult to control the number or the length of useful genes as the start/stop codons are randomly distributed. Another effect of the start/stop codon is that it allows the overlap reading of the genes, which means that the gene segments can share some portions. However, there is no empirical evidence to show the advantages of such a overlapping. Furthermore, if necessary, the overlapping can be easily realized by rereading the chromosome from another starting point chosen in a random or heuristic way, without the definition of start/stop codon. As a result, the start/stop codon is not defined in this work.

The details of the three coding methods used here are further discussed in the following.

Genetic algorithm using integer coding method

Using the integer coding method to encode the fuzzy rule base is quite straightforward. Each individual string represent a complete fuzzy rule base, which includes all the input states which are indexed from 1 to 243. The length of the individual

1 2	3 4	5 6	7 8	9 10	11 12	.....	483 484	485 486

1- Output role for robot number 1 under input state 1
2- Output role for robot number 2 under input state 1
3- Output role for robot number 1 under input state 2
4- Output role for robot number 2 under input state 2
5- Output role for robot number 1 under input state 3
6- Output role for robot number 2 under input state 3
7- Output role for robot number 1 under input state 4
8- Output role for robot number 2 under input state 4
. . . . . .
. . . . . .
483- Output role for robot number 1 under input state 242
484- Output role for robot number 2 under input state 242
485- Output role for robot number 1 under input state 243
486- Output role for robot number 2 under input state 243

Figure 8.7: Structure of the chromosome encoded with integer coding method

string is of $243 \times 2 = 486$ integers and fixed through the GA process. The integer set {1, 2, 3} is mapped to the three output linguistic values: attacker, midfielder and defender. As there are two robot roles to be decided, every two consecutive integers stand for the output resulting from one indexed input state (Figure 8.7). The coding method is a typical position dependent one, because the meaning of every integer is merely determined by its absolute position. In this case, for the n-th integer of the individual string: if n is odd, the integer stands for the output role of robot number 1 under the input state indexed as $\lfloor \frac{n}{2} \rfloor + 1$; if n is even, it indicates the role of robot number 2 for the $\lfloor \frac{n}{2} \rfloor$-th input state.

The GA population size is set to 200. The stochastic universal sampling (SUS) [124] with elitism selection is used, while the number of elite individuals are set to 3. Uniform crossover [125] and random multiple points mutation are adopted. The crossover probability P_c and mutation probability P_m are set as 0.6 and 0.05, respectively.

	A	C	G	T
A	Codon 1	Codon 1	Codon 2	Codon 2
C	Codon 1	Codon 1	Codon 2	Codon 2
G	Codon 1	Codon 3	Codon 3	Codon 2
T	Codon 3	Codon 3	Codon 3	*

Table 8.3: Index of the two-letter codons with DNA coding method 1

Input/Output Variables	Codon 1	Codon 2	Codon 3
Distance: Ball to own goal	Near	Medium	Far
Distance: Opponent 1 to own goal	Near	Medium	Far
Distance: Opponent 2 to own goal	Near	Medium	Far
Distance: Robot 1 to own goal	Near	Medium	Far
Distance: Robot 2 to own goal	Near	Medium	Far
Role: Robot 1	Attacker	Midfielder	Defender
Role: Robot 2	Attacker	Midfielder	Defender

Table 8.4: Translation from codons to fuzzy rules with DNA coding method

Genetic algorithm using DNA coding method 1

In the DNA coding method 1, the four DNA nucleotides, A, C, G and T, make up the coding alphabet. Each individual is a string of characters randomly selected from the alphabet. Different from the natural three-letter protein codons, totally 16 two-letter couplets are employed and mapped onto three index codons (Table 8.3). Fifteen two-letter couplets are randomly grouped to represent codons 1, 2 and 3, which are in turn used to encode the three linguistic values for both the five inputs and two outputs. The sixteenth couplet is used as a wildcard codon, which is randomly mapped to codon 1, 2 or 3. That is to make sure that the three codons have equal chances to appear in the individual string. In this way, all the 16 couplets have specific meaning.

Following the translation Table 8.4, the genes are decoded to fuzzy rules, which

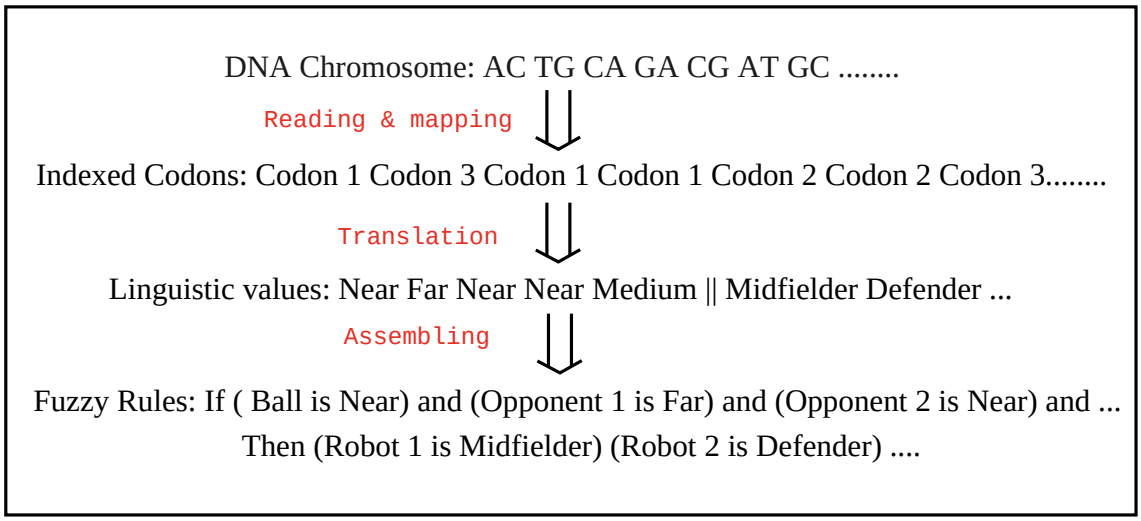


Figure 8.8: Decoding process for DNA coding method 1

make up the final rule base. Among the resultant fuzzy rules, some of them may have the same antecedent parts standing for same input states. Only the first rule among them is included in the rule base. As all the couplets are meaningful and there is no start/stop point, the individual chromosome contains no meaningless part (intron). The decoding process from DNA strings to fuzzy rules is depicted in Figure 8.8. The meaning of codons is partially related to their absolute positions. For instance, the codon in the 3rd, 11th and 19th positions are always translated to the 3rd input variable: the distance of the opponent robot 2 to home goal. However, to which input state the input variable belongs remains undecided. This codon needs to be combined with the two codons ahead, which are associated to the other two input variables' values, to depict a definitive antecedent of a fuzzy rule. In this point of view, the DNA coding method 1 is context dependent. On the other hand, each codon are represented by at least five two-letter couplets, resulting in some kind of redundancy. The individual string can be of variable length. The SGA operators are used while the validity of individual strings remain unaffected.

Similar to the GA with integer coding, the population size is set to 200. Individual chromosome's initial length is set to 500 characters, and the string length is allowed to change throughout the GA process. The genetic operators and the parameter settings (P_c, P_m) are almost the same as that of GA with integer coding,

	A	C	G	T
A	–	–	Codon 1	Codon 1
C	Codon 3	–	Codon 2	–
G	–	–	–	Codon 2
T	–	Codon 3	–	–

Table 8.5: Index of the two-letter codons with DNA coding method 2

except that the random two points crossover are used here to replace the normal crossover.

Genetic algorithm with DNA coding method 2

The DNA coding method 2 defers from the method 1 in the mapping mechanism between couplets and codons. Among the 16 two-letter couplets, only 6 of them are meaningful while the others are meaningless (Table 8.5). The selection of these six meaningful couplets are rather arbitrary. The only requirement is that the characters in the alphabet – A, C, G and T – have equal number of instances in these six couplets. In other words, these six couplets should have the same chance of appearance in chromosomes. The existence of the meaningless couplets introduces intron parts into individual chromosomes. If such as intron part is read in the chromosome, it is just bypassed and the meaningful parts are translated to fuzzy rules using the same translation table used in DNA coding method 1 (Table 8.4 and Figure 8.9). No start/stop codon is used but the introduction to introns already makes the coding method as a context dependent one. The other features of DNA coding method 1 like redundancy and variable string length are also existing in DNA coding 2.

The genetic algorithm with DNA coding method 2 uses the SUS with elitism selection, two-point crossover and multi-point mutation operators. The parameter settings are also the same as those used in the case of method 1.

There are other considerations for the DNA coding method representing fuzzy

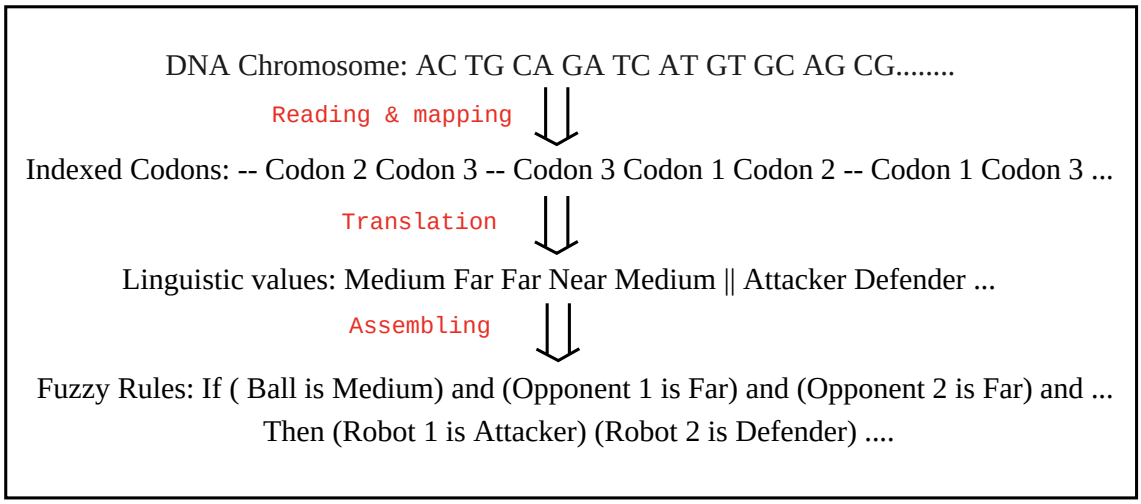


Figure 8.9: Decoding process for DNA coding method 2

rule bases. In both the methods 1 and 2, one valid fuzzy rule is represented by 7 meaningful couplet codons. The number of valid fuzzy rules represented by an individual is related to the string length. Since the string length may vary during evolution, there could be some extremely long individual strings containing lots of fuzzy rules. However, totally 243 fuzzy rules with different antecedents are enough to set up a complete rule base covering all the inputs states. It is unnecessary for an individual to contain too many fuzzy rules. Furthermore, one promising purpose of DNA coding here is to find a smaller but yet effective rule base covering only the most important input states. In consequence, an upper-limit (denoted as R_{num}) is set to the number of fuzzy rules decoded from one individual. R_{num} is usually chosen as 100 in the simulation, while the setting of 250 and 25 are also tested for comparison.

Since the size of rule base represented by an individual is varied with the string length and is limited by R_{num}, it is normal to find in the simulation that no fuzzy rule is fired at some time steps. This means that the input states at those instances are not covered by the current rule base. At such moments, a default role setting is chosen as the output. At the beginning of the simulation, the default roles are the attacker for one robot and the defender for the other. After that time, the default output is simply the role assignment of the last time step. Since the input space of the fuzzy controller is continuous, it is reasonable to assume that the input state

of the current step is closer to that of the last step. Hence their outputs should share some similarity too.

8.4.2 Simulation results

Simulations are performed on the robot soccer simulator explained in Chapter 7. The rule base under evolution is evaluated by competing with a team using the original rule base (Chapter 7). Two teams of robots are pitched for a match for a certain period (set as 600 steps). The game also ends immediately when the total number of goals scored reaches a limit. The limit is set to 3 there, which is found to be the maximum number of goals can be score by both teams in 600 steps. Similar to the real world human soccer game, the performance of a team is summarized by some statistical data. The first data is of course the match score. Secondly, the time duration for which the ball is within which side of the field is recorded. Usually, the ball in one side of the field implies that the team at this side is under attack. Different from the fitness function used in Chapter 7, one more factor is considered, which is the control of the ball. Except for the two goalies, the robot which is nearest to the ball is regarded as the possessor of the ball. The fitness function F_{role} is constructed as follows:

$$F_{role} = u + v \cdot (Goal - Lost) + w_1 \cdot \frac{T_{ball}}{T_{total}} + w_2 \cdot \frac{T_{opp}}{T_{total}} \tag{8.1}$$

where u, v, w_1 and w_2 are integer constants, $Goal$ and $Lost$ are scores of the home and opponent teams, T_{ball} is the time for which the ball is under the control of the home team, T_{opp} is the time for which the ball is in the opponent half of the field, and T_{total} is the total match time. With a weight w_1 of 100, the ratio of T_{ball} to T_{total} implies which team has possession in the game. The ratio of T_{opp} to T_{total} is an indication of which team takes the upper hand in the game and the associated weight (w_2) is set to 80. It is observed that the score is indeed not a stable performance index. Furthermore, the two teams seldom score in the match due to the rather short period of 600 steps. As a result, the weight attached to the score difference (v) is set to a relative small value of 10. Since a team may lose

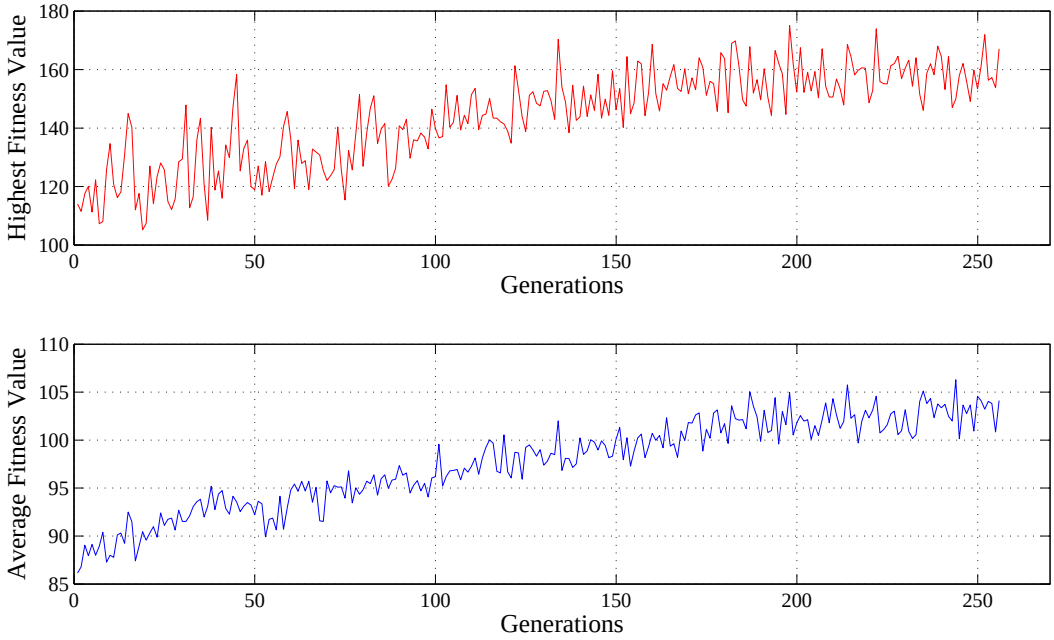


Figure 8.10: Fitness curve for integer coding method

the game, the second part of F_{role} can be negative. u is set to 30 and is utilized to keep F_{role} non-negative.

In the simulation, the random nature of the soccer competitions brings in some degree of variation in fitness evaluations. The fitness value of the same chromosome may not be constant. Different rule bases, though similar, are evolved from different GA runs with the same coding method. The average values across 50 different runs for each coding method are depicted as fitness value curves (Figure 8.10 to 8.13), in which fluctuations are noticed. The fluctuation is especially serious in the "highest fitness value" curves. However, the trend and property of the evolution process are still observable.

When the integer coding method is used, the fitness values for each generation are summarized in Figure 8.10. Both the average fitness value and the highest fitness value of the whole population cease to improve after 200 iterations. The average fitness value falls in the range between 100 and 105, while the highest fitness value is stabilized around 160.

The DNA coding method 1 is used with the upper-limit on the number of fuzzy rules decoded from an individual set to 100 ($R_{num} = 100$). In other words, each

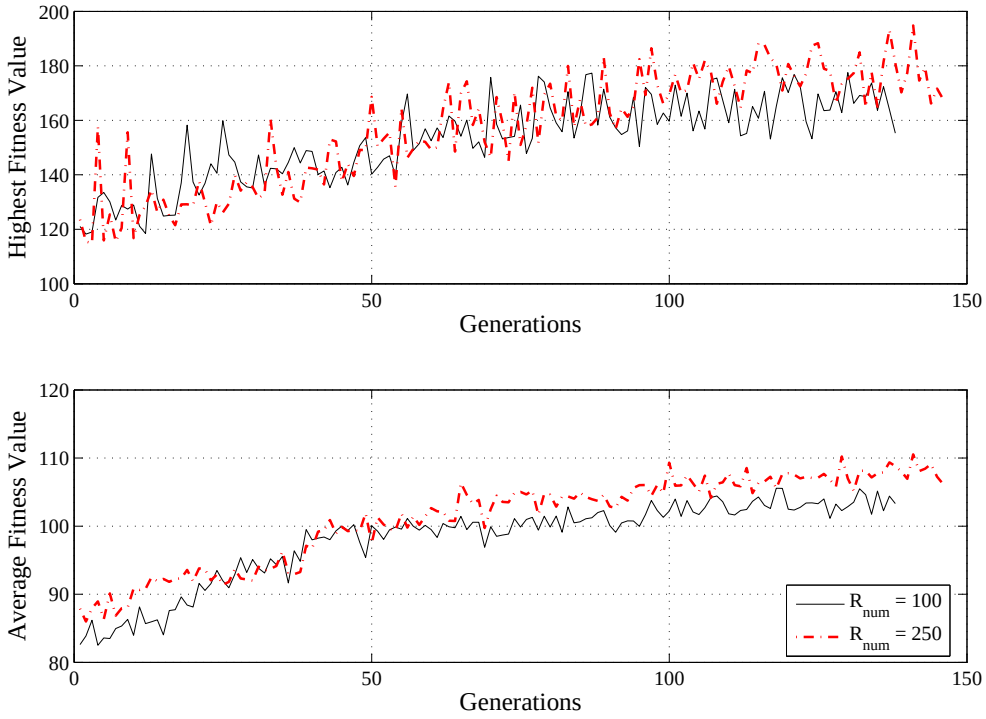


Figure 8.11: Fitness curve for DNA coding method 1

individual represents a fuzzy rule base containing at most 100 rules. The size of the fuzzy rule base is less than half of the one represented by integer coding method. In the simulation, the GA process converges around 125 iterations. The average fitness and highest fitness values are stabilized around 103 and 165, respectively (Figure 8.11). In the case of $R_{num} = 250$, both the average and highest fitness converge to a higher value (Figure 8.11). The range of improvement on fitness value is from 5 to 10.

The same setting of R_{num} for DNA coding method 1 is also applied to DNA coding method 2. The evolutionary process with DNA coding method 2 usually reaches the convergence to around 140 generations. The final average fitness value lingers around 110 and the highest fitness value fluctuates slightly around 180 (Figure 8.12). If the R_{num} increases to 250, the average and highest fitness curve raise to the levels around 115 and 185, respectively.

The fitness curves of GAs using three coding methods are compared in Figure 8.13. R_{num} is set to 100. The staring points of the fitness curves for three coding methods are almost equal. Despite of the fluctuations, it is observable that the

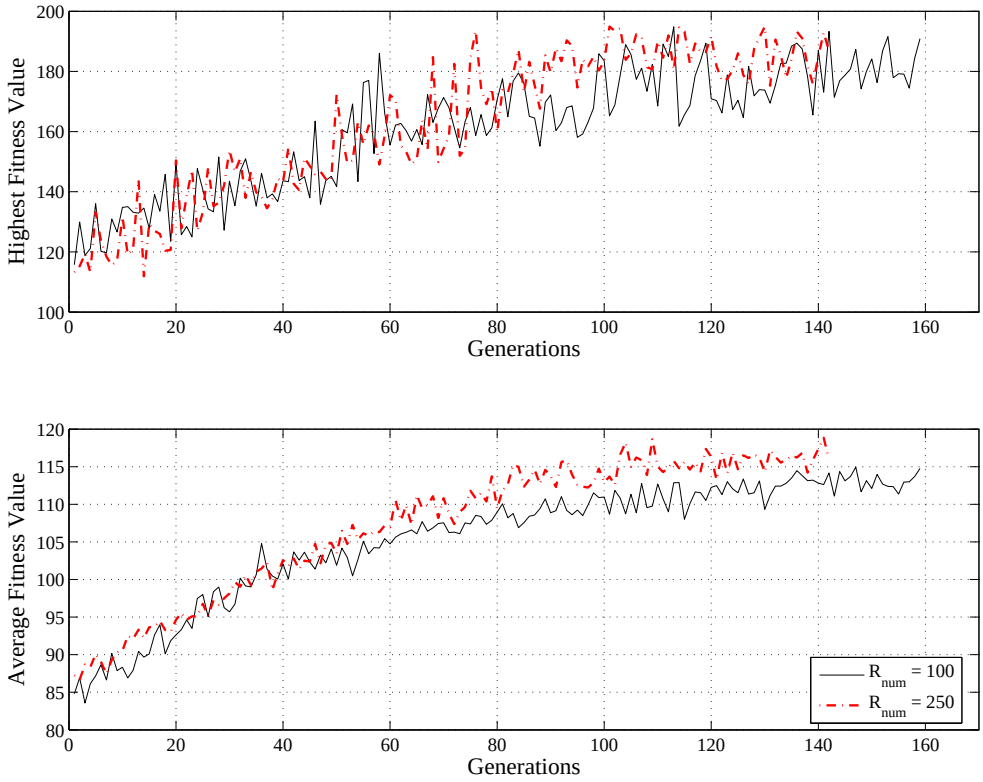


Figure 8.12: Fitness curve for DNA coding method 2

fitness curves with DNA coding 1 reach the same level as the curve with integer coding does (Figure 8.13), if not slightly higher. Nevertheless, the DNA coding 2 obviously provides the best fitness curves in respect of both the average fitness or highest fitness.

Performance validation is carried out with the evolved rule bases and the original rule base through simulation soccer matches. The performance data (the scores, number of shots, $\frac{T_{ball}}{T_{Total}}$ and $\frac{T_{opp}}{T_{Total}}$) of the rule bases from the same coding method are collected and averaged. The average performance data for each coding method are compared in Table 8.6.

From the fitness comparison, it seems that DNA coding methods outperform the integer coding method in this problem, although the size of rule base represented by a DNA coded chromosome is only 100 rules at most. One important reason for such a performance is due to the interactions inside the fuzzy rule base. For the fuzzy rule base discussed here, the fuzzy rules are in fact not independent from each other. Rules with similar antecedents should also have related consequents. More

Performance Data	Integer Coding		DNA Coding 1		DNA Coding 2	
	Original	Evolved	Original	Evolved	Original	Evolved
Scores	0.3	1.6	0.2	1.4	0.2	2.0
Number of shots	11.7	18.0	12.6	17.2	10.4	18.8
(T_{ball}/T_{Total})	33.7%	66.3%	31.6%	68.4%	23.9%	76.1%
(T_{opp}/T_{Total})	36.9%	63.1%	34.9%	65.1%	28.2%	71.8%

Table 8.6: Comparison of simulation match performances

importantly, the results of certain fuzzy rules change the situation in the match field and in turn trigger certain other rules. In the position dependent integer coding method, each rule has mapped to a fixed position in the string. The nonlinear causality between rules results in a nonlinear interaction of characters/codons at different positions. The fitness of one character/codon at one position may depend on the value of other characters/codons. This effect is usually referred as epistasis [155, 156], which may decrease the performance of GA [157, 158]. According to the schema theorem, GA works well if a string with high fitness can be build from short, low-order and above-average schemata. Thus, one of the basic requirement of GAs to be successful is low epistasis of the problem. To this problem, it is justifiable to assume that a rational schema should contain the characters interacting with each other. However, if the positions of two characters are far from each other, the schema will be long and liable to destruction by GA operators. For the fixed coding method, this problem is unavoidable. But for the context dependent DNA codings, the rules are not fixed to absolute positions. Those nonlinear related characters may be moved together during the GA operations, resulting in short and reasonable schemata. The position-independent parameters in the individual string in fact alleviate the difficulty caused by epistasis. That is a major advantage of DNA coding methods.

Meanwhile, both the DNA coding method 1 and 2 have some degree of redundancy in the coding. Referring to Tables 8.3 and 8.5, each indexed codon is represented by more than one couplets. The result of such a redundancy is that two individuals displaying the same phenotype (i.e. representing the same fuzzy rule

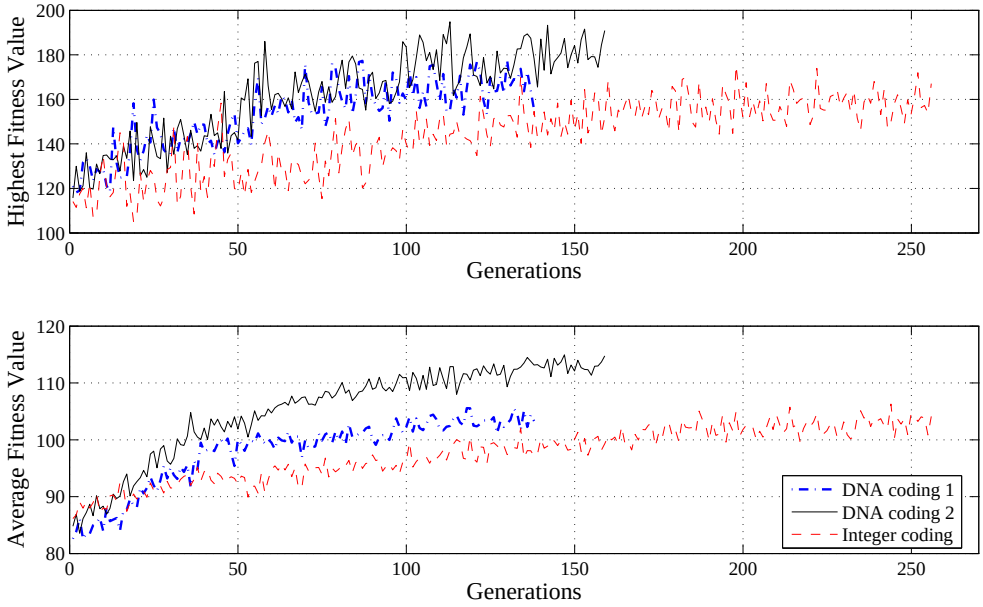


Figure 8.13: Fitness curve comparison for the three coding methods

base) may have different genotype (different strings). A crossover between them may produce offsprings with different phenotype. This feature helps to increase the population diversity which may be another benefit with DNA coding methods.

Furthermore, the DNA coding method 2 introduces introns into individual strings. Researchers have observed that introns may lead to considerable improvement in performance of genetic algorithms [148, 154] and genetic programming [159]. In this work, DNA coding method 2 demonstrates a better performance than that of DNA coding 1 (Figure 8.13). It seems that the meaningless parts (introns) in the string play a useful role in absorbing disruption caused by genetic operations. They also provide themselves as building materials which may be transferred to meaningful codons at any time during GA operations. This is some kind of enlargement of the searching space covered by current population and also an increase in the population diversity.

Besides the fitness values, some other parameters related to DNA coding's performance are also looked into. The fuzzy rule base decoded from an individual is evaluated in a match lasting for 600 steps. A counter C_{fire} is set up for each individual to denote how active the rule base is. In each step, if at least one rule in the rule base is triggered, the counter is increased by unity. Thus the maximum

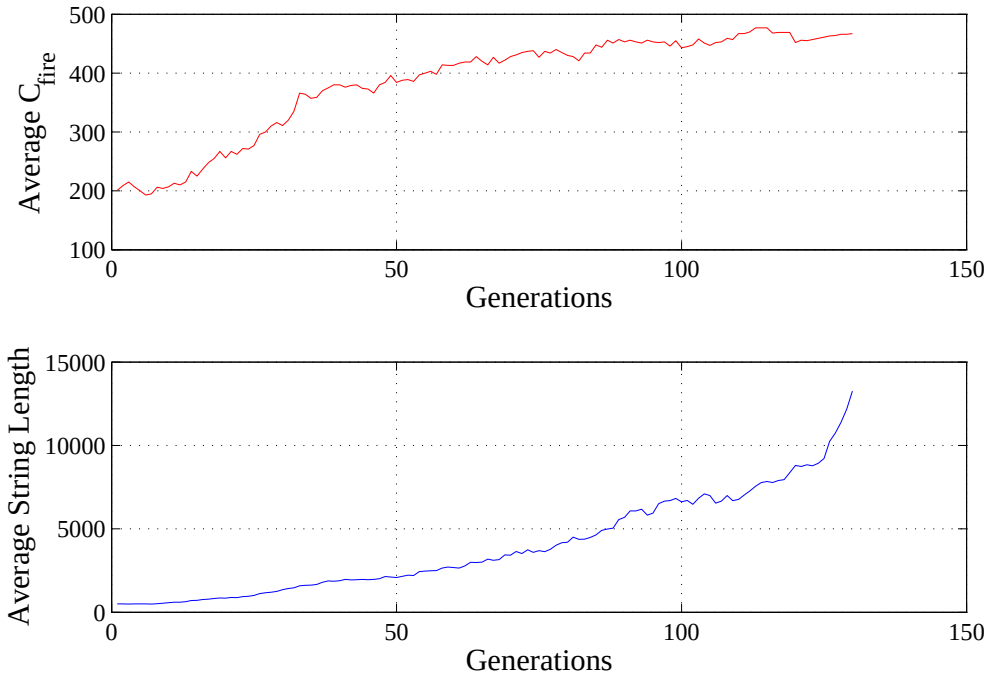


Figure 8.14: The change of C_{fire} and string length throughout evolution

of C_{fire} is 600. The counter remains unchanged when none of the fuzzy rules is fired and the output is determined by the default setting. A bigger value of C_{fire} indicates a more active rule base. A fuzzy rule base has to be active enough at first before taking any effects. In fact, C_{fire} is related to how many input states are covered by the fuzzy rule base, which is in turn associated with the size of rule base and the length of the individual string. Both the average of C_{fire} and length of the population are plotted in Figure 8.14 for the DNA coding method 2 with R_{num} as 100.

The initial length for DNA coding chromosomes is set to 500 characters. With this length, a chromosome contains at most 35 valid fuzzy rules, which implies a quite small rule base. The average C_{fire} is about 200, which means the fuzzy rule base is not active for two-thirds of the time. As the evolution goes on, the average length of all individuals increases quickly. At the end of evolution, the average length is around 13000 characters. The chromosome then contains about 6500 two-letter couplets, in which approximately six-sixteenths of them are valid codons (Table 8.5). As a result, about $6500 \times \frac{6}{16} \times \frac{1}{7} \approx 348$ valid rules can be decoded. The rules with duplicated antecedent parts are further filtered. The final

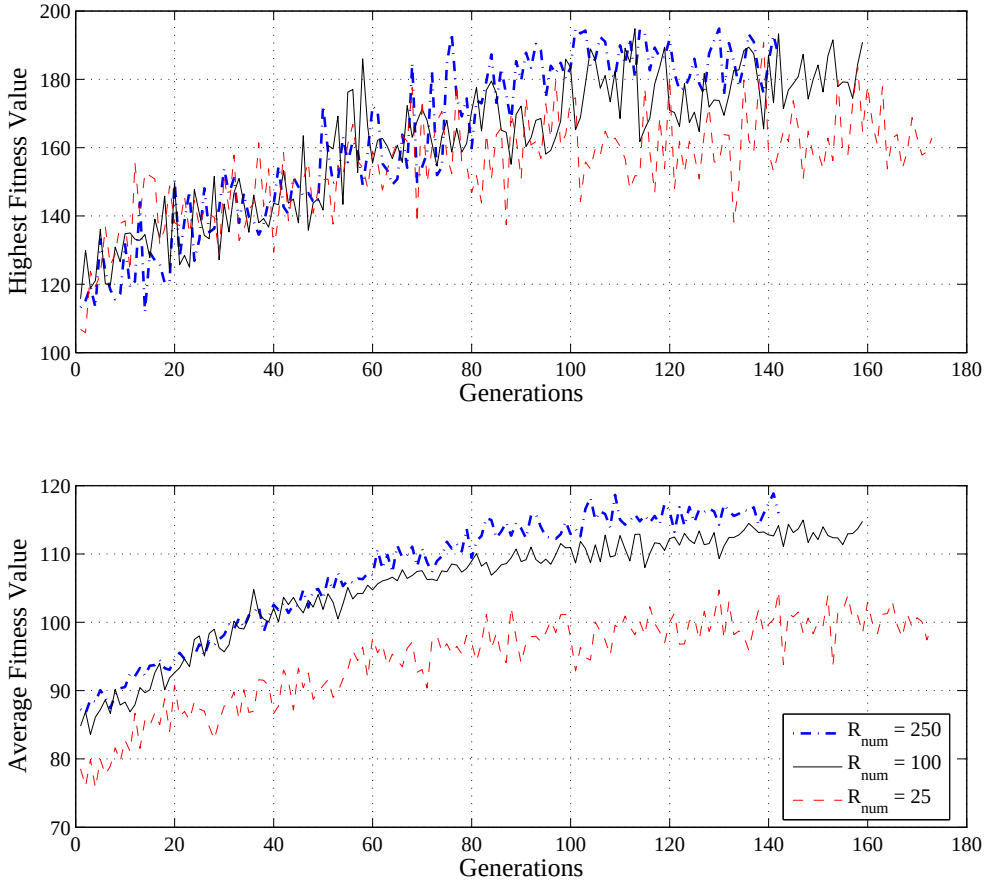


Figure 8.15: Fitness curve comparison for different R_{num} settings

rule base should already reach the limited size of 100 rules. Consequently, the average C_{fire} increases steadily from 200 to 460. The rule base becomes more and more effective as its size increases. The DNA coding allows individual to be of variable length. That means the size of the encoded fuzzy rule base is also flexible and can be evolved by GA.

The benefits of the variable size for fuzzy rule base seems not prominent at first glance. It is obvious that the bigger rule base is always preferred with respect to the coverage of input states. If the R_{num} is not imposed, the size of rule base surely grow to the maximum of 243 during evolution. However, it seems that the effectiveness of the rule base is not linear to its size. While the size is limited by $R_{num} = 100$, the rule base is already active for 76% (460/600) of the time. To study the influence of the variable R_{num} on the GA process, the fitness curves with different setting of R_{num} are depicted in Figure 8.15. While the R_{num} is increased

from 100 to 250, the size of the resultant rule base is more than doubled. However, the improvements on both the average fitness and highest fitness are just around $2\% \sim 3\%$ (Figure 8.15). A even smaller R_{num} of 25 is tested. With such a small rule base, the decline of fitness curves is only about $12\% \sim 15\%$.

Such results suggested that not all the rules in the rule base are of the same importance. Some of them are related to those most crucial input states and thus dominate the performance of the fuzzy rule base. A compact rule base containing only those important rules will be more computationally efficient than the complete rule base covering all the input states. This feature will become more and more meaningful as the fuzzy rule base getting more complicated and bigger. On the other hand, it is usually hard to tell which input states are more important at the design stage. With DNA coding methods, GAs can automatically search for and optimize the crucial rule base with different size settings. As to the integer coding method, the input states are attached to absolute positions on strings. Under this circumstance, GAs can not optimize the fixed structure of the rule base.

8.5 Conclusion

In this chapter, the context dependent DNA like coding methods are discussed and implemented to the evolution of fuzzy rule base. DNA coding methods are flexible and easy to use. It can be directly used with the standard genetic operators. Two types of DNA coding methods, with and without the intron parts, are compared with the traditional integer coding method. GAs with the three coding methods are applied to the fuzzy role assignment problem in a robot soccer system.

The results shows that DNA coding methods outperform the integer coding method in this problem. The context dependent coding can handle the negative effect of epistasis, which degrades the performance of the position dependent coding. The redundancy in the DNA coding increases the population diversity. The DNA coding with intron parts displays an even better performance. It seems that

the intron parts are helpful in preventing useful schemata from disruption and increasing population diversity. The DNA coded individuals are of variable length. Such a feature provides GA with the possibility to evolve both the size and the structure of the fuzzy rule base. As the result, a compact but efficient rule base can be developed by DNA coded GA.

Chapter 9

Conclusions and Future Directions

9.1 Conclusions

Following the introductory chapters, an extensive fuzzy behavior based architecture for multiple robotic system is proposed in Chapter 5. The decomposition of a system into various simple behaviors in a hierarchy fashion is a promising way for implementing complex control systems. Such a hierarchy architecture also leads to a distributed fuzzy control system constructed from simple sub-systems, instead of a centralized and complex one. The fuzzy behavior based architecture is realized directly and comprehensively on the actual robot soccer hardware. Encouraging performance is obtained in the experimentations as well as at robot soccer official matches. The soccer robots are observed to move with better agility and greater purpose. Meanwhile, the real world approach poses a lot of practical considerations and limitations. The results obtained under such conditions can be put into better perspective.

In Chapter 6, an adaptive tuning method is applied to the fuzzy behavior based robot soccer system proposed in Chapter 5. Fine tuned by the method outlined in Chapter 6, fuzzy actions have achieved improvements in terms of less overshoot, shorter rise-time and shorter settling-time, as well as smaller steady-state error. The robots as the attacker or defender have shown the desired offensive or defensive behaviors whenever necessary. The team strategy, together with the adaptively

tuned behaviors and actions, is capable of achieving better performance in a dynamic match environment. The proposed mechanism provides certain adaptive tuning ability to the system for handling deleterious environmental and system changes. With the feedback data, the system can adjust its fuzzy behaviors to regain the performance. In comparison with the manual trial and error approach, the method proposed is more easy to use. Furthermore, the parameter files of behaviors are able to be loaded on-line, which enable on-line switching of different pre-defined settings catering to different strategies. However, the adaptive tuning process is still manually activated, due to the limited computation capacity of the current hardware setting.

In Chapter 7, genetic algorithm is employed to develop and optimize fuzzy behaviors at different levels of the fuzzy behavior based architecture. Although the original system in Chapter 5 is already well designed, noticeable improvements were achieved at different levels of the architecture. According to the results of the simulation and real-world experimentations, the accuracy of primitive behaviors are obviously enhanced. More skillful behaviors (like shoot to the far-end corner of the goalie) are developed, as well as a more effective role assignment mechanism which embodies the group behavior. It is indeed desirable and useful to introduce evolutionary algorithms into the fuzzy behavior based multi-robotic systems. Meanwhile, the evolutionary algorithm provides an efficient way to develop and optimize a complex multi-robotic system, especially when there is not enough knowledge of the system for the development in a manual fashion.

Context dependent DNA like coding methods are discussed in Chapter 8. Genetic algorithms with three coding methods, two for DNA coding and one for integer coding, are implemented to evolve the fuzzy rule base for role assignment task in the robot soccer system. Some beneficial features of DNA coding methods are explored. At first, the DNA like coding method is flexible. With different translation rules, the individual string can be used to represent numeric values, linguistic words, or even structural information. Meanwhile, the compatibility with standard genetic operators is not compromised. Secondly, the feature of context

dependency is helpful in dealing with the detrimental impacts of epistasis. Furthermore, the redundancy in the DNA coding increases the population diversity. The existence of intron parts can decrease the chances of useful schemata being disrupted and increase the population diversity as well. The last but not the least, the DNA coded GA can evolve both the size and the structure of the fuzzy rule base, resulting in a compact but efficient rule base.

9.2 Future Directions

Based on the works in this thesis, the future research can be carried out along the following directions.

The fuzzy behavior based architecture outlined in Chapter 5 is proposed as a general control architecture for multiple robots systems. Obviously, it can be implemented to other multi-robotic systems as well. The architecture itself can be improved and refined in such applications. Other control methods, such as the traditional PID control, neural networks or expert systems, can be included to form a hybrid structure.

The adaptive tuning mechanism in Chapter 6 is manually triggered. Although the computation power of current system cannot support an extensive on-line monitoring and feedback system, future research in this direction is very meaningful. Under the current system capacity, it is feasible to set up a rule base for self-alteration of game strategies and certain crucial behaviors.

As to the evolutionary fuzzy system in Chapter 7, there are also space for improvement. The refinement on the kinetic model of soccer robot will definitely improve motion control of robots, both in real world and in simulation. For the evolution at the strategy level, designing a better fitness function for GA is crucial for further improvement. On the other hand, co-evolution of related fuzzy behaviors and fuzzy roles seems to be a promising direction.

The DNA coding methods discussed in Chapter 8 are not limited to encoding

the fuzzy system. Under the general scheme, DNA coding is not defined by using the alphabet based on DNA bases, but the features. It is quite suitable to represent the neural networks in evolutionary-neural systems. On the other hand, applying the DNA coding methods to the genetic programming will also be an interesting direction.

Bibliography

[1] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[2] Lotfi A. Zadeh. Preface in fuzzy logic technology and applications (by Marks-II R.J.). IEEE Technical Activities Board, 1994.

[3] Vilem Novák, Irina Perfilieva, and Jiri Močkoř. *Mathematical principles of fuzzy logic*. Kluwer Academic Publishers, Boston, 1999.

[4] Roberto L. O. Cignoli, D'Ottaviano Itala M.L, and Daniele Mundici. *Algebraic Foundations of Many-Valued Reasoning*, volume 7 of *Trends in Logic*. Kluwer, Dordrecht, nov 1999.

[5] Siegfried Gottwald. *A Treatise on Many-Valued Logics*, volume 9 of *Studies in Logic and Computation*. Research Studies Press, Baldock, November 2000.

[6] Vilem Novák. *Fuzzy Sets and their Applications*. Adam Hilger, Bristol, 1989.

[7] Hung T. Nguyen and Elbert A. Walker. *A First Course in Fuzzy Logic*. CRC Press International, 1996.

[8] I. G. Umbers and P. J. King. An analysis of human decision-making in cement kiln control and the implications for automation. *International Journal of Man-Machine Studies*, 12(1):11–23, 1980.

[9] C. C. Lee. Fuzzy logic in control systems: Fuzzy logic controller – part II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):419–435, March-April 1990.

[10] G. Antonelli and S. Chiaverini. Fuzzy redundancy resolution and motion coordination for underwater vehicle-manipulator systems. *IEEE Transactions on Fuzzy Systems*, 11:109– 120, Feb 2003.

[11] C. L. Lin, H. Z. Hung, Y.-Y. Chen, and B. S. Chen. Development of an integrated fuzzy-logic-based missile guidance law against high speed target. *IEEE Transactions on Fuzzy Systems*, 12, April 2004.

[12] Lotfi. A. Zadeh. The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems*, 11(3):199–227, 1983.

[13] Brian R. Gaines and Mildred L. G. Shaw. Induction of inference rules for expert systems. *Fuzzy Sets and Systems*, 18(3):315–328, 1986.

[14] Tadeusz Radecki. A theoretical background for applying fuzzy set theory in information retrieval. *Fuzzy Sets and Systems*, 10(2):169–183, 1983.

[15] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.

[16] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[17] David E. Goldberg. *The Design of Competent Genetic Algorithm: Steps Toward a Computational Theory of Innovation.* Kluwer Academic Publishers, Dordrecht, 2002.

[18] David E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[19] David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Don't worry, be messy. In Richard K. Belew and Lashon B. Booker, editors, *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, pages 24–30, San Mateo, CA, 1991. Morgan Kaufmann.

[20] David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid accurate optimization of difficult problems using fast messy genetic

algorithms. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 56–64, San Mateo, CA, 1993. Morgan Kaufmann.

[21] V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 177–183, San Mateo, CA, 1993. Morgan Kaufmann.

[22] Erick Cantú-Paz. A summary of research on parallel genetic algorithms. Technical Report 95007, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999.

[23] Lourdes Araujo. A parallel evolutionary algorithm for stochastic natural language parsing. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villacañas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII*, pages 700–709, Berlin, 2002. Springer.

[24] S. Oliker, M. Furst, and O. Maimon. A distributed genetic algorithm for neural network design and training. *Complex Systems*, 6:459–477, 1992.

[25] F. Herrera and M. Lozano. Gradual distributed real-coded genetic algorithms. *IEEE-EC*, 4(1):43, April 2000.

[26] M. G. Arenas, Pierre Collet, A. E. Eiben, Márk Jelasity, J. J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer. A framework for distributed evolutionary algorithms. In Juan Julián Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José-Luis Fernández-Villacañas, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII*, pages 665–675, Berlin, 2002. Springer.

[27] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *Proc. of the First Int. Conf. on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, 1985. Lawrence Erlbaum.

[28] Kalyanmoy Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[29] P. J. Bentley and J. P. Wakefield. Finding Acceptable Solutions in the Pareto-Optimal Range using Multiobjective Genetic Algorithms. In P. K. Chawdhry, R. Roy, and R. K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, Part 5, pages 231–240, London, June 1997. Springer Verlag London Limited. (Presented at the 2nd On-line World Conference on Soft Computing in Design and Manufacturing (WSC2)).

[30] David B. Fogel. Evolutionary algorithms in engineering applications. *IEEE Transactions on Evolutionary Computation*, 2(2):72, July 1998.

[31] Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, 2002.

[32] Jürgen Branke. Evolutionary algorithms in neural network design and training – A review. In Jarmo T. Alander, editor, *Proc. of the First Nordic Workshop on Genetic Algorithms and their Applications (1NWGA)*, pages 145–163, Vaasa, Finnland, 1995. Department of Information Technology and Production Economics, University of Vaasa.

[33] F. H.F. Leung, H. K. Lam, S. H. Ling, and P. K.S. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE-NN*, 14:79– 88, Jan 2003.

[34] Paul G. Harrald and Mark Kamstra. Evolving artificial neural networks to combine financial forecasts. *IEEE Transactions on Evolutionary Computation*, 1(1):40–52, April 1997.

[35] Kumar Mehta and Siddhartha Bhattacharyya. Combining rules learnt using genetic algorithms for financial forecasting. In *1999 Congress on Evolutionary Computation*, pages 1245–1252, Piscataway, NJ, 1999. IEEE Service Center.

[36] Lawrence Davis, David Orvosh, Anthony Cox, and Yuping Qiu. A genetic algorithm for survivable network design. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 408–415, San Mateo, CA, 1993. Morgan Kaufmann.

[37] H. G. Sandalidis, P. P. Stavroulakis, and J. Rodriguez-Tellez. An efficient Evolutionary Algorithm for Channel Resource Management in Cellular Mobile Systems. *IEEE Transactions on Evolutionary Computation*, 2(4):125, November 1998.

[38] Gibilisco Stan, editor. *The McGraw-Hill Illustrated Encyclopedia of Robotics and Artificial Intelligence*. McGraw-Hill Inc., 1994.

[39] Robin R. Murphy. *An Introduction to AI Robotics*. MIT Press, 2000.

[40] Robert K. Lindsay, Bruce G. Buchanan, Edward A. Feigenbaum, and Joshua Lederberg. *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*. McGraw-Hill, New York, 1980.

[41] B. G. Buchanan and E. H.(eds) Shortliffe. *Rule-based Expert Systems: The MYCIN experience of the Stanford Heuristic Programming Project*. Addison-Wesley (Reading MA), 1985.

[42] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, MA, 1998.

[43] R.A. Brooks. A robust layered control system for mobile robot. *IEEE Journal of Robotics and Automation*, RA-2, no.1:14–23, 1986.

[44] Ronald C. Arkin. *Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-made Environments*. PhD thesis, University of Massachusetts, 1987.

[45] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 809–815, San Jose, CA, USA, July 1992. AAAI Press.

[46] D. M. Lyons and A. J. Hendricks. Planning, reactive. In *Encyclopedia of Artificial Intelligence*, pages 1171–1181. John Wiley, 2nd edition, 1992.

[47] G.J. Klir and T.A. Folger. *Fuzzy sets, uncertainty, and information.* Prentice Hall, Eaglewood Cliffs, N.J., 1992.

[48] K.M. Passino and S. Yurkovich. *Fuzzy control.* Addison-Wesley, Menlo Park, California, 1998.

[49] Z. Wasik and A. Saffiotti. A fuzzy behavior-based control system for manipulation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1596–1601, Lausanne, CH, 2002.

[50] Andrea Bonarini, G. Invernizzi, T.H. Labella, and M. Matteucci. An architecture to coordinate fuzzy behaviors to control an autonomous robot. *Fuzzy sets and systems*, 134, no.1:101–115, 2003.

[51] A. Bonarini and M. Restelli. An architecture to implement agents cooperating in dynamic environments. In *Proceedings of AAMAS 2002 - Autonomous Agents and Multi-Agent Systems*, pages 1143–1144, New York, NY, 2002.

[52] E. Mendelson, O. Nayer, S. Berman, and Y. Edan. Behavior-based control of multi-robot assembly/palletizing systems. In *Proceedings of the 5th Biannual World Automation Congress, 2002.*, pages 1–6, 2002.

[53] M.F. Selekwa and E.G. Collins. Centralized fuzzy behavior control for robot navigation. In *2003 IEEE International Symposium on Intelligent Control.*, pages 602–607, 2003.

[54] D. Driankov and A. Saffiotti. *Fuzzy logic techniques for autonomous vehicle navigation.* Physica-Verlag, Heidelberg, 2001.

[55] António Abreu and Luís Correia. Behavior based decision control in autonomous vehicles:a fuzzy approach using khepera. In *Fuzzy Systems, 2000. FUZZ IEEE 2000. The Ninth IEEE International Conference*, volume 1, pages 140–145, 2000.

[56] Kiyotaka Izumi, Keigo Watanabe, and Sang-Ho Jin. Obstacle avoidance of mobile robot using fuzzy behaviour-based control with module learning. In *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference*, volume 1, pages 454–459, 1999.

[57] C.C. Wong, M.F. Chou, C.P. Hwang, C.H. Tsai, and S.R. Shyu. A method for obstacle avoidance and shooting action of the robot soccer. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 4, pages 3778–3782, 2001.

[58] M. J. Jung, H. S. Kim, H. S. Shim, and J. H. Kim. Fuzzy rule extraction for shooting action controller of soccer robot. In *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE '99. 1999 IEEE International*, volume 1, pages 556–561, 1999.

[59] François Michaud. Selecting behaviors using fuzzy logic. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 585–592, Barcelona, SP, 1997.

[60] P. Pirjanian and M. Mataric. A decision theoretic approach to fuzzy behavior coordination. In *IEEE International Symposium on Computational Intelligence in Robotics & Automation (CIRA99)*, Monteray, CA, Nov. 1999.

[61] 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.

[62] Jane Yung jen Hsu, Der-Chiang Lo, and Shih-Chun Hsu. Fuzzy control for behavior-based mobile robots. In *Proceedings of the NAFIPS/IFIS/NASA'94*, San Antonio, USA, December 1994.

[63] W. Li and X. Feng. Behavior fusion for robot navigation in uncertain environments using fuzzy logic. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 1790–1796, 1994.

[64] Takanori Shibata and Toshio Fukuda. Coordinative behavior by genetic algorithm and fuzzy in evolutionary multi-agent system. In *Proceedings of the*

1993 IEEE International Conference on Robotics and Automation, volume 1, pages 760–765, 1993.

[65] Edward Tunstel, Marco A.A. de Oliveira, and Sigal Berman. Fuzzy behaviour hierarchies for multi-robot control. *International Journal of Intelligent Systems*, 17:449–470, 2002.

[66] Hakan Duman and Huosheng Hu. Fuzzy logic for behaviour co-ordination and multi-agent formation in robocup. In *International Conference on Recent Advances in Soft Computing 2000*, pages 3758–3763, De Montfort University, Leicester, England, 2000.

[67] E. Dehghan and G. Mitchell. Cooperative multi-agent robot soccer team, 1999.

[68] F. Hoffmann Oscar Cordon, Francisco Herrera. *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. World Scientific, Singapore, 2001.

[69] Plamen P Angelov. *Evolving Rule-Based Models: A Tool for Design of Flexible Adaptive Systems*, volume 92 of *Studies in Fuzziness and Soft Computing*. Springer-Verlag, Wurzburg, 2002.

[70] Kenneth De Jong. Learning with genetic algorithms: An overview. *Machine Learning*, 3:121–138, 1988.

[71] C. L. Karr, L. M. Freeman, and D. L. Meredith. Improved fuzzy process control of spacecraft autonomous rendezvous using a genetic algorithm. *Intelligent Control and Adaptive Systems*, 1196:274–288, 1989. G. Rodriguez (ed), The International Society of Photo-Optics Instrumentation Engineers (SPIE), Philadelphia.

[72] C. L. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6:26–31, 1991.

[73] C. L. Karr and E. J. Gentry. Fuzzy control of ph using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 1:46 –53, Feb. 1993.

[74] C. L. Karr. Fuzzy-evolutionary systems. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages D2.1:1–2:9. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[75] Oscar Cordón, Francisco Herrera, and Manuel Lozano. A three-stage method for designing genetic fuzzy systems by learning from examples. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 720–729, Berlin, 1996. Springer.

[76] Hakan Berat Gürocak. A genetic-algorithm-based method for tuning fuzzy logic controllers. *Fuzzy Set and Systems*, 108(1):39–47, 1999.

[77] A. Satyadas and K. Krishnakumar. EFM-based controllers for space station attitude control: Application and analysis. In J.L. Verdegay F. Herrera, editor, *Genetic Algorithms and Soft Computing*, pages 152 – 171. Physica-Verlag, Wurzburg, 1996.

[78] Ben Burdsall and Christophe Giraud-Carrier. Evolving fuzzy prototypes for efficient data clustering. In *Proceedings of the Second International ICSC Symposium on Fuzzy Logic and Applications (ISFL'97)*, pages 217–223. ICSC Academic Press, February 1997.

[79] Philip Thrift. Fuzzy logic synthesis with genetic algorithms. In Richard K. Belew and Lashon B. Booker, editors, *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, pages 509–513, San Mateo, CA, 1991. Morgan Kaufmann.

[80] Stephen F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, Pittsburgh, 1980.

[81] Frank Hoffmann and Gerd Pfister. Evolutionary design of a fuzzy knowledge base for a mobile robot. *International Journal of Approximate Reasoning*, 17(4):447–469, 1997.

[82] John H. Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern–Directed Inference Systems*, pages 313–329. Academic Press, New York, 1978.

[83] Andrea Bonarini. Evolutionary Learning of Fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Norwell, MA: Kluwer Academic Press, 1996. ftp://ftp.elet.polimi.it/pub/Andrea.Bonarini/ELF/ELF-Pedrycz.ps.gz.

[84] T. Nakashima H. Ishibuchi and T. Murata. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 29(5):601–618, 1999.

[85] Gilles Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *LNAI*, pages 280–296, Vienna, Austria, April 1993. Springer Verlag.

[86] O. Cordón, M. J. del Jesús, F. Herrera, and M. Lozano. MOGUL: a methodology to obtain genetic fuzzy rule-based systems under the iterative rule learning approach. *International Journal Intelligent Systems*, 14(11):1123 – 1153, 1999.

[87] A. González and R. Pérez. SLAVE: A genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, 7(2):176 –191, April 1999.

[88] A. Homaifar and E. McCormick. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3:129–139, May 1995.

[89] Ferdinando Cicalese, Antonio di Nola, and Loia Loia. A Fuzzy Evolutionary Framework for Adaptive Agents. In *Proceedings of the 1999 ACM symposium on Applied computing (SAC'99)*, pages 233–237, 1999.

[90] Yasuhiko Dote. Neuro-fuzzy control. In Ajith Abraham, Javier Ruiz-del-Solar, and Mario Köppen, editors, *Soft Computing Systems - Design, Management and Applications*, Frontiers in Artificial Intelligence and Applications Vol. 87, pages 9–10. IOS Press Amsterdam, Berlin, Oxford, Tokyo, Washington D.C., 2002.

[91] Enzo Mumolo, Massimiliano Nolich, and Scalamera Scalamera. Genetic-fuzzy optimization algorithm for adaptive learning of human vocalization in robotics. In Franz Rothlauf, Juergen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors, *Applications of Evolutionary Computing, EvoWorkshops2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC*, volume 3449 of *LNCS*, pages 368–377, Lausanne, Switzerland, 30 March-1 April 2005. Springer Verlag.

[92] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[93] Michio Sugeno, editor. *Industrial Applications of Fuzzy Control*, Amsterdam, 1985. Elsevier Science Publishers.

[94] Cyberbotics. *Webots 2.0 User Guide*, September 13 1999.

[95] Cyberbotics. *Webots 2.0 Reference Manual*, September 13 1999.

[96] K-team. *Khepera User Manual*, 1999.

[97] G. Ulivi. On fuzzy logic control of mobile robots. In C. Bonivento, Fantuzzi C., and Rovatti, editors, *Fuzzy Logic Control - Advances in Methodology*, pages 255 – 276. World Scientific Publishing Co. Pte. Ltd., Singapore, 1998.

[98] A. Saffiotti. The uses of fuzzy logic for autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[99] A. Saffiotti, E. H. Ruspini, and K. Konolige. Using fuzzy logic for mobile robot control. In H. Prade D. Dubois and H. J. Zimmermann, editors, *International Handbook of Fuzzy Sets and Possibility Theory*, volume 5. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands, 1997. Forthcoming.

[100] R. Kruse, J. Gebhardt, and F. Klawonn. *Foundations of Fuzzy Systems*. Wiley, 1994.

[101] Hans Bandemer and Siegfried Gottwald. *Fuzzy Sets, Fuzzy Logic Fuzzy Methods with Applications*. John Wiley & Sons, New York, NY, 1995.

[102] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), 1998.

[103] Chin-Teng Lin and C. S. George Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Transactions on Computers - Special Issue on Artificial Neural Networks*, 40(12):1320–1336, December 1991.

[104] Ji-Cheng Duan and Fu-Lai Chung. Cascading fuzzy neural networks. In *1999 IEEE International Fuzzy Systems Conference Proceedings*, volume 1, pages 55–60, 1999.

[105] M. Wagenkecht. Cascading of rule-based systems — an algorithmic approach. 17th Conf. on EURO XVII, 2000.

[106] Parker B.S. *Demonstration of Using Genetic Algorithm Learning*. Information Systems Teaching Laboratory, 1992.

[107] C. R. Stephens and H. Waelbroeck. Effective degrees of freedom in genetic algorithms and the block hypothesis. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 34–40, San Francisco, July 19–23 1997. Morgan Kaufmann.

[108] A. E. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5:78–88, 1992.

[109] D. Park, A. Kandel, and G. Langholz. Genetic-based new fuzzy reasoning models with application to fuzzy control. *IEEE Trans. System Man Cybernet*, 24:39–47, Jan./Feb. 1994.

[110] C.C. Wong and S.M. Feng. Switching type fuzzy controller design by genetic algorithm. *Fuzzy Sets Syst.*, 74, No.2:175–185, 1995.

[111] A. Homaifar and E. McCormick. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Trans. Fuzzy Syst.*, 3:129–139, May 1995.

[112] H. Surmann. Genetic optimization of a fuzzy system for charging batteries. *IEEE Trans. Ind. Electron.*, 43:541–548, Oct. 1996.

[113] C.C. Lee. Fuzzy logic control systems: fuzzy logic controller–part 1. *IEEE Trans. System Man Cybernet*, SMC-20:404–418, 1990.

[114] RobotCup. Robot World Cup Institative, 1993. http://www.robocup.org.

[115] FIRA. Federation of International Robot-soccer Association, 1995. http://www.fira.net.

[116] Q.C. Meng, X.D. Zhuang, C.J. Zhou, J.S. Xiong, Y.L. Wang, T. Wang, and B. Yin. Game strategy based on fuzzy logic for soccer robots. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 3758–3763, 2000.

[117] Prahlad Vadakkepat, Ooi Chia Miin, Xiao Peng, and Tong Heng Lee. Fuzzy behaviour based decision control of mobile robots. *IEEE Trans. on Fuzzy Systems*, 12(4):559–565, Aug 2004.

[118] Wei-Po Lee, John Hallam, and Henrik Hautop Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of IEEE 4th International Conference on Evolutionary Computation*, volume 1. IEEE Press, 1997. to appear.

[119] Donald Dewar Leitch. *A New Genetic Algorithm for the Evolution of Fuzzy Systems*. PhD thesis, University of Oxford, 1995.

[120] Detlef Nauck and Rudolf Kruse. A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation. In *Proc. IEEE Int. Conf. on Neural Networks 1993*, pages 1022–1027, San Francisco, 1993.

[121] J. Gómez Marín-Blázquez, Q. Shen, and A. Tuson. Tuning fuzzy membership functions with neighbourhood search techniques: A comparative study. In *Proceedings of the 3rd IEEE International Conference on Intelligent Engineering Systems*, pages 337–342, November 1999.

[122] A. F. GSmez Skarmeta and F. Jimenez. Generating and tuning fuzzy rules using hybrid systems. In *Proceedings of the 6th International Conference on Fuzzy Systems*, volume 1, pages 247–252, 1997.

[123] B. Kosko. *Neural Networks and Fuzzy Systems.* Prentice-Hall, 1992.

[124] J. E. Baker. Reducing bias and inefficience in the selection algorithm. In *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*, pages 14–21, 1987.

[125] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. David Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, George Mason University, June 1989. Morgan Kaufmann.

[126] SRG. Singapore robotic games, 2001. http://guppy.mpe.nus.edu.sg/srg/.

[127] Randy L.Haupt and Sue Ellen Haupt. *Practical Genetic Algorithm.* Wiley-Interscience, 1997.

[128] Robin Biesbroek. The genetic algorithm tutorial webpage. http://www.estec.esa.nl/outreach/gatutor/Default.htm.

[129] Lakhmi C. Jain. *Evolution of Engineering and Information Systems and Their Applications.* CRC Press International, Sept. 1999.

[130] Neil A. Campbell. *Biology.* The Benjamin Cummings Publishing Company, 1996.

[131] Takeshi Furuhashi. Development of *IF-THEN* rules with the use of DNA coding. In Witold Pedrycz, editor, *Fuzzy Evolutionary Computation*. Kluwer Academic Publishers, Boston, 1997.

[132] Tomohiro Yoshikawa and Yoshiki Uchikawa. Effect of new mechanism of development from artificial DNA and discovery of fuzzy control rules. In *Proc. of IIZUKA'96*, pages 498–501, 1996.

[133] Ki-Youl Lee, Dong-Wook Lee, and Kwee-Bo Sim. Evolutionary neural networks for time series prediction based on L-system and DNA coding method. In *IEEE. Proc. of the 2000 Congress on Evolutionary Computation*, volume 1.2, pages 1467–1474, 2000.

[134] Tomohiro Yoshikawa, Takeshi Furuhashi, and Yoshiki Uchikawa. A combination of DNA coding method with pseudo-bacterial ga for acquisition of fuzzy control rules. In *Proceedings of the First Online Workshop on Soft Computing (WSC1)*, pages 107–112, 1996.

[135] Lihong Ren, Yongsheng Ding, and Shihuang Shao. DNA genetic algorithms for design of fuzzy systems. In *The Ninth IEEE International Conference on Fuzzy Systems*, volume 2, pages 1005–1008, May 7-10 2000.

[136] Lihong Ren and Yongsheng Ding. Parameter optimization for a class of general ts fuzzy controllers via a new DNA-based genetic algorithm intelligent control and automation. In *Fifth World Congress on WCICA*, volume 3, pages 2149–2153, June 15-19 2004.

[137] J. David Schaffer, Richard A. Caruana, Eshelman Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. David Schaffer, editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 51–60, George Mason University, June 1989. Morgan Kaufmann.

[138] Lawrence Davis. A genetic algorithms tutorial. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, pages 1–101. Van Nostrand Reinhold, New York, 1991.

[139] Rick Belew and Lashon Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.

[140] Stephanie Forrest, editor. *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, San Mateo, CA, 1993. Morgan Kaufmann.

[141] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996. Contains introductory chapter on LCS.

[142] Hillol Kargupta, Kalyanmoy Deb, and David E. Goldberg. Ordering genetic algorithms and deception. Technical Report IlliGAL Report No 92006, University of Illinois, Urbana, 1992.

[143] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In Aravind Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 162–164, Los Angeles, CA, August 1985. Morgan Kaufmann.

[144] David E. Goldberg and Robert Lingle. Alleles, loci, and the traveling saleman problem. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications (ICGA'85)*, pages 154–159, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[145] Gilbert Syswerda. Schedule optimization using genetic algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, pages 332–349, New York, 1991. Van Nostrand Reinhold.

[146] H. Mühlenbein. Evolution in Time and Space – The Parallel Genetic Algorithm. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337, San Mateo, CA, USA, 1991. Morgan Kaufmann Publishers.

[147] Thomas Bäck and Martin Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Evolutionary Programming IV: Proc. of the Fourth*

Annual Conf. on Evolutionary Computation, pages 33–51, Cambridge, MA, 1995. MIT Press.

[148] James R. Levenick. Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 123–127, San Mateo, CA, 1991. Morgan Kaufman.

[149] Annie S. Wu and Robert K. Lindsay. Empirical studies of the genetic algorithm with noncoding segments. *Evolutionary Computation*, 3(2):121–147, 1995.

[150] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications (ICGA'85)*, pages 183–187, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.

[151] N.J.Dibb. Why do genes have introns? *Federation of European Biochemical Societies*, 325(1,2):135–139, year.

[152] John H.Rogers. The role of introns in evolution. *Federation of European Biochemical Societies*, 268(2):339–343, 1990.

[153] Laurent Duret. Why do genes have introns? recombination might add a new piece to the puzzle. *Trends In Genetics*, 17(4), April 2001.

[154] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, CA, 1993.

[155] Colin R. Reeves and Christine C. Wright. Epistasis in genetic algorithms: An experimental design perspective. In *ICGA*, pages 217–224, 1995.

[156] Ralf Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39:263–278, 1996.

[157] David Beasley, David R. Bull, and Ralph R. Martin. Reducing epistasis in combinatorial problems by expansive coding. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 400–407, San Mateo, CA, 1993. Morgan Kaufmann.

[158] Yuval Davidor. Epistasis variance: A viewpoint on GA-hardness. In Gregory J. Rawlins, editor, *Foundations of genetic algorithms*, pages 23–35. Morgan Kaufmann, San Mateo, CA, 1991.

[159] Peter J. Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 75–98. MIT Press, Cambridge, MA, 1994.

Appendix A

Author's Publications

Publications related to the thesis

1. P. Vadakkepat, C. M. Ooi, P. Xiao and T. H. Lee, Fuzzy behavior based decision control of mobile robots, *IEEE Trans. on Fuzzy Systems*, Vol.12(4): 559–565, Aug 2004.

2. P. Xiao, P. Vadakkepat, and T. H. Lee, Context dependant DNA coding with redundancy and introns, (2nd review) *IEEE Trans. on Systems, Man and Cybernetics Part B*, 2005.

3. P. Vadakkepat, P. Xiao, B. K. Quek and T. H. Lee, Evolution of fuzzy behaviors for multi-robotic system, (2nd review) *Int. J. of Robotics and Autonomous System*, 2005.

4. P. Vadakkepat, P. Xiao, C. L. Kwan and T. H. Lee, Adaptive fuzzy behavior based control of multiple mobile robotic system, submitted to *Int. J. of Intelligent Systems*, 2005.

5. P. Xiao, P. Vadakkepat and T. H. Lee, DNA coded GA for the rule base optimization of a fuzzy logic controller, in: *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, IEEE Press, COEX, Seoul, Korea, 2001, Pages 1191–1196.

6. P. K. Kim, P. Vadakkepat, T. H. Lee and P. Xiao, Evolution of control systems for mobile robots, in: D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, M. Shackleton (Eds.), *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, 2002, Pages 617–622.

7. P. Xiao, P. Vadakkepat and T. H. Lee, Mobile robot obstacle avoidance: DNA coded GA for FLC optimization, in: *Proceedings of the Congress on FIRA Robot World Cup 2002*, Seoul, Korea, 2002, Pages 553–558.

8. J. S. Chaal, P. Vadakkepat, T. H. Lee and P. Xiao, Fuzzy sensor fusion for reactive navigation of mobile robots, in: *Proceeding of the 2nd International Symposium on Autonomous Minirobots*, Feb. 18-23, 2003.

9. P. Vadakkepat, L. Xin, P. Xiao, A. R. Vasudev and T. H. Lee, Behavior based and evolutionary techniques in robotics: some instances, in: K. Murasse, T. Asakura (Eds.), *Dynamic Systems Approach for Embodiment and Sociality: From Ecological Psychology to Robotics*, Vol. 6 of International Series on Advanced Intelligence, Advanced Knowledge International Pty Ltd, Adelaide, 2003, Pages 137–150.

10. P. Xiao, P. Vadakkepat and T. H. Lee, Mobile robot obstacle avoidance: DNA coded GA for FLC optimization, In *New Optimization Techniques in Engineering*, edited by B V Babu, pages 503–513. Springer-Verlag. 2003.

11. P. Vadakkepat, P. Xiao, C. S. Khor and T. H. Lee, DNA coding in evolutionary computation, *Proceedings of IEEE Conference on Cybernetics and Intelligent Systems 2004*, Singapore, 1-3 December 2004.

Other publications

- C. K. Wai, P. Vadakkepat and P. Xiao, The hierarchical robot control structure and the newton's divided difference approach to robot path planning, *Journal of Harbin Institute of Technology*, 2001, Pages 303–308.

- X. Liu, P. Vadakkepat, T. H. Lee, P. Xiao and P. K. Kim, Comparison of robot navigation by evolutionary neural networks and pain-based algorithm, in: D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, M. Shackleton (Eds.), *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, IEEE Press, 2002, Pages 1994–1999.

- P. Vadakkepat, Y. T. Quek, P. Xiao and T. H. Lee, "Step-by-step" evolutionary learning process Of fuzzy rules for mobile robots, In *Intelligent Robots: Vision, Learning and Interaction*, edited by Hyungsuck Cho. KAIST Press, Korea, 2003.