University of Alberta

Fuzzy Anticipatory Learning Classifier System for Mobile Robot Navigation

by ©

Pawel Maksymilian Pytlak

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Electrical & Computer Engineering

Edmonton, Alberta
Fall 2007

Your file Votre référence
ISBN: 978-0-494-33329-7
Our file Notre référence
ISBN: 978-0-494-33329-7

One cannot escape the feeling that these mathematical formulae have an independent existence and an intelligence of their own, that they are wiser than we are, wiser even than their discoverers, that we get more out of them than we originally put into them.

– Heinrich Hertz

This thesis is dedicated to my parents.

Abstract

Designing autonomous intelligent control systems for real-world problems is a daunting task. The complex input-output relationships resulting from the interaction between a process and its environment are often not readily solvable by traditional mathematical methods. A growing amount of research is being performed in designing control systems which develop their own solution by utilizing methods borrowed from nature. This thesis presents work performed in the aforementioned field, specifically in developing an extension to the Anticipatory Learning Classifier System (ALCS) to facilitate the transparent use of real-valued inputs as well as outputs in order to make the system more applicable to real-world problems. This has been accomplished through the application of concepts borrowed from Fuzzy Logic to implement a variation of an evolvable Fuzzy Controller within the ALCS paradigm. As such, the Fuzzy Anticipatory Learning Classifier System (or FALCS) allows the user to evolve an adaptive control system capable of latent learning as well as utilizing the best known course of action in the absence of previous knowledge. The FALCS-based controller was tested to be successful in generating a rule-base that kept a simulated agent "alive" in a virtual environment. Furthermore, a FALCS-based controller was successfully implemented to allow a simulated robot to navigate a previously unknown environment, as well as seeking a goal location while avoiding obstacles at the same time.

Acknowledgements

Firstly, I would like to thank my supervisors: Drs. Petr Musílek and Marek Reformat for granting me the incredible opportunity to conduct this exciting research, for their support and guidance. Secondly, I would also like to thank all the FACIA members, both support staff and peers for their input as well as their friendship. Finally, I would like to thank my parents for their support and encouragement in this endeavor.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Today's modern society is more and more dependent on technology, and as a result, we are pushing the engineering frontiers faster than ever before. Nevertheless, the devices which surround us everyday are still extremely dependent on humans to give them instructions on how they should function. This is especially true of control systems which operate an extremely wide range of today's products: from consumer electronics, to business and factory equipment, to aerospace vehicles and extraterrestrial probes. However, the design and implementation of these control systems is a complex task, which requires immense expertise and a great deal of patience due to the large and complex spaces that the inputs and outputs encompass.

To alleviate the burden of this problem, designers are more frequently turning to artificial intelligence and statistical methods to help simplify the designs and allow for the devices to learn their own control functions. Through the use of intelligent systems, the devices and machines are capable of making human-like decisions on their own without having a human designer provide solution for every problem that could be encountered.

In the field of intelligent systems, a great deal of promise has been shown in the area of evolutionary computation [1] [2] [3]. As the name suggests, this paradigm attempts to "evolve" a solution to a given problem over successive generations by using methods observed in nature such as Darwinian natural selection [4], immune systems [5] [6], swarm intelligence [7] [8], self organizations and so forth. In effect, these methods could be seen as performing a guided stochastic search [9] when viewed from a classical artificial intelligence perspective. Furthermore, many systems employ different combinations of these techniques in order to benefit from each method's own strengths while at the same time minimizing their own independent weaknesses.

One of the many benefits of applying evolutionary computation strategies to solve optimization problems is the inherent massive parallelism that exists inside the many stages of evolutionary computing algorithms. This is of fair significance in today's world, due to the multi-core processing push by the industry, as these methods can be readily coded to make fuller use of all the parallel computational power available, thus providing results much faster as opposed to competing sequential algorithms.

One of the most amazing aspects of using evolutionary computation methods is the fact that they allow one to generate desirable intelligent systems with minimal human intervention. This results in reducing the costs of bringing new devices to the market as well as developing more robust devices which are capable of better applying themselves to their working environment as opposed to a "one-size-fits-all" strategy [10] [11] [12]. In particular, evolutionary computing methods can be applied to systems which must operate in environments that cannot be modelled in the laboratory either due to impracticality, limited knowledge or due to completely new or frequency changing operating environments which are not know at the time of development. In such situations, evolutionary computing can be applied online, where the system receives feedback on its current performance from a governor to evolve a more optimized system for future use.

A prime example of such an application exists in the realm of mobile robotics. This is a fast growing sector in today's world with applications ranging from the benign such as toy pets, to the useful such as robotic lawnmowers and vacuum cleaners, and to the exotic in the form of exploratory robotics being sent into space. Furthermore, advanced robotic applications are being employed to more pressing matters in our current society such as search and rescue, counter-terrorism and military applications, all of which aim to either save lives or to reduce the risk of human death as opposed to that of the machine.

The objective of this research is to develop a novel autonomous intelligent system capable of learning goal oriented behavior for the control of a robotic mobile platform, where a robot would glean situations from its environment and learn the correct input output mapping with limited or no human intervention. Moreover, the goal is to design a transparent control system whose actions can be analyzed by a human expert and which is capable of continuous online learning.

To accomplish this objective, the work has aimed on expanding an intriguing evolutionary control technique based on the Anticipatory Learning Classifier System (ALCS). The Anticipatory Learning Classifier System forms a rule-based intelligent system which evolves a population of rules to solve the control problem at hand by exploiting psychologically significant observations of anticipatory behavior from the real world. As such, the research described in this thesis focuses on an extension of the Anticipatory Learning Classifier system originally developed by Stolzmann and Butz [13] that would deal directly with real-valued data and be well suited for integration into control systems, in particular, those found in mobile robotic applications.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

- A novel system, based upon the ALCS along with concepts of fuzzy logic, was designed such that it is capable of transparently handling real-valued inputs and providing real-valued outputs.

- ALCS learning algorithms were modified to allow for machine learning on fuzzy sets as opposed to discrete symbol strings.

- Enhanced classifier selection and action generation methods were devised that exploit the advantages of fuzzy logic.

- The developed system was successfully applied as an intelligent, autonomous controller for two separate application domains.

1.3 Organization

This thesis is organized into 5 chapters. Chapter 2 presents a brief description of the fundamental workings of components which comprise the Fuzzy Anticipatory Learning Classifier System, while Chapter 3 provides a detailed view of how the system is designed. Chapter 4 reviews experimental results obtained by testing the system in two settings: a simulated agent and a mobile robot. Finally, Chapter 5 concludes the thesis, summarizes main contributions, and provides suggestions for extending the present system in the future.

Chapter 2

Background

This chapter provides an overview of the different constituents which are incorporated and/or which have provided inspiration for the development of the Fuzzy Anticipatory Learning Classifier System.

2.1 Intelligent Systems

An intelligent system [14] is an instantiation of an algorithm or set of algorithms capable of performing actions or giving outputs which would be deemed intelligent should they be performed by a human. This means, that for a system to be an "intelligent system," it must be capable of performing several, ideally all of the following key tasks. It must be capable of gleaning information from its surroundings or assigning labels to inputs as well as be capable of analyzing the gathered information so that it can recognize patterns in the data. It must be able to perform inference from incomplete information, as well as extract meaning from inexact and noisy data. Furthermore, it must be capable of dealing with unfamiliar situations and adapting to them. Finally, it must be capable of providing an informed output based on the collected knowledge.

Many different architectures are available for devising an intelligent system. The system described in this thesis focuses on utilizing intelligent systems based upon soft computing principles [15]. These are computational methods based on approximate and/or qualitative representations of knowledge in a manner similar to what is believed to be used by human reasoning.

As such, they are well suited for dealing with incomplete as well as imprecise information.

The central focus of this thesis is to expand the Anticipatory Learning Classifier System to utilize real values instead of discrete symbols. This will make the system more suitable for direct incorporation into intelligent control systems.

2.2 Reinforcement Learning

Reinforcement learning in an artificial intelligence (AI) learning technique which was pioneered by Sutton and Barto [16]. It lies in an area between *supervised learning*, where the system receives direct instructions as to what the correct input-output response pairs should be in order to mimic them, and *unsupervised learning*, where the algorithm must discover innate features and patterns from raw data without receiving any guidance (e.g. clustering techniques or self-organization). In effect, reinforcement learning attempts to devise a solution from limited feedback from the environment in the form of rewards based on how it performs when going from one state to another. Thus, the goal of a reinforcement learning algorithm is to find an optimal solution to a sequential problem in the form of a policy π, which specifies what action to take for each given state in the problem so as to maximize the total reward received from all possible steps.

An offshoot from the original reinforcement learning algorithm is the group of Temporal Difference learning algorithms that aim to find the optimal policy π by means of sampling the environment and learning from sequential steps. This is ideal for situations where the entire range of possible states and actions is not known such as in the case of a robot controller.

The reinforcement learning process present in the Anticipatory Learning Classifier System draws its roots from a reinforcement learning algorithm developed by Watkins termed Q-learning [17]. The basis for this methodology is that through successive iterations, using the update rule shown in Formula 2.1, an approximation of the Q-function may be learnt (if the environment is static and the learning rate α lies in [0..1] and takes on successively decreasing values such that $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < inf$, an optimal Q value may be learnt; see [18] for specifics). As a result of learning the Q-value, a policy is generated which allows the agent to find the method of reaching a goal by selecting actions which maximize the estimated state-action value.

$$Q'(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)\left[r_t + \gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \quad (2.1)$$

where:

Q is the expected utility value

Q' is the newly updated expected utility value

s_t is a given state at time t

a_t is an action performed at time t

r_t is a reward received at time t

α is the learning rate in [0..1]

γ is the discount factor in [0..1)

2.3 Evolutionary Computing

The developed Fuzzy Anticipatory Learning Classifier System (or FALCS) falls into the broad field of evolutionary computation. Evolutionary Computing (or EC) [2] is a paradigm inspired by concepts of Darwinian natural selection to evolve a system which is capable of reaching a particular objective by means of processing successive generations of candidate solutions. During each learning phase, a set of candidate solutions is formed. Through various operators, new solutions are generated which come either as a result of the current population or by other means from the problem space. These candidate solutions are then evaluated to determine how well they solve a particular task. Undesirable solutions are removed from the population and surviving candidates live on to form the next population.

Many different algorithms have been developed which fall into this category and even more utilize partial concepts. This thesis will focus on two particular instances: the genetic algorithm and its close relative the learning classifier system, the main foundation for the Fuzzy Anticipatory Learning Classifier System.

2.3.1 Genetic Algorithms

Genetic Algorithms (GA) [19] [20] were developed by Holland [21] as a means to solve optimization problems. They accomplish the task through successive populations of candidate solutions. During each iteration of the algorithm, GAs apply genetic operators modeled from nature. The resulting offspring populations are successively selected, recombined and altered to form an iteratively better solution to the problem.

Genetic algorithms operate upon sequence of bits known as chromosomes, where each bit, commonly referred to as a gene, encodes a particular feature in the problem space. Initially, a random population of chromosomes is generated after which the genetic operators are applied to form successive candidate populations.

The two most commonly applied genetic operators utilized in Genetic Algorithms are the crossover and the mutation operator. The crossover operation splices genes between two randomly selected chromosomes to form two new chromosomes. Depending on the particular implementation, the parent chromosomes die off (i.e. they are removed from the population), or remain which is the case for the GA used in the implementation of the Anticipatory Learning Classifier System.

The mutation operator acts upon randomly selected chromosome and changes a specified number of genes to random values. It is through this operation that the GA performs exploratory actions to search for completely new solutions.

Finally, the new population is passed through an evaluation stage which then computes each chromosome against a fitness function to determine its quality in solving the optimization problem. The outcome of this process determines the probability that the individual will be selected in future iterations for the crossover population. The iterations terminate once a desired quality of solution is achieved or no further improvement is attained i.e. the solution converges.

Genetic Algorithms can be seen as forming an intelligent hill-climbing algorithm [22] that searches a known problem space to find an optimal solution through the recombination of previous candidates while at the same time attempting to avoid local minima through mutation. Under the correct circumstances, genetic algorithms may find a near-optimal solution to the problem [23].

By setting parameters of a GA, one can achieve a desirable ratio between exploration and exploitation.

2.3.2 Learning Classifier Systems

Learning Classifier Systems (LCS) are a family of evolutionary computing algorithms which develop a rule-based solution to a particular input-output problem. These algorithms draw their roots from the original implementation by Holland [24], whereby a solution is formed through successive trial and error of a set of rules which are augmented by parameters to form so called classifiers, that match their input condition against the current state of the operating environment and impart a stored action onto it. The condition and action strings inside the classifiers themselves take on values of either *true*, *false* or *don'tcare.*

When an input condition or message is presented to the system, it is recorded in an internal memory space which can be seen as a blackboard message list. From the population of classifiers, $[N]$, a set of matching rules is formed by comparing each classifier's condition string against the environmental input. In order for a classifier to be selected for insertion into the set of matching classifiers, referred to as the Match Set $[M]$, all of the condition symbols in the classifier's condition string must be the same as those from the input message, except for don'tcare symbols which ignore the environmental input for that particular bit's position.

The winning classifier is selected from the Match Set through a bidding process that utilizes the quality of match in terms of its specificity and the adaptive strength parameter associated with each rule as a bid amount. The classifier present in the Match Set with the highest bid is then placed into the internal message board list. This process of selecting classifiers against the environmental input is repeated until all the available slots in the message list are filled up. The final action is chosen from the selected classifiers via a roulette wheel selection process, whereby the probability of a classifier being selected is proportional to that classifier's bid. The winning classifier's action is then imparted onto the environment after which the success of the action is gauged.

The success or failure of a particular action determines the credit to be assigned to the matching classifiers through a bucket brigade credit apportionment algorithm [25]. The bid made by each classifier in the message list is placed into to a "bucket". The winners of the previous iteration receive an equal proportion of the current contents of the bucket. Furthermore, the reward from the environment is equally distributed to all the previously selected classifiers.

Finally, a Genetic Algorithm is utilized on the entire population to create new rules based on recombination of the current classifiers. The selection of classifiers to take part in the genetic operations is based in proportion to the respective classifier's fitness. Thus the GA attempts to evolve a cooperative set of classifiers rather than having a single fittest classifier for each situation [26].

A close relative of Holland's original LCS system is Wilson's [27] Zeroth-level Classifier System (ZCS) which simplifies the original LCS framework through the elimination of the complex message board system and rule bidding inside the Learning Classifier System. Furthermore, the notion of an Action Set, $[A]$, was introduced which treats all matching classifiers with the same action strings as one for both selection and reinforcement learning, rather than dealing with individual classifiers by themselves. The perceived reward at the next iteration is then distributed to the previous Action Set equally to all classifiers less a predefined discount amount. Also, all classifiers not present in the Action Set are given a penalty to reduce the chance that they will be selected in future iterations.

2.3.3 XCS

One of the disadvantages of the classical Learning Classifier algorithms is that they tend to focus on niche environments. This is a direct result of the reward system that favors actions which have the highest utility value thus reducing the system's motivation to explore other alternatives in a given environment. In order to adapt the classifier system to model an environment more accurately, Wilson [28] devised a strategy where the strength aspect of the classifier is replaced with a predictive system which attempts to maximize the accuracy of predicting the reward payoff that will be attained when its action is imparted onto the environment. The resulting system, called XCS [29], [30], no longer focuses on niche areas but rather performs a wider exploration of the environment, as it aims to maximize the prediction accuracy of reward received when performing an action rather than maximizing the reward itself. As such, it creates a state-action model of the explored environment in the form of a set of *if-then* rules that are augmented by a parameter signifying the expected payoff of each rule.

2.3.4 ALCS

A close descendant of the XCS is the Anticipatory Learning Classifier System (ALCS) algorithm developed by Stolzmann and Butz [31], [13], [32]. The ALCS follows the desire to develop a model of the working environment as in XCS, but takes that concept to the next level. Incorporating observations from the psychological work of anticipatory behavior in humans and animals [33], it aims at modeling the environment in terms of input conditions, an action and the effect that the action has on the environment.

In the most basic form, the Anticipatory Learning Classifier System works in a manner similar to a combination of both LCS and XCS. The system focuses on a population of classifiers which hold *Condition-Action-Effect* rules augmented by a few key parameters. Thus, each classifiers rule specifies the *Condition* upon which the classifier is active ("fires"), the *Action* which the classifier will impart onto the environment, and the *Effect* that its action will have on the environment, (i.e. the next state of the environment). The condition bitstring can take on discrete symbols i.e. "1", "0" or "#" (a don'tcare value which matches all possible symbols). The effect bit string is formed in the same way with the exception that the don'tcare symbol represents a "no change" in the environment rather than simply ignoring the value during the learning stages.

Each classifier contains several key parameters which are modified during the learning process. The two most important are the reward prediction value, r, and quality parameter, q. The reward prediction parameter is similar to the Q utility value in Watkins's Q-Learning algorithm [17], which is an indicator of the expected return of taking the current action and performing the best known actions thereafter. In ALCS it can be seen as a means of predicting the benefit that will be attained when the rule is utilized. The quality parameter, q, of the classifier rule represents the predictive quality on a scale of [0..1] indicating how well the condition-action-effect triplet matches the given environmental message. Finally, the third important parameter is the Markset, M, which stores all the input conditions where the given classifier fails to anticipate the output correctly.

The operation of the algorithm is fairly straight forward. The working environment is probed for a representation of the current state. This state is used to generate a set of matching classifiers known as the Match Set $[M]$ based upon each classifier's condition bitstring. An exploratory action is selected based on the exploration probability parameter, or a classifier is selected from the Match Set which has the highest product of predictive quality, q, and reward prediction, r. After this, an Action Set $[A]$ is formed from the Match Set, where each classifier's action $[A]$ is the same as the winning action. The winning action is then imparted onto the environment and the classifier sets are retained for the learning algorithms.

Three main learning algorithms work in conjunction to evolve an optimally general but accurate predictor. The aim is to evolve a classifier that encompasses the widest possible range of input conditions while at the same time providing an accurate prediction of the next state of the environment.

The anticipatory learning process provides the means by which the classifiers are able to learn the condition-action-effect mapping in a given environment. After an action has been imparted, each classifier in the Action Set is assessed to check whether it correctly predicted the next state of the environment.

If a given classifier correctly predicted its action outcome, one of two cases can occur. Should the classifier not have any marks, its quality is incremented by means of an adapted Widrow-Hoff delta rule [34] as shown in Formula 2.2.

$$q' = \begin{cases} q + \beta(1-q) & \text{if correct prediction} \\ q - \beta q & \text{otherwise} \end{cases} \tag{2.2}$$

where:

q' is the classifier's new quality value

q is the classifier's previous quality value

β is the learning rate

Otherwise if the markset is not empty, the system attempts to generate a new classifier by first creating a clone and then adding don'tcare symbols to positions in the condition bitstring and to a difference bitstring, which represents the symbols that are different between the markset and the triggering condition. The number of added don'tcare symbols is determined by maintaining the specificity of the two bitstrings above a specificity threshold. Finally, the new classifier clone is specialized with the remaining bits in the difference bitstring that are not empty and do not have don'tcare symbols.

On the other hand, if a given classifier does not correctly predict its outcome, its quality is decreased as dictated by Formula 2.2. Next, the environmental input for which the classifier fired is recorded in the Mark Set. After this, if the classifiers condition bitstring has don'tcare bits which can be specialized to match the input condition, a new clone is generated with those bits set and the effect bitstring is adjusted accordingly to correctly predict the outcome.

If no correctly predicting classifier is found and if no alternative candidate classifier is generated, a new classifier is formed by a covering algorithm. The algorithm creates a new classifier, whose condition, action and effect bitstrings are equal to the observed state transition triplet. Furthermore, the quality and reward prediction parameters are initialized to default values and the numerosity and experience parameters are both set to one. Finally, the learning timestamps are set to the current iteration number.

During learning, reward prediction parameters are updated by a reinforcement learning-like algorithm in a manner similar to what could be utilized in traditional Q-learning. The reward received from the environment is passed onto all classifiers inside the Action Set $[A]$ by means of Formula 2.3.

$$r' = r + \beta \left(\rho + \gamma \max_{cl \in [M](t+1)} (q \cdot r) - r \right) \tag{2.3}$$

where:

r' is the classifier's new reward prediction parameter

r is the classifier's previous reward prediction parameter

ρ is the reward from the environment

β is the learning rate

γ is the discount factor

This effectively builds up a prediction of what the average reward will be attained when the classifier is utilized.

The final component of the system is a generalization algorithm, based on the traditional genetic algorithm that performs two tasks. First, it selects candidate classifiers from the Action Set $[A]$, upon which it performs a crossover operation on the condition bitstring to find better matching conditions. It also performs a mutation operation on the condition bitstring. It should be noted, however, that the mutation operator only sets don'tcare values in the bitstring, rather than inserting specific bits. This is due to the desired objective of generalizing the classifiers rather than specializing them (as this already occurs effectively during the Anticipatory Learning stage).

Assuming that the system converges to a stable population, a model will emerge inside the system that provides a condensed temporal representation of how a set of possible actions for all known condition will affect the environment. Due to the focus on maintaining high quality predictions without full regard for the reward attained, the model itself covers as much of the area as the system is trained on. Furthermore, due to the reinforcement learning component of ALCS, it is possible to ascertain the desirability of each alternative action for a given environment state.

The complete architecture of the system is shown in Figure 2.1.

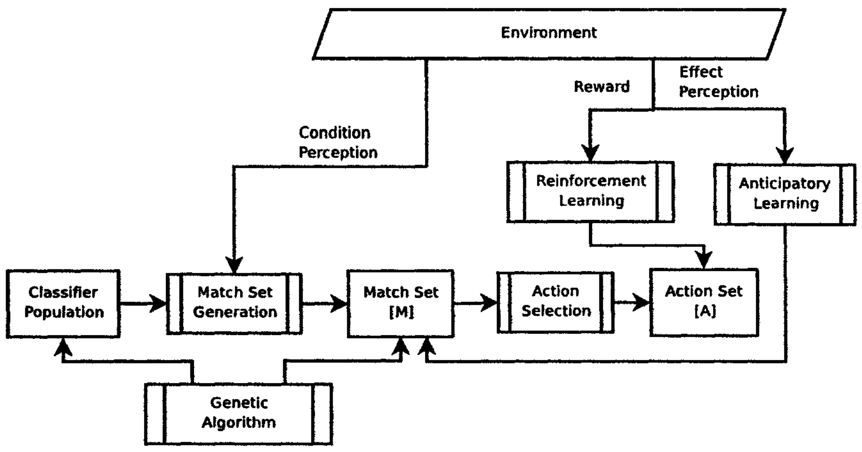


Figure 2.1: Structure of ALCS

The advantage of holding an anticipation value for an intelligent system, in addition to a condition-action pair, comes directly from psychology [35]. It has been shown through animal experiments that many behaviors are far more complex than what can be modeled using classical Pavlovian behavior of Stimulus-Response [36]. Animals, and Humans in particular, are capable of forming models of their environments without receiving any direct reward from the surroundings, and yet they are extremely capable of using this knowledge "when the time is right".

A classical demonstration of this fact comes from rat in maze experiments [37] [38]. These experiments are conducted in two phases. In the first phase, each individual from a group of the rodent subjects is allowed to explore given maze with no food present. After this exploratory phase, a desirable treat for the rat is placed in a goal location and the rat is placed at the starting position. The time is measured to see how long it will take the rat to find its treat. In the second phase of the experiment, a different set of rats is used to measure how long they will take to find the desirable treat from a starting position without first being allowed to explore the maze. It has been shown conclusively that the rats which have previously explored the maze have a significantly faster search time than those which did not have the opportunity to explore the maze. Hence, it can be concluded that latent learning does occur, where the rats do form a representation of the maze and are able to use this representation to find the goal much faster.

More so, it is well known that human thinking is far more complex than just responding to stimuli. We are able to choose a particular course of action depending on what we perceive to be the most desirable outcome and we can predict what outcome our actions will have.

Hence, it can be easily seen that the main advantage of using anticipations in a control system for a robot or agent is that it can learn its environment without having to be given an explicit reward. With such an approach, it can be placed in an unknown surrounding, and allowed to explore it initially without having any direct goal. Upon introducing an objective, the robot or agent can attain this objective rather than relying on a pure reward mechanism. This has been illustrated by replicating an experiment similar to the aforementioned rat experiment using Khepera robots [39].

2.4 Fuzzy Logic

Fuzzy logic [40] [41] was originally introduced by Zadeh [42] in the 1960's as a way to represent imprecise human knowledge in a more formal mathematical manner. Classical boolean logic deals with absolutes: either something is true or it not, something is part of a set or is excluded from a set. However, in the real world, not everything can be stated in such absolutes; many concepts have degrees of truthfulness and degrees of belongingness to different sets. Furthermore, humans usually use general and approximate expressions of information on a daily basis as opposed to using precise quantities. As such, it is difficult to map the vague information into classical mathematical formulations.

To accommodate qualitative knowledge, fuzzy logic utilizes the notion of fuzzy *membership functions*. This allows a specific instance x from the domain, termed *universe of discourse* of X, to have a range of possible membership to a particular set, A, between 0 and 1. Thus, when the belongingness or membership is given as 0, x is completely excluded from the set A and when the membership is 1 it is completely included in A. The real advantage comes from the range of values in between 0 and 1 which allows a complete range of degree of belongingness to the set. Examples of possible membership function types include the delta function, trapezoid function, sigmoid function, Gaussian function, generalized bell function, and so forth. The basis for the fuzzy membership function is shown in Formula 2.4 below:

$$\mu_A(x) : X \rightarrow [0,1] \tag{2.4}$$

By associating linguistic descriptors with membership functions, one is able to encode imprecise human concepts such as "near" and "far", "hot" and "cold" to varying degrees of belongingness on a range of possible values in the universe of discourse.

The different operators such as AND, OR, NOT, union, etc are defined in fuzzy logic through appropriate mathematical operators that meet specific criteria. This allows one to use fuzzy logic in normal boolean algebra equations. For specifics, see [41].

To obtain a crisp set of elements belonging to fuzzy set A whose degree of membership is equal or greater than a particular threshold value α, an *alpha cut* is taken of the membership function. This process is defined by Formula 2.5:

$$A_\alpha = \{x \in X | \mu_A(x) \geq \alpha\}, \alpha \in [0,1] \tag{2.5}$$

In order to encode human knowledge with fuzzy logic, a set of *if-then* or *if-then-else* rules is utilized which allows an expert to encode associations between a condition or *antecedent* with a particular output, or *consequent.* As such, they take on the form of IF condition X is x_1 then output Y is y, where x and y are fuzzy values as opposed to specific numerical values. To accommodate for more complex input space, separate antecedents are stringed together though the use of the AND operator.

To obtain a specific output or inference, I, from the knowledge base of fuzzy rules, R, a process known as the *compositional rule of inference* (CRI) is utilized. This process forms an agregate function composed of the current data D along with the entire set of fuzzy rules present. This is formally stated in Formula 2.6:

$$I = D \circ R \tag{2.6}$$

Furthermore, the actual membership function obtained from the CRI process is defined as:

$$\mu_I = \sup \min \left(\mu_D, \mu_R\right) \tag{2.7}$$

In order to obtain a crisp numerical value which can be utilized by a real-world process from an output u, a defuzzification process is employed which either employs a threshold method such as an α-cut or a weighted combination of the rule inferences. One such possibility is the center of gravity method which finds the centroid of a given inference membership function. This is shown in Formula 2.8:

$$\hat{u} = \frac{\int_u u \cdot \mu_I(u)\, du}{\int_u \mu_I(u)\, du} \tag{2.8}$$

The typical structure of a fuzzy controller is illustrated in Figure 2.2. First, a real-world value is passed through a fuzzification stage where its belongingness to different fuzzy membership functions is assessed. Next, the fuzzifed value is used to perform a composition of the different rules present in the fuzzy rule base by means of an inference engine. Finally, the composite function is defuzzified to give a resulting crisp output which is then utilized by given application. It is also possible to have a closed-loop feedback system which utilizes sensors to observe the actual outcome of the controller and compare it against the desired value. This is the reprocessed back into the fuzzification stage to repeat the cycle and infer the next output value.

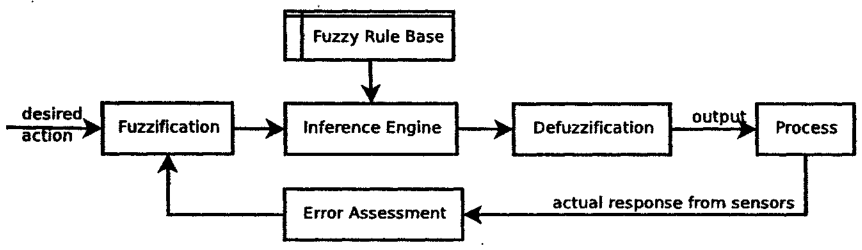


Figure 2.2: Structure of a typical fuzzy controller

Fuzzy logic control systems are particularly well suited for applications where it is not possible to form a model of the process itself but rather it is necessary to rely on a set of known good heuristic rules that dictate what action to perform for a set of input conditions. In addition, control systems based on fuzzy logic have the advantage of transparency to a human user. They encode the solution to the control problem in a readily recognizable form thus allowing for easy analysis and modification by an expert [43]. Furthermore, through the inherent ambiguity present, the fuzzy controllers are able to handle continuous ranges without overcomplicated mathematical formulations, which consequently help to preserve the simplicity of a compact human readable rule base. Fuzzy logic controllers are also able to gracefully handle noisy inputs that normally exist in real-world systems without causing erratic behavior.

Chapter 3

FALCS Architecture

This chapter documents the development of the Fuzzy Anticipatory Learning Classifier System (FALCS) and describes all the underlying mechanisms which allow it to function. It describes the concepts borrowed from Fuzzy Logic to extend the parent Anticipatory Learning Classifier System (ALCS) to work with real-valued problems, as well as modifications performed upon ALCS to allow it to transparently use the fuzzified classifiers.

3.1 System Overview

The structure of the FALCS system can be seen in Figure 3.1. The description of the individual components is described in the proceeding sections.

3.2 Fuzzy Classifier

For the transparent operation of the Anticipatory Learning Classifier System with real-valued inputs, the use of the bitstring based classifiers has been replaced by sequences of functions. The condition bitstring effectively becomes a set of fuzzy membership functions based on generalized bell functions.

The action bitstring becomes a set of singleton variables which can take on a number of predefined numerical values, hence they are referred to as "granulated" singleton bits in this thesis. The rationale behind using discrete intervals for the actual actions is to allow for the learning mechanisms to operate effectively upon the classifiers, in particular when determining if two

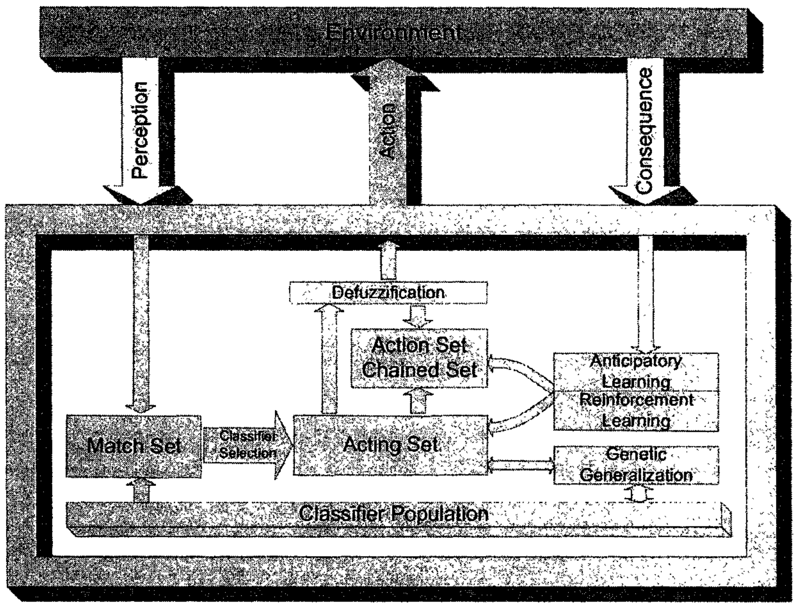


Figure 3.1: Overall structure of FALCS

classifiers perform the same action. It should be noted that a full range numerical output values emerges through the use of defuzzification, as described in Section 3.6.2. The number of granules can be directly related to the concept of fuzzy resolution, where the greater the number of granules, the finer the resolution and, subsequently, control. However, the tradeoff is a significant increase in the population size as well as training time required to learn the finer control rules. Also, using a smaller number of granules reduces the possibility of falling into a local minima and taking a long time to escape from it. The tradeoff off is a more general solution which is learnt faster.

Finally, the effect bitstring becomes a set of fuzzy membership functions based on the generalized bell function. However, default width of these functions is much narrower compared to that of the condition bitstring and the overall shape is much sharper i.e. it has a more trapezoidal shape as opposed to a bell shape as illustrated in Figure 3.2.

The complete triplet, *cl*, is shown below in Formula 3.1.

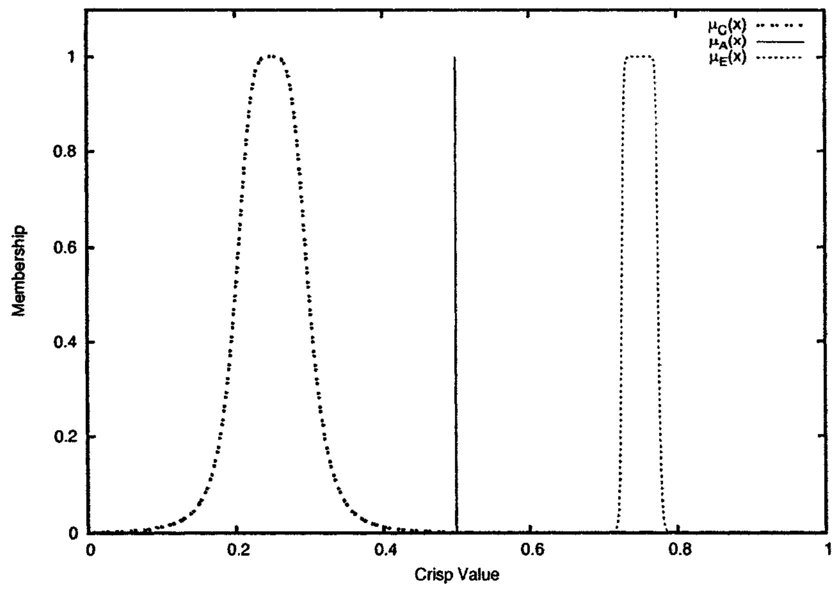


Figure 3.2: Example of fuzzy membership functions utilized

$$cl = \begin{cases} C = \left\{\mu_{C_1}(x), \mu_{C_2}(x), \ldots, \mu_{C_{d_p}}(x)\right\} \\ A = \left\{\mu_{A_1}(x), \mu_{A_2}(x), \ldots, \mu_{A_{d_a}}(x)\right\} \\ E = \left\{\mu_{E_1}(x), \mu_{E_2}(x), \ldots, \mu_{E_{d_p}}(x)\right\} \\ \textit{internal parameters} \end{cases} \tag{3.1}$$

where:

C is the classifier's condition fuzzy membership function set

A is the classifier's action fuzzy membership function set

E is the classifier's effect fuzzy membership function set

d_p is the perceivable dimensionality

d_a is the actable dimensionality

The concept of markset is no longer present inside the fuzzy classifiers due to feasibility and effectiveness issues. It has been replaced with an alternative form more appropriate for the method of encoding the input conditions. This is further described in Section 3.8.1.

The indicative parameters present in each classifier system are the same as those of the Anticipatory Learning Classifier System.

3.3 Rule Generation

In order for the learning classifier system to be useful, a population of rules embodied as classifiers must exist. These classifiers are the means through which the system learns i.e. stores knowledge and interacts with its surroundings. Thus, to allow for consistent and effective learning, well-defined classifier generation method must be devised.

3.3.1 Random Classifiers

In order to bootstrap the system during the first stage of learning, an initial random set of classifiers is generated according to Formula 3.2. These classifies are limited in number and have different but relatively general input conditions, each with different random actions whose effect will be imparted onto the environment. This allows for an initial fast exploration of the working environment in the hopes that at least some will provide a seed for new classifiers to be generated. However, for complex and high-dimensional environments, this method provides limited utility due to the low probability of actually finding the correct condition-action-effect combination, even with fairly general inputs.

$$cl = \begin{cases} \mu_{C_i}(x) = \left(1 + \left|\frac{x-\text{Random}(0,1)}{0.5\cdot\text{Random}\left(0,\text{i}_{\text{w}_\text{C}}\right)}\right|^{2s_C}\right)^{-1} \\ \mu_{A_i}(x) = \delta\left(x - \text{Random}\left(0,1\right)\right) \\ \mu_{E_i}(x) = \left(1 + \left|\frac{x-\text{Random}(0,1)}{0.5\cdot\text{Random}\left(0,\text{i}_{\text{w}_\text{E}}\right)}\right|^{2s_E}\right)^{-1} \\ q = i_q \\ r = i_r \\ n = 1 \\ e = 1 \end{cases} \tag{3.2}$$

where:

$\mu_{C_j}(x)$ is the classifier's condition membership function for input j

$\mu_{A_j}(x)$ is the classifier's action membership function for input j

$\mu_{E_j}(x)$ is the classifier's effect membership function for input j

$\delta(x)$	is a delta/singleton function
$cl.q$	is the classifier's quality parameter
$cl.r$	is the classifier's reward prediction parameter
$cl.n$	is the classifier's numerosity
$cl.e$	is the classifier's experience
i_w	is the initial membership function width
s	is the membership function sharpness modifier

3.3.2 Covering

The main process through which classifiers are created inside the Fuzzy Anticipatory Learning Classifier system, is known as covering. Each time that a condition-action-effect triplet is encountered and no existing classifier matches the input, action and output sequence, or when no classifier with adequate quality exists, a new classifier is generated such that it matches the triplet sequence observed in the environment as shown in Formula 3.3. Other parameters such as initial reward and quality are set to default values (see Section A.1.1 in the Appendix for details). This initial classifier generated through the covering procedure is then handed off to the classifier insertion mechanism (described in Section 3.8.2), which aims to reduce redundancy and increase the generality of the population. Therefore, there is no guarantee that this particular instance will actually be part of the population. If the newly generated classifier can be merged with an existing classifier, a merging process described in Section 3.8.4 is performed.

$$cl = \begin{cases} \mu_{C_i}(x) = \left(1 + \left|\frac{x-\sigma_i(t)}{0.5w_C}\right|^{2s_C}\right)^{-1} \\ \mu_{A_i}(x) = \delta\left(x - a_i\left(t\right)\right) \\ \mu_{E_i}(x) = \left(1 + \left|\frac{x-\sigma_i(t+1)}{0.5w_E}\right|^{2s_E}\right)^{-1} \\ q = i_q \\ r = i_r \\ n = 1 \\ e = 1 \end{cases} \tag{3.3}$$

where:

$\sigma(t)$ is the environment condition perception at time t

$\mu_{C_j}(x)$ is the classifier's condition membership function for input j

$\mu_{A_j}(x)$ is the classifier's action membership function for input j

$\mu_{E_j}(x)$ is the classifier's effect membership function for input j

$\delta(x)$ is a delta/singleton function

$a(t)$ is the action taken at time t

q is the classifier's quality parameter

r is the classifier's reward prediction parameter

n is the classifier's numerosity

e is the classifier's experience

w is the initial membership function width

s is the initial membership function sharpness modifier

3.4 Match Set Generation

Upon probing the environment for a numerical representation of the current situation, the first step of the Fuzzy Anticipatory Learning Classifier algorithm is to generate a Match Set $[M]$ in a manner similar to the original algorithm.

Each classifier inside the population is checked for match with the current environmental perception. The degree to which a given classifier matches a given input is determined by an average of all the condition fuzzy membership functions. This is illustrated by Formula 3.4. It should be noted that if a particular membership function is denoted as a don'tcare, the value of the membership function is automatically set to $\mu_c(x) = 1.0$.

$$\mu_M^{cl_i}(\sigma) = \frac{\sum_{j=1}^{d_p} \mu_{C_j}(\sigma_j)}{d_p} \tag{3.4}$$

where:

μ_M is the degree of match membership function

σ is the environment condition perception

d_p is the environment condition perception dimensionality

$\mu_{C_j}(x)$ is the classifiers condition membership function for input j

If the degree of match is above a predefined threshold, α^T, the classifier is deemed to match the current situation.

$$doesMatch(\sigma, cl_i) = \mu_M^{cl_i}(\sigma) > \alpha^T \tag{3.5}$$

where:

σ is the environment condition perception

α^T is the acceptance threshold parameter (i.e. alpha-level)

The given classifier is added to the Match Set population as show in Formula 3.6.

$$[M](t) = \{cl \in [N] \mid doesMatch(\sigma(t), cl)\} \tag{3.6}$$

where:

$[M](t)$ is the Match Set at time t

$[N]$ is the classifier population

$\sigma(t)$ is the environment condition perception at time t

3.4.1 Actable Set Generation

In situations were it is known that certain actions cannot be performed due to physical or other constraints, it is possible to integrate this knowledge directly into the system for faster learning. This is done by creating an Actable Set, $[C]$, which contains all classifiers from the Match Set which do not include a Disallowed Action Set, D. This process is stated formally in the formula below:

$$[C](t) = \{cl \in [M](t) \,|\, cl.A \not\subseteq D\} \tag{3.7}$$

where:

$[C](t)$ is the Actable Set at time t

$[M](t)$ is the Match Set at time t

D is the disallowed actions set

While forming an Actable Set is not explicitly required for proper system operation, it helps to speed up the learning process by disregarding certain types of actions that should always be excluded. The disadvantage of using this method is that the model inside FALCS does not contain these undesirable actions. If such behavior is desirable, the Actable Set is used in place of the Match Set for classifier selection, Acting Set generation and learning algorithms.

3.5 Rule Selection

In order to choose an action to impart onto the environment, a mechanism needs to be in place to select a winning classifier, i.e. the classifier which best represents the given state of the environment and whose action will be utilized to form the action to be imparted onto the working environment. To accomplish this objective within the FALCS architecture, five major methods of selecting classifiers have been implemented. Each method has its own benefits and appropriate situations when it is most useful.

3.5.1 Max Q-R

The first classifier selection mechanism comes directly from the parent ALCS algorithm. It is based on an elitist selection ideology that deems the winning classifier the one which has the highest predictive quality of the next state of the working environment and at the same time provides the highest reward from the environment as described by Formula 3.8. The methodology behind this selection mechanism can be seen as a greedy best-first search strategy [44] which aims to maximize the short term profit without explicitly taking into consideration any long term strategy. However, this provides an excellent method for choosing an action when a non detrimental step needs to be taken, such as during active system operation when the system should not try to perform actions that have not been verified to be acceptable. Furthermore, this selection mechanism provides the fastest means of selecting the action, and may be the only viable means of choosing an action under real-time constraints within a large population. This process is shown in Formula 3.8:

$$cl^{w} = \underset{cl \in [M](t)}{\arg\max} \left(cl.q \cdot cl.r \right) \tag{3.8}$$

where:

cl^{w} is the winning classifier

$[M](t)$ is the Match Set at time t

$cl.r$ is the classifier's reward prediction parameter

$cl.q$ is the classifier's quality parameter

To form the actual Acting Set, $[T]$ which will later be used to calculate the actual crisp action, the top n_{acl} classifiers selected using the above formula are inserted into $[T]$.

3.5.2 Best Known Action

When working with real-valued inputs in large dimensional spaces, it can be expected that no matching classifiers will be present for numerous input situations, especially during the initial training phase. Relying solely on using random actions to fill this void would be a significant impediment to the learning mechanism in the learning classifier system. This is because of the enormous search space which would need to be handled to find a single adequate solution. However, due to the use of fuzzy logic, it is possible to choose the best matching classifier from the existing population, even if the match degree of the best matching classifier is below the acceptable threshold. This process is illustrated in Formula 3.9. This best matching classifier is selected as the winner and is utilized as if it were normally chosen by the other selection mechanisms. The actual crisp numerical value is calculated in the same manner as for fully matching classifiers, as presented in Section 3.6.2.

$$cl^{w}\left(t\right) = \underset{cl\in[N]}{\arg\max}\left(degreeMatch\left(\sigma\left(t\right), cl\right)\right) \tag{3.9}$$

where:

cl^{w} is the winning classifier

$[N]$ is the classifier population

$\sigma\left(t\right)$ is the environmental perception

When the selected classifier's action is utilized, one of two situations may occur. First, a new classifier will be generated and an attempt will be made to add it to the population. Due to the generalization mechanism in place (see Section 3.8.2), it is very likely that it will be merged with the classifier which spawned the action. Secondly, if the action performed was deemed detrimental, i.e. the immediate reward is below the min_r threshold, the condition membership functions of the originally spawning classifiers will be shifted away from the triggering condition by means of the ALP function learning procedure described in Algorithm 3. It should be noted that the user must set the min_r value to a level which indicates what actions receiving an immediate reward value below the specified threshold are never acceptable. Furthermore, due to the fact that the newly generated classifier has a reward value below min_r, an exploratory action (described in Section 3.5.5) will be utilized.

3.5.3 Roulette Wheel Selection

In order to improve upon the exploratory mechanisms present inside FALCS, a roulette wheel selection mechanism has been introduced as an alternative rule selection mechanism. Roulette wheel selection mechanisms are commonly used in genetic algorithms to select chromosome for genetic operators. The particular implementation used for classifier selection inside FALCS is detailed in Algorithm 1.

Algorithm 1: RouletteSelect

```
Input  : Match Set $[M]$
Output: Winning classifier $cl^w$
begin
    $m_r \leftarrow \min_{cl \in [M]} (cl.r)$
    $s \leftarrow \sum_{i=1}^{|[M]|} (cl_i.n \cdot cl_i.q \cdot (cl_i.r - m_r))^d$
    $p \leftarrow s \cdot \text{Random}(0,1)$
    $i \leftarrow 1, j \leftarrow 0$
    repeat
        $j \leftarrow j + (cl_i.n \cdot cl_i.q \cdot (cl_i.r - m_r))^d$
        $i \leftarrow i + 1$
    until $j \geq s$
    return $cl_i$
end
```

where:

$cl.r$ is the classifier's reward prediction parameter

$cl.q$ is the classifier's quality parameter

$cl.n$ is the classifier's numerosity

d is a small factor, altering the selection probability difference between strongest candidate and weakest candidate

---

The benefit of using the roulette wheel as opposed to the elitist selection stems from the likely selection of higher quality classifiers while still allowing less proven classifiers to be selected, although with much smaller probability. As such, it eliminates the dominance of high quality classifiers from monopolizing and thus focusing on a niche in the environment. Thus, the method provides a broader range of actions, all of which may be beneficial to the desired solution. It can also discover previously unexplored paths to the system's objective which would have otherwise been ignored.

For more cooperative action generation, this classifier selection method is also used to select a predefined number of classifiers, dictated by the n_{acl}

variable, for insertion into the Acting Set, $[T]$, to perform an action. This allows for cooperative action generation as with the Max Q-R method, but also for varied combinations of classifier's actions, given that the probability of a classifiers selection is in proportion to its quality and reward value.

While this selection mechanism is excellent for training an initial classifier population to a completely unknown environment, due to the randomness of the rule selection mechanism it may not be an ideal candidate for use in a production operating environment. This is due to the fact that it may lead to a previously unencountered state which is not desirable when only exploitative operation is required i.e. when only proven solutions are to be utilized. Therefore, for these types of situations it is best to use the roulette wheel selection mechanism during the initial phase of training, and then switch to a more stable action selection mechanism such as the "Max Q-R" and the "Best Known Action".

3.5.4 Desired Effect

This method of selection comes from the fact that the system provides an anticipated effect of each classifier's action. Hence, after an initial training that builds up an effective population of rule triplets, it is possible to choose the action based on the desired effect that will occur in given environment as shown in Formula 3.10. This is of particular use when one wishes to use more complex action planning algorithms.

$$cl^{w}(t) = \arg\max_{cl\in[M]} \left(degreeSimilar\left(de, cl.E\right)\right) \tag{3.10}$$

where:

cl^{w} is the winning classifier

$[M]$ is the Match Set

de is the desired outcome in the environment

E is the classifier's Effect

However, unlike the parent Anticipatory Learning Classifier System, the possibility of chaining rules in a forward manner i.e. performing plan ahead is not readily possible in the current FALCS implementation. This is due to the fact that the fuzzy output prediction does not specify an exact output like in the discrete case. It is not known to which particular state an action will lead; rather a rough estimate of the different possible number of states which are very similar to each other is known, and these states are considered the same when presented to a FALCS classifier.

3.5.5 Exploratory Actions

In order to facilitate a life-long learning process, the concept of taking exploratory actions was incorporated from previous classifier systems (including ALCS) into the FALCS architecture. Exploratory actions are required in the classifier system to prevent the learning classifier system from stagnating in a suboptimal state such as being stuck at local minima, both in terms of the reward attainable from the environment as well as from attaining the best possible predictions for future outcomes. Without such actions, the system would continue to choose the current elite classifiers without taking into consideration that better actions are possible. Also, it would never recognize that the current action being performed may not be the best suited alternative.

To allow for exploratory actions within FALCS, a fuzzy membership function generator has been designed to generate new singleton action bits in one of two ways. For environments where only a single action is possible i.e. only one of the outputs may be active at a time, a random action generator first chooses a particular output bit, and then assigns a random value (singleton function) in the allowed output range. However, for environments where all bits can take on different random values, all action bits are randomized.

No matter which main classifier selection method is used, a probability function exists where a random action is chosen if a drawn random number falls below an exploration probability threshold, ϵ. This ensures that the learning classifier system is able to discover previously unseen alternatives which would be otherwise missed. Furthermore, when only a single classifier is inserted into the Acting Set, and its reward value is below the min_r threshold, the exploratory action generation mechanism will also be utilized in an attempt to find an action that would be deemed more acceptable by the working environment.

3.6 Action Generation

3.6.1 Discrete Output

When the environment upon which FALCS operates does not require (or does not accept) real-valued actions, the system can use the granulated singleton bits as discrete symbols for outputs. Also, in such a configuration, only one winning classifier needs to be selected for insertion to the Acting Set, $[T]$, i.e. $n_{acl} = 1$. In this case, the degree the winning classifier matches given situation can be seen as its corresponding confidence value.

3.6.2 Real-valued Output

The classifier's condition membership function is augmented by a scaled reward component as shown in Formula 3.11. The inclusion of the reward scaling is to reduce/eliminate the utilization of detrimental actions in the solution.

$$\mu_{c_i'} = \mu_{c_i} \cdot \left(\frac{cl.r - \min\limits_{cl \in [T]} (cl.r)}{\max\limits_{cl \in [T]} (cl.r) - \min\limits_{cl \in [T]} (cl.r)} \right) \tag{3.11}$$

where:

$\mu_{c_i'}$ is the classifier's effective condition membership function for action generation

μ_{c_i} is the classifier's condition membership function

r is the given classifier's reward

In the case that real-valued actions are desirable, n_{acl} classifiers in the Acting Set, $[T]$, are composed together as shown in into the composed function I, Formula 3.12. Each classifier keeps track of its contribution to the solution,

κ. This whole process effectively forms what could be seen as the inference engine in a classical Fuzzy Control system.

$$I = P \circ [T].C \tag{3.12}$$

where:

P is the set of perceived environmental data

$[T]$ is the Acting Set

C is the set of classifier's condition membership functions

I is the composed inference

$$\mu_I(p) = \max_S \min(\mu_P, \mu_{C'}) \tag{3.13}$$

where:

μ_I is the composed inference membership function

μ_P is the membership function of the perceived environment condition

μ_C' is the classifier's condition membership function (see Formula 3.11)

S is domain of possible perceptions P

The crisp value, which is then presented to the working environment as the action to be imparted onto, is determined through the a Center of Gravity (CoG) defuzzificaton process. This is illustrated by Formula 3.14

$$a = \frac{\int_P \sigma \mu_I(\sigma)\, d\sigma}{\int_P \mu_I(\sigma)\, d\sigma} \tag{3.14}$$

where:

a is the generated action to be imparted onto the environment

σ is the perceived environmental situation

P is the set of perceived environmental data

μ_I is the composed fuzzy membership function

A top-down view of the actual action generation procedure is provided in Figure 3.3.

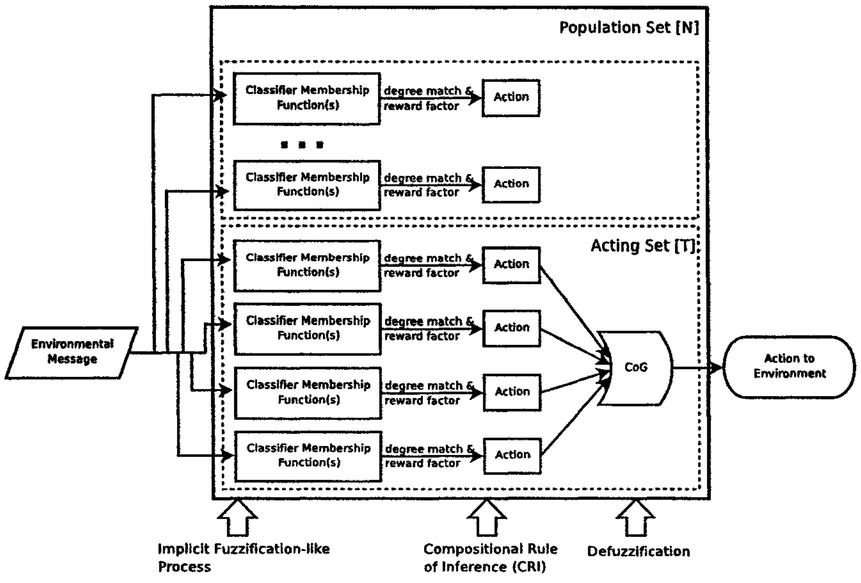

Population Set [N]
Classifier Membership Function(s)
degree match & reward factor
Action
Acting Set [T]
Environmental Message
CoG
Action to Environment
Implicit Fuzzification-like Process
Compositional Rule of Inference (CRI)
Defuzzification

Figure 3.3: Structure of FALCS action generation mechanism

3.7 Action Sets Generation

3.7.1 Action Set

To properly credit all classifiers which correspond to the generated action, an Action Set, $[A]$, is formed in a manner similar to the parent Anticipatory Learning Classifier System. The most dominant classifier which has the greatest impact on the solution is automatically inserted into this set. In addition, all classifiers that match the given chosen action are also inserted into this set.

$$[A]\,(t) = \{cl \in [M]\,(t)\,|cl.A \sim a\,(t)\} \tag{3.15}$$

$[A](t)$ is the Action Set at time t

$[M](t)$ is the Match Set at time t

$cl.A$ is the classifier's action

$a(t)$ is action performed at time t

3.7.2 Contribution Set

To allow for learning to occur in classifiers which contributed to the solution via the fuzzy composition and defuzzification steps (i.e. are present in the Acting Set $[T]$, but are not included within the Action Set because they do not have the same type of action as that which has been ultimately generated) a Contribution Set, $[K]$, is formed according to Formula 3.16. This set includes all classifiers which are not part of the Action Set but made a contribution to the solution greater than a predefined threshold $\kappa > min_{\kappa}$.

$$[K]\,(t) = \{cl \in [T]\,(t)|cl.\kappa > min_k, cl \notin [A]\,(t)\} \tag{3.16}$$

where:

$[K](t)$ is the Contribution Set at time t

$[T](t)$ is the Acting Set at time t

$[A](t)$ is the Action Set at time t

κ is the classifier's contribution to the action

min_{κ} is minimum contribution amount required for accreditation

3.7.3 Chained Set

Each effect in the classifier is dependent on the previous action. In many situations, such as in robot navigation, the difference between a given state and the next may be very small. To compensate for this, a stack of n_{ch} classifiers is maintained to provide the learning mechanisms a means of identifying which classifiers were used in sequence to attain given consequence in the environment as show in Formula 3.17. It should be noted however, that this set only contains winning classifiers, and does not account for other classifiers such as those in the Action and Contribution Sets.

$$[H](t) = \{cl \in [N] \, | IsMainClassifier(cl(u)), t > u > (t - n_{ch})\} \tag{3.17}$$

where:

$[H](t)$ is the stack of classifiers at time t

n_{ch} is the length of the chain

$[N]$ is the classifier population Set

3.8 Anticipatory Learning Process

The majority of the anticipatory learning concepts implemented in the ALCS algorithm were incorporated into FALCS in their original spirit while allowing for learning to take place with the fuzzy membership functions as opposed to discrete symbols. However, many changes have been made to allow learning for more general fuzzy membership functions. A generalized overview of the main methodology behind the anticipatory learning process utilized in FALCS can be seen in Algorithm 2.

Algorithm 2: AnticipatoryLearningProcess

```
begin
    foreach $cl \in [A]$ do
        increment $cl.e$
        update learning timestamp
        if does cl anticipate outcome correctly then
            if $cl.r$ similar to $\rho$ then
                increase condition matching
            end
            increment $cl.q$ according to Formula 2.2
        else
            decrease condition matching
            decrement $cl.q$
            create more specialized classifier based on cl if possible
            if $cl.q <$ then
                Delete (cl)
            end
        end
    end
end
```

3.8.1 Function Learning

The Anticipatory Classifier System parent algorithm utilized a concept of *mark bits* to note each input condition where a classifier failed to correctly predict the next state of its working environment. This enabled the system to evolve a population which would minimize the number of incorrectly predicted outcomes while at the same time allow a partially working population to exist. Directly importing this concept into the real-domain where each incorrect input condition would be noted is not feasible due to the boundless number of marks that could be generated per classifier. Furthermore, utilizing these marks would prove challenging due to the need for an equality operator which would need to compare if a mark already exists in a system or not: it would be necessary to define how close must a particular value be to be considered equivalent. Small variations of this threshold could have vastly significant influence on the learning characteristics.

The remedy for this dilemma follows directly from the utilization of the expanding and contracting membership functions. Because each "bit" in the classifier's bitstring is in fact a fuzzy membership function, it is possible to note the positions of inputs where the classifier fails to generate correct results. Thus, every time a classifier is utilized to generate an action for given condition, if the effect is not correctly anticipated, the current classifiers quality is decremented as shown in Formula 2.2. Furthermore, each membership function composing the classifier condition string is contracted away from the input situation by a rate dictated by the function learning parameter η_{fl}. Also, if the given input value for a particular membership function is between the two bounds for that particular bit, the nearest bound is shifted towards the input value by a degree dictated by the boundary learning rate, η_{bl}.

The major idea behind using the bounds is to avoid oscillations where the input membership functions would expand in a particular phase of learning and then contract in another phase. The added benefit is the higher quality value that a given classifier will attain as it will not need to be reprimanded for firing due to too general condition bitstrings.

The two main procedures utilized in the function learning process are the enhance and inhibit matching shown in Algorithms 3 and 4 respectively.

Algorithm 3: EnhanceMatching

```
Input: membership function $\mu_A(x)$, crisp value $v$

begin
    $c \leftarrow$ center of $\mu_A(x)$
    $l \leftarrow \arg\min_{x \in X} (\mu_A(x) > 0.5)$
    $r \leftarrow \arg\max_{x \in X} (\mu_A(x) > 0.5)$
    if $v < c$ then
        $d \leftarrow \max(l - v, 0)$
        $l' \leftarrow \max(l - \eta_{fl} \cdot d, b_l)$
        $r' \leftarrow r$
    else
        $d \leftarrow \max(v - r, 0)$
        $r' \leftarrow \min(r + \eta_{fl} \cdot d, b_r)$
        $l' \leftarrow l$
    end
    if $l' \approx l$ AND $r' \approx r$ then
        increase sharpness of $\mu_A(x)$
    end
    adjust $\mu_A(x)$ such that $\mu_A(l') \approx 0.5$ and $\mu_A(r') \approx 0.5$
end

where:
    $b_l$ is the left boundary
    $b_r$ is the right boundary
    $\eta_{fl}$ is the function learning rate
```

Algorithm 4: InhibitMatching

```
Input: membership function $\mu_A(x)$, crisp value $v$

begin
    $c \leftarrow$ center of $\mu_A(x)$
    $l \leftarrow \arg\min_{x \in X} (\mu_A(x) > 0.5)$
    $r \leftarrow \arg\max_{x \in X} (\mu_A(x) > 0.5)$
    if $v < c$ then
        $d \leftarrow \max(v - l, 0)$
        $l' \leftarrow \max(l + \eta_{fl} \cdot d, b_l)$
        $r' \leftarrow r$
        if $v > b_l$ then
            $b_l \leftarrow b_l + \eta_{bl}(v - b_l)$
        end
    else
        $d \leftarrow \max(r - v, 0)$
        $r' \leftarrow \min(r - \eta_{fl} \cdot d, b_r)$
        $l' \leftarrow l$
        if $v < b_r$ then
            $b_r \leftarrow b_r - \eta_{bl}(b_r - v)$
        end
    end
    if $l' \approx l$ AND $r' \approx r$ then
        decrease sharpness of $\mu_A(x)$
    end
    adjust $\mu_A(x)$ such that $\mu_A(l') \approx 0.5$ and $\mu_A(r') \approx 0.5$
end
```

where:

b_l	is the left boundary
b_r	is the right boundary
η_{fl}	is the function learning rate
η_{bl}	is the boundary learning rate

3.8.2 Insertion

Before a classifier is inserted into the population, it is first verified that it does not already exist or it is not subsumed by another classifier.

In order to avoid adding redundant classifiers, while at the same time ensuring that the most fit classifiers are promoted, a subsumption check is performed each time a new classifier is added into the population. This procedure is a set of checks with aim to find either a more general classifier which subsumes the insertion candidate or an existing classifier which is similar to a prescribed degree to the candidate. If a subsuming classifier is found, then its numerosity parameter is increased as is its quality. Furthermore, the classifier's condition and effect membership functions are checked and adjusted if necessary to better match the condition and effect which spawned the new insertion candidate.

If a subsuming classifier is not found, the population is then checked for classifiers that could be joined together with the insertion candidate. If this is possible, the new candidate is merged into a clone of the already existing classifier and is inserted into the population. This is to ensure that an already good classifier is not destroyed in the merging process, in case the new combination does not prove effective (i.e. its condition is too general or the predicted effect is too broad and covers too wide of a range of possibilities)

Otherwise, the new candidate classifier is inserted into the population normally.

3.8.3 Subsumption

In order to reduce the number of redundant classifiers and to promote the development of highly reliable yet as general as possible classifiers, a subsumption check algorithm has been incorporated into FALCS. In principle, this check operates much in the same manner as in the original Anticipatory Learning Classifier System.

This procedure performs a series of checks between two classifiers as outlined in Algorithm 5 listed below. It is run iteratively upon the entire population, whereby a discovered subsumer becomes the candidate and terminates when all possible classifiers have been checked. The last classifier to successfully pass all the checks subsumes the original candidate.

Algorithm 5: SubsumptionTest

```
Input: original classifier $o_{cl}$, candidate classifier $c_{cl}$

begin
   if $o_{cl}.condition \supseteq c_{cl}.condition$ AND
        $o_{cl}.action = c_{cl}.action$ AND
        $o_{cl}.effect \supseteq c_{cl}.effect$ AND
        $o_{cl}$ function bounds $\leq c_{cl}$ function bounds AND
        $o_{cl}.quality \geq c_{cl}.quality$ AND
        $o_{cl}.reward \geq c_{cl}.reward$ AND
        $o_{cl}.experience \geq c_{cl}.experience$
   then
      return $o_{cl}$ subsumes $c_{cl}$
   else
      return $o_{cl}$ does not subsume $c_{cl}$
   end
end
```

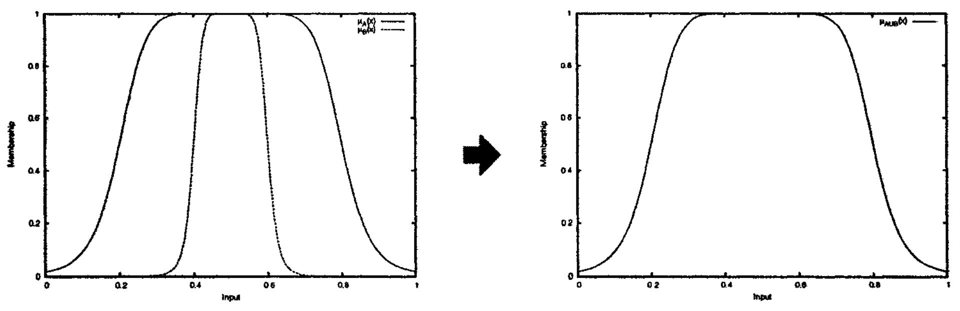
These checks are aimed at finding the most general classifier possible while at the same time ensuring that the chosen classifier has the highest possible predictive quality and is beneficial to the working environment.

This check is needed to ensure that we only throw away candidate classifiers if there indeed exists a proven classifier that is more general and is well performing as opposed to a freshly inserted general classifier whose condition and effect may be a superset of the candidate but in reality its performance is poor due to the fact that it has not yet been fully taught and evaluated. Furthermore, it is needed to ensure that we find the most fitting individual from a set of similar classifiers for further enhancement as opposed to a random choice. Moreover, the checks ensure that the subsuming classifier has better experience, thus it has more reliable knowledge about the condition-action-effect mapping, resulting in a more stable classifier population.

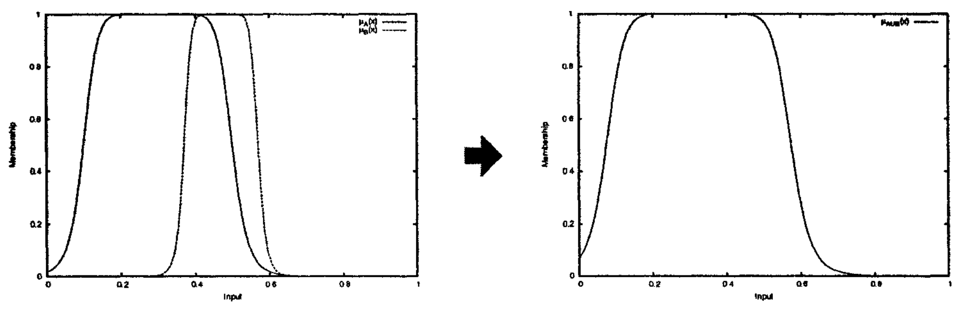
3.8.4 Merging

In order to ensure that only similar classifiers are merged, a set of criteria must be met:

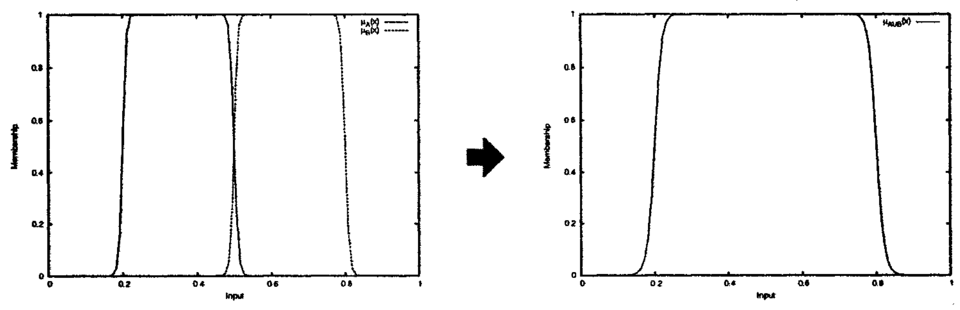
- both classifiers' must correspond to the same action
- at least one of the fuzzy membership function bits inside the condition or effect bitstring can be combined together (as shown in Figure 3.4).
- both classifiers' quality parameters must be greater than the reliability threshold, θ_q
- both classifiers' reward prediction parameters must be greater than the inadequacy threshold, θ_r
- the classifiers' quality parameters must not differ more than Δ_q
- the classifiers' reward prediction parameters must not differ more than Δ_r
- both classifiers' experience must be greater than the experience threshold, θ_e



(a) One function completely overlaps another



(b) One function partially overlaps another



(c) Two touching side-by-side functions

Figure 3.4: Range of allowable cases for membership function merging

If all the aforementioned conditions are met, then the two classifiers can be merged. The reward parameter is merged together based on a weighted average with the classifier's experience and numerosity value (i.e. how many individual classifier instances would exist if no merging and subsumption check were used) are added up together. The quality parameter is the lowest value of the two classifiers and the remaining parameters are summed together. The effective computation of these new values is illustrated in Formula 3.18. Each fuzzy membership bit is combined together with the other corresponding membership value and the bounds are merged together based on the most restrictive possible combination.

$$cl_m = \begin{cases} C = \text{Merge}\left(\text{cl}_1.\text{C}, \text{cl}_2.\text{C}\right) \\ r = \frac{cl_1.r \cdot cl_1.n \cdot cl_1.e + cl_2.r \cdot cl_2.n \cdot cl_2.e}{cl_1.n \cdot cl_1.e + cl_2.n \cdot cl_2.e} \\ q = \min\left(cl_1.q, cl_2.q\right) \\ n = cl_1.n + cl_2.n \\ e = cl_1.e + cl_2.e \end{cases} \tag{3.18}$$

where:

cl_m is the newly merged classifier

C is the set of Condition fuzzy membership functions

r is the classifier's reward prediction parameter

q is the classifier's quality parameter

e is the classifier's experience

n is the classifier's numerosity

3.9 Reinforcement Learning Process

Due to the changes in the manner that a given classifier's action is transformed into an action imparted onto the environment, the reinforcement learning algorithm method used in FALCS must account for the varying amount of contribution that each classifier carries in generating given action which then leads to the particular consequence.

The classifiers present in the Action Set are allocated a reward amount in the same manner as that in the parent ALCS algorithm, as show in Formula 3.19.

$$cl.r' = cl.r + \eta_{rl}\left(\rho + \gamma \max_{cl \in [M](t+1)} (cl.q \cdot cl.r) - cl.r\right) \tag{3.19}$$

where:

$cl.r'$ is the classifier's new reward prediction parameter

$cl.r$ is the classifier's previous reward prediction parameter

ρ is the reward message from the environment

η_{rl} is the reinforcement learning rate

γ is the discount factor

To account for the fact that pervious actions have an effect for the current situations, the $n_c s$ previous acting classifiers are updated with a small reward dictated by learning rate augmented by a factor whose magnitude is inversely proportional to the classifiers position in the chain, β/i. This results in an exponentially decreasing reward. The complete rewarding function is illustrated in Formula 3.20

$$cl.r' = cl.r + \frac{\beta}{i}\eta_{rl}\left(\rho + \gamma \max_{cl \in [M](t+1)} (cl.q \cdot cl.r) - cl.r\right) \tag{3.20}$$

where:

$cl.r'$ is the classifier's new reward prediction parameter

$cl.r$ is the classifier's previous reward prediction parameter

ρ is the reward message from the environment

β is the learning rate augmenting factor

i is the position of a given classifier in the chain

η_{rl} is the reinforcement learning rate

γ is the discount factor

Classifiers having a substantial effect on the solution need to be rewarded for their contribution. Therefore, each classifier from the Contributing Set has its reward parameter updated in a manner similar to that of the main Action Set. To reflect their limited role, a smaller learning rate, η_{rl_κ}, is used and the degree to which they matched the input condition is accounted for. The augmented update formula has the following form:

$$cl.r' = cl.r + \kappa\eta_{rl_\kappa}\left(\rho + \gamma \max_{cl \in [M](t+1)} (cl.q \cdot cl.r) - cl.r\right) \tag{3.21}$$

where:

$cl.r'$ is the classifier's new reward prediction parameter

$cl.r$ is the classifier's previous reward prediction parameter

ρ is the reward message from the environment

κ is the classifier's contribution to an action

η_{rl_κ} is the contribution set reinforcement learning rate

γ is the discount factor

3.10 Genetic Generalization

The main concept behind the genetic generalization is very similar to the parent Anticipatory Learning Classifier System method. However, rather than working on individual discrete symbols, the GA works upon sets of membership values. It should be noted, that the genetic algorithm operates on whole membership functions and does not split or recombine functions together. A major change is the extension of the mutation operator that can expand the membership functions to include a wider range of possible inputs which provide a valid match as opposed to just being able to set a particular bit as a don'tcare. The effect of settings a bit to a don'tcare value can be seen as setting the membership value to a constant of 1 i.e. $\mu_A(x) = 1$.

3.11 Population Control

Due to the immense expansion in the search space cased by the inclusion of real-valued ranges as inputs, actions and outputs, additional mechanisms must be considered to ensure that the system does not grow out of hand which would result in extensive memory usage and computational infeasibility.

3.11.1 Forced Merging

In order to control the population growth FALCS executes an algorithm to compact the population after a predefined number of learning steps (i.e. after a single training session or n_{cit} iterations). Each classifier in the population is compared against all others to check whether a merger is possible. If so, the condition-action-effect bitstrings are combined together, as are all the augmenting parameters as dictated by the merging process.

3.11.2 Size Control

FALCS does not perform any direct limitations to the population size. While this type of behavior is readily implementable within the FALCS framework, it is not desired due to goal of representing the working environment as completely as possible. By directly limiting the population size, the system focus is restricted to particular niches of the environment. This would be beneficial for small embedded systems with limited computing resources or for systems aiming to exploit only a specific environment. However, the ultimate objective is to allow the system to provide an optimal course of action with accurate next state predictions for as many situations as possible.

To this end, there are alternative means of restricting the population size present inside the FALCS architecture. The three primary ways means of restricting unbound population growth are:

1. Restricting the maximum match set size $n_{[M]}$

2. Restricting the number of acting classifiers n_{acl}

3. Removing unfit classifiers after each training session based on poor reward and poor quality when better alternatives exist.

The maximum Match Set size limit ensures that there are at most $n_{[M]}Cn_{acl}$ main solutions. While this does reduce the comprehensiveness of the model, the information that is thrown away would be of negligible use. This is due to the fact that a very large number of matching classifiers could be formed that would correctly predict the next state but be of little use due to a low environmental utility. The process of enforcing the Match Set size is show in Algorithm 6.

Algorithm 6: EnforceSetSize

```
begin
    $e \leftarrow$ Size([M]) $- n_{[m]}$
    $e \leftarrow min(e, 0)$
    $i \leftarrow 0$
    while $i <$ Size([M]) $AND$ $e > 0$ do
        if $cl_i.q < min_q$ then
            Delete($cl_i$)
            $e \leftarrow e - 1$
        end
        $i \leftarrow i + 1$
    end
    while $e > 0$ do
        $cl = \arg\min_{cl \in [M]}(cl.r)$
        Delete(cl)
        $e \leftarrow e - 1$
    end
end
```

Restricting the number of acting classifiers forces the system to only maintain the best actions and not keep the alternatives. Furthermore, because only a small number of classifiers are used to generate the action, learning should occur faster as there are fewer possible actions that could be formed and less classifiers to evaluate.

Forcibly removing bad classifiers helps ensure that the system does not have to deal with unneeded classifiers that would otherwise take up valuable computing resources. It also ensures that the classifiers do not get used though the exploratory processes when no better solutions exist, but rather promote the system to come up with new alternative possibilities.

3.12 Effective Model

When the FALCS system is trained to a degree satisfactory to the user, the system embodies a model of the working environment in the form of condition-action-effect triplets. These form what could be described as a $d_p + d_a$ dimensional model of the working environment, where d_p is the perception dimension, i.e. the inputs to the system; and d_a is the action dimension, i.e. the degrees of freedom though which the system can interact with the environment by means of effectors. In effect, the classifiers from a piece-wise function for every single condition previously encountered and trained to respond with up to g number of granulated actions, which were tested during training and/or operation, to give the expected next state in the environment. It should be noted that this generated model is a continuously changing approximation of a snapshot of its working environment which continues to be updated after every single action taken unless this behavior is inhibited.

Due to this configuration, it is possible to extract useful, human readable information. In particular, it is possible to map out a sequence of steps that will most likely need to be performed to go from state A to state B without the need to actually perform the action. Furthermore, it is possible to perform backwards search to provide a list of actions and the prior conditions that can result in a known output state. As such, the information gleaned is more human readable and can be analyzed by experts to give insight into the functionality of the system and/or to analyze given environment. This is a clear advantage compared to alternative approaches such as neural networks.

However, the system does not aim to provide definite knowledge of both the input and output states. It only provides a range of possible numerical values which could trigger a given rule and a range of possible output numerical values. This is a direct result of incorporating fuzzy logic into the system. Furthermore, FALCS does not aim at putting human readable labels to membership functions, but rather forms these functions as it deems fit for the given environment and for the range that give the best results. Thus, it is up to the human user to assign linguistic labels such as “close”, “near”, ”far“ as well as linguistic modifiers such as “very” or “slight” to the functions, generated as a result of learning. This behavior is by design to allow the learning classifier to come up with its own subjective representation of a particular grouping of data without undue influence of a human designer.

Chapter 4

Experiments

In order to evaluate the performance of the devised system, two sets of experiments have been performed. In the first experiment, FALCS has been used to provide goal-oriented behavior for a simulated agent. In the second experiment, FALCS has been used as an obstacle avoidance controller as well as an objective-based controller for a mobile robot.

4.1 Simulated Agent

4.1.1 Overview

In order to provide a platform simple enough to debug and test the Fuzzy Anticipatory Learning Classifier System while still allowing for comprehensive evaluation of the performance and effectiveness of the new algorithm, a simulated agent test system was devised. This simulation environment was modeled on the simple electronic toys known as "virtual pets" or "nano pets". The objective in this game is to keep a virtual animal or creature alive as long as possible by means of appropriately applying different necessary actions such as feeding the pet, playing with it, giving it medicine, letting it sleep, etc. These actions are conceptually similar to actions that an autonomous robot would need to perform when aiming to carry out an objective without a human to supervise and care for it, but of course in a much simplified fashion. It also illustrates that FALCS is a potential candidate for controlling in-game characters.

This scenario provides a toy domain in which the system must devise a set of procedures to care for the virtual agent. Each rule performs a particular action which has an observable effect that can be deemed good (pet is content and lives), bad (pet complains or dies) or neutral (no immediate observable change). Furthermore, each action has a fairly deterministic outcome on the environment in that a particular action on the environment will result in the same outcome each time rather than giving a random output. Thus, this experiment would show that the Fuzzy Anticipatory Learning Classifier System can mimic the behavior of a child playing with a nano pet toy.

The additional benefit of this type of simulation is that it can be performed faster than real time which allows for easier evaluation of the system, and it also allows to give consistent and repeatable trial runs. Hence, it can be used to tune parameters as well as observe the effect of introducing and removing different methods inside the Fuzzy Anticipatory Learning Classifier System.

4.1.2 Setup

A simple version of a virtual pet was implemented where the virtual pet had five distinct parameters: tiredness, hunger, fitness, health and happiness. All five conditions were fed to the Fuzzy Anticipatory Learning Classifier System as a value in the range of [0..1]. In turn, FALCS could select to perform one of the following actions: sleep, eat, play, exercise, take medication, see a doctor. A limitation was enforced, where only one action can be performed at a single time. The maximum duration that the agent can live regardless of its internal parameters was set to 1000 learning steps.

The rewarding function utilized has three components: the first component ensures that no critical parameter of the simulated agent is outside a predefined threshold; if there is a problem and it is not being remedied, a penalty is imposed. The second component provides a minimal reward for non-detrimental actions that improved the overall status of the agent. The final component checks the state of the agent. If the agent has either attempted to perform an illegal operation or is dead, a heavy penalty is given. The effective rewarding scheme is summarized in Formula 4.1.

$$\rho(t) = \sum_{P} \begin{cases} -100 & p(t) > \Theta_{\mathrm{p}} \\ -5 & p(t) > \theta_{\mathrm{p}} \\ \alpha_{\mathrm{p}} \cdot (p(t) - p(t-1)) & otherwise \end{cases} \tag{4.1}$$

where:

ρ is the environmental reward at time t

P is the set of parameters

$p(t)$ is an observed parameter of the agent at time t

Θ_{p} is the critical parameter threshold

θ_{p} is a soft, non-critical threshold

α_{p} is the parameters importance

4.1.3 Trial Runs

A number of trial runs have been performed using the designed simulator. Results of the experiments are illustrated in Figures 4.1 - 4.4.

From these figures we can clearly see that the Fuzzy Anticipatory Learning Classifier System is capable of devising a control strategy for keeping the pet alive as long as possible.

The most direct measure of the success of the controller can be observed in Figure 4.1. This graph illustrates the number of trials that the agent has survived with respect to the total number of life cycles evaluated. From the smoothed average curve, we can clearly observe the improvement in the survival rate. Furthermore, the density of the data points illustrate how at the beginning, the classifier clearly finds successful paths to achieve the maximum age of 1000 training trials; however, it continues to explore other possibilities. We also see that the agent continues to have an average age that is less the than optimal trials after it would appear that a solution has been forged. This occurs by design due to the exploratory actions that the FALCS-based controller will continue to make.

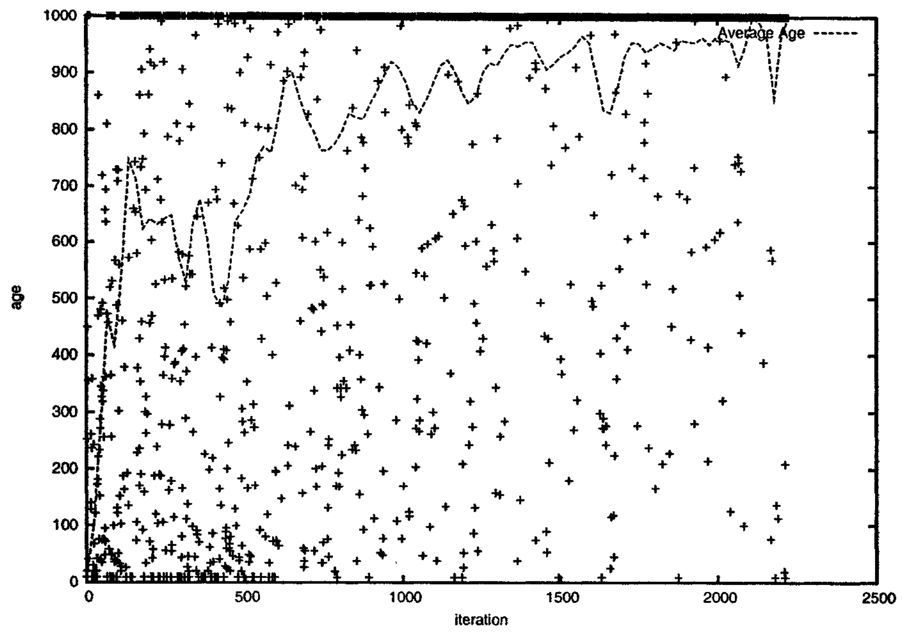


Figure 4.1: Survival Age of Simulated Agent w.r.t. Training Iterations

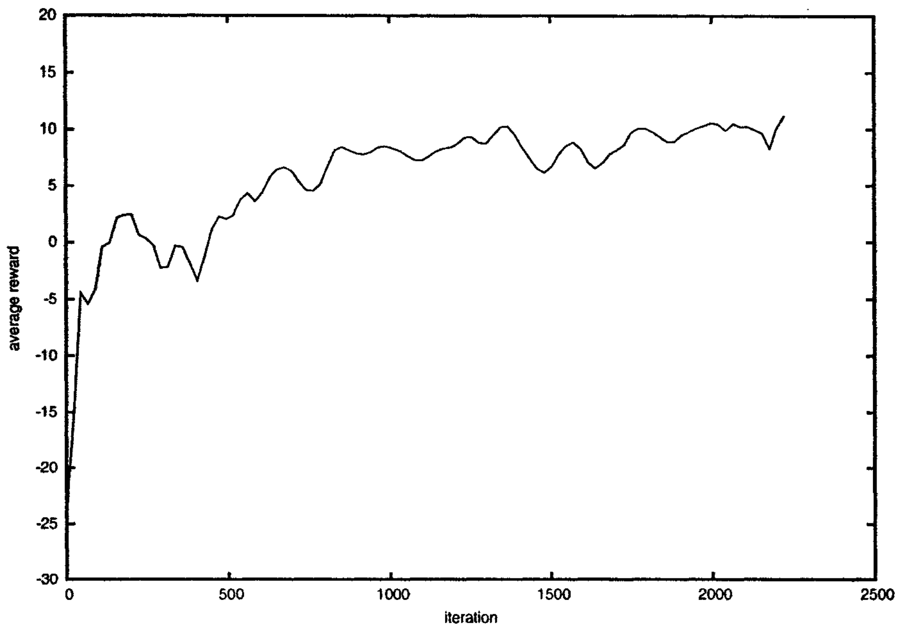


Figure 4.2: Average Reward to FALCS w.r.t. Training Iterations

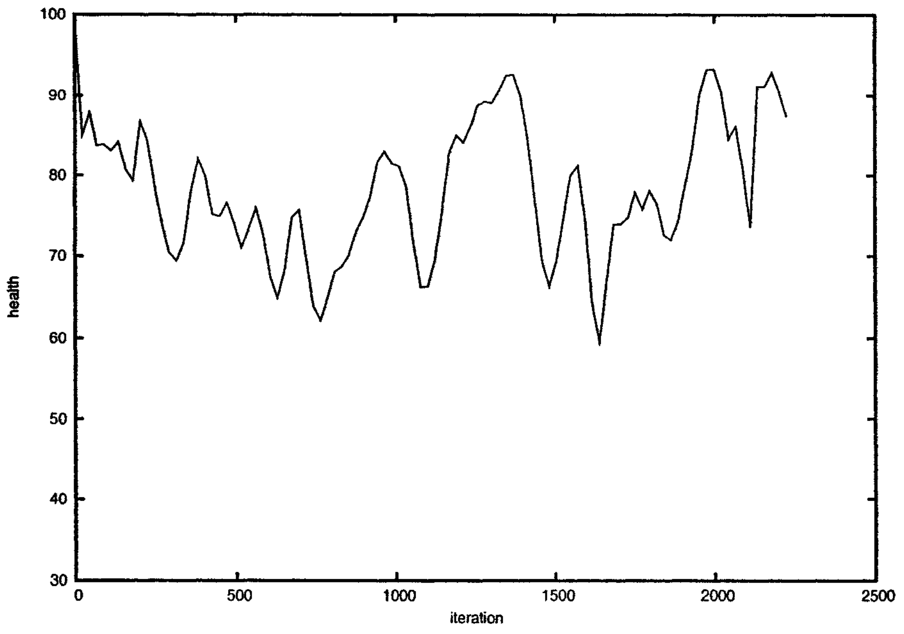


Figure 4.3: Average Pet Health to FALCS w.r.t. Training Iterations

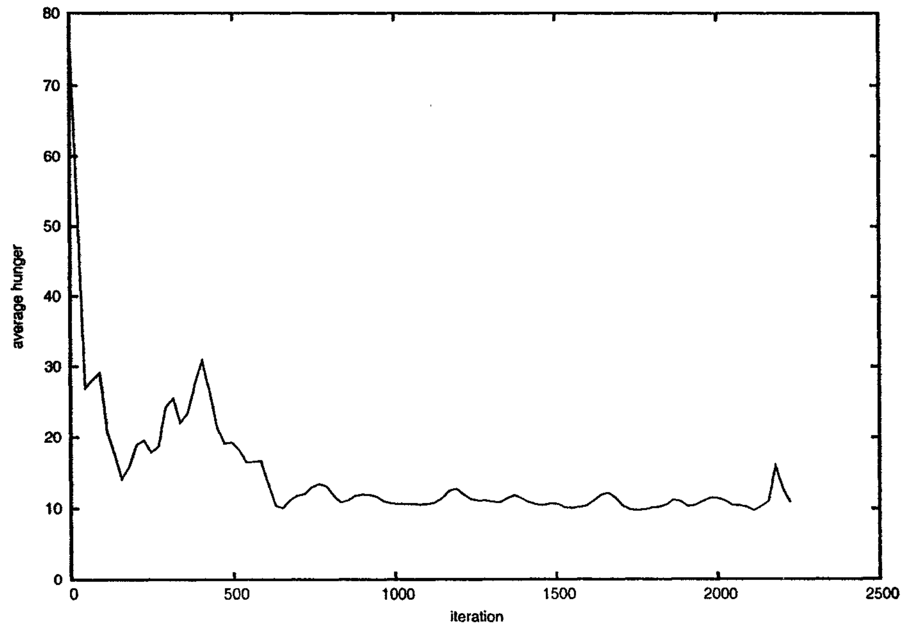


Figure 4.4: Average Pet Hunger to FALCS w.r.t. Training Iterations

The most influential characteristic of the virtual agent is the hunger level as without food, the pet will quickly die. The average hunger level per learning iteration can be seen in Figure 4.4. The plot clearly shows that the average hunger level per learning trial decreases early on, as the agent depends on having energy for survival. Taking the concept to the robotic realm, without power a robot is of no use. It should be noted that the hunger level will also have a significant influence on the well being ("health") of the virtual agent.

It is interesting to see in the average health graph, shown in Figure 4.3, a U-shaped curve. Given that the pet health is the second most influential parameter in regards to its survival, and all other parameters such as fitness, happiness, tiredness influence the health level; it can be seen that initially it is learnt that health needs to be kept at a high level.

The convergence of learning can be seen in the average reward function illustrated by Figure 4.2. It shows that initially we have many negative penalties, however, after about the first quarter of the trials, the number of critical penalties decreases and we see a leveling out with rewards hovering with small positive values in the latter portion of the learning trials.

4.2 Robot Controller

4.2.1 Objectives

One of the most important tasks in the field of mobile robotics is to provide a means for autonomous mobility, where a human operator does not need to supervise the robot in simple navigation tasks. This involves the ability for a robot to control its actuators (usually motors connected to a drive train) while at the same time avoiding obstacles in its path detected by means of sensors such as sonar, laser, LIDAR, and so forth.

The difficulty in achieving this task efficiently is the enormous space encompassed by the sensor data. This is further compounded by the fact that no sensor works ideally and usually gives rather noisy data. There can also be variances with the readings due to environmental conditions and deterioration, as well as intermittent false readings.

4.2.2 Setup

To facilitate the testing of the Fuzzy Anticipatory Learning Classifier System in a controlled and repeatable manner, the implemented robotic controller was designed using the Player Project interface. The Player program [45], [46] is an open-source implementation of a high quality mobile robot control platform which can handle a variety of different robots. It also can utilize its sister project, Stage [47] which allows for high precision robotic simulation, and this was used for the majority of the simulations performed and shown in this thesis. The Pioneer P2DX robot was chosen due to its popularity as a mobile platform and the numerous sonar sensors surrounding the circumference of its body. Driving is achieved by two drive motors which also provide a means of turning by driving the two motors at different speeds.

The simulation environment provides limited variability in the sonar readings. The data is assumed to be fairly close to what would be expected on a real robotic platform. However, for a real-world implementation, a filtering and smoothing process would be required to provide a means to reduce the effect of obvious outliers.

For the main robot navigation task, all sixteen sonar readings have been presented directly into the Fuzzy Anticipatory Learning Classifier System. The outputs from the system directly control the speed and heading of the robot. The rewarding function utilized for the development of the FALCS-based obstacle avoidance is illustrated by Formula 4.2. The training environments used to evaluate the performance of the robot can be found in Appendix A.2.

$$\rho(t) = \begin{cases} -100 & od(t) < \theta^{OD} \\ -50 + (od(t) - od(t-1)) & \text{approaching obstacle} \\ 10 + 25 \cdot (od(t) - od(t-1)) & \text{moving away from obstacle} \\ 5 \cdot fs(t) & \text{moving forward} \\ 5 + fs(t) & \textit{otherwise} \end{cases} \tag{4.2}$$

where:

ρ is the environmental reward

θ^{OD} is the closest allowable distance to an obstacle

$od(t)$ is the distance to the nearest obstacle at time t

$fs(t)$ is the robot's forward speed at time t

4.2.3 Trial Runs on Object Avoidance

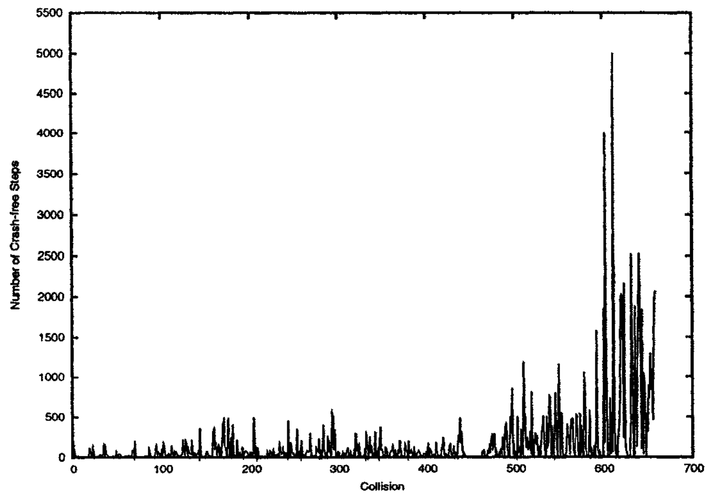


Figure 4.5: Number of steps between collisions for obstacle avoidance in Environment # 2

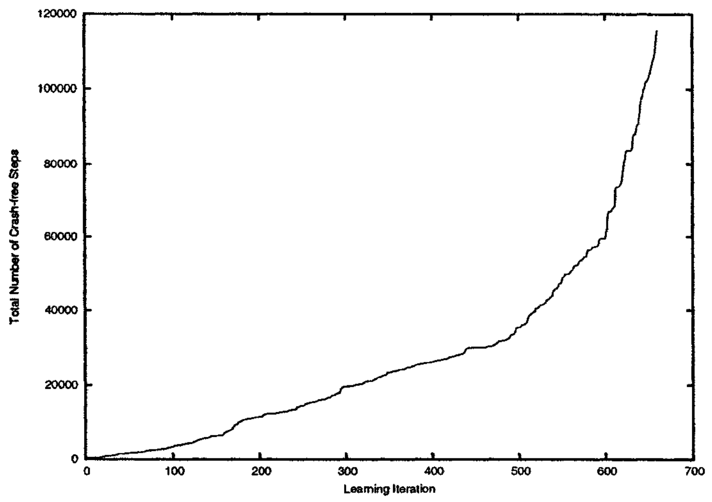


Figure 4.6: Total number of crash-free steps for obstacle avoidance for Environment # 2

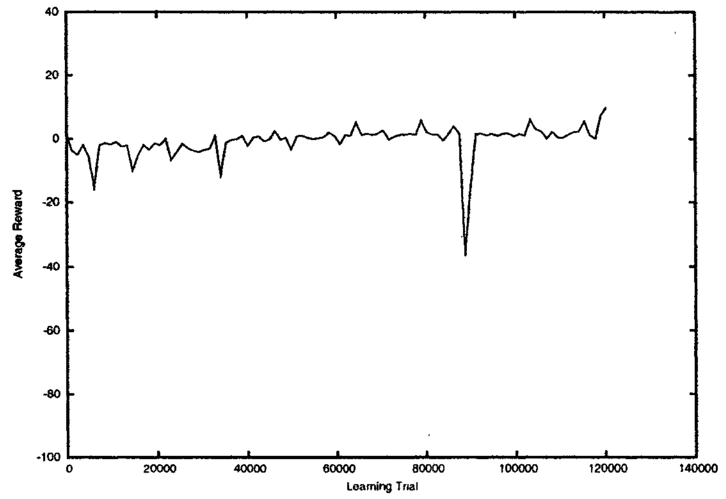


Figure 4.7: Average reward given for obstacle avoidance in Environment # 2

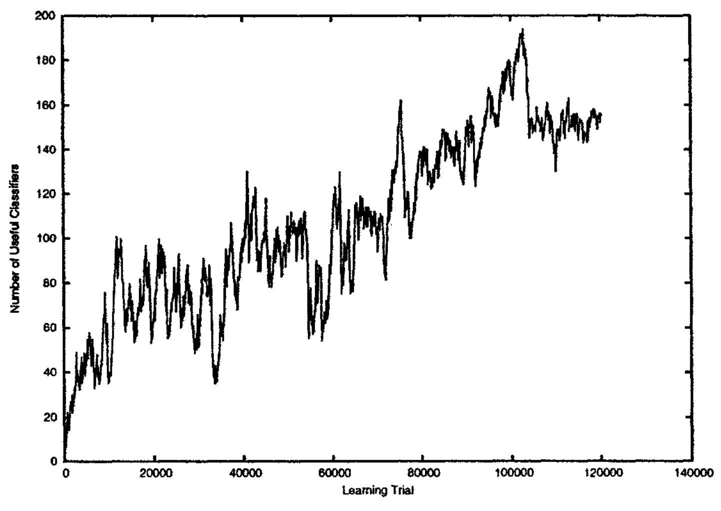


Figure 4.8: Number of useful classifiers for obstacle avoidance in Environment # 2

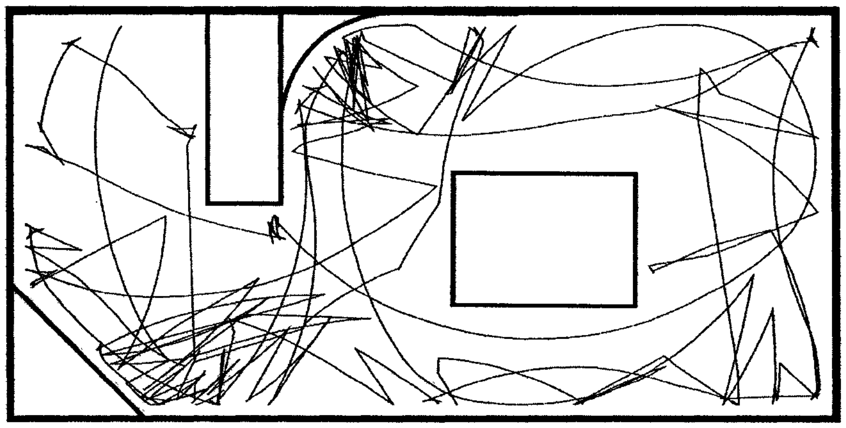


Figure 4.9: Sample trajectory in learning environment #4 for obstacle avoidance

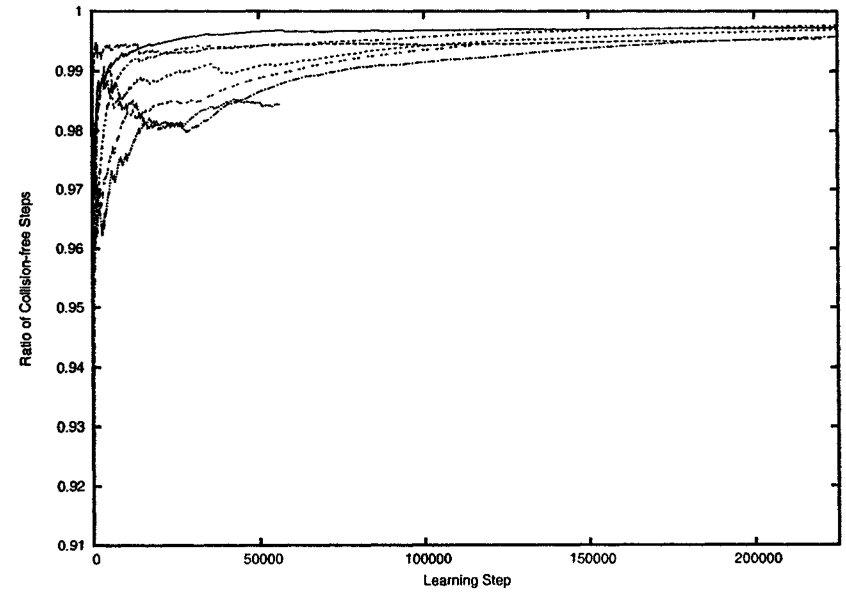


Figure 4.10: Ratio of collision-free steps to total number of steps in multiple environments

As can be seen in Figure 4.5, the FALCS-based obstacle avoidance controller incrementally learns to avoid obstacles resulting in longer durations of collision free navigation. It should be noted that even though it may seem that there appear to be frequent collisions between the larger peaks of collision-free behavior, the reality is that these are relatively brief periods of time. This illusion is a result of the fact that the x-axis does not illustrate linear time but rather the training iteration whose duration is dependent on the time between collisions. When an obstacle is hit, the robot is moved a few steps back and is then allowed to resume training. The system will take a few iterations to reprimand the incorrectly acting classifier(s) until this behavior is unlearnt.

The resulting data can also be illustrated in an alternative manner. In Figure 4.6, one can observe the total crash-free time in terms of robot movement steps with respect to the number of learning iterations. Also, the average reward function in Figure 4.7 confirms the fact that the system is learning due to the increasing average reward being applied to the classifiers making up the controller. Additional examples of learning trials can be found in Appendix A.3.

Further illustration of the system's performance can be seen by taking the ratio between the number of collision-free steps and the total number of steps for multiple simulation runs. This is shown in Figure 4.10, where one can clearly see that the FALCS-based obstacle avoidance system is capable of developing a control strategy for various situations.

The classifier population size which has been adequately trained and proven to be beneficial to the solution is illustrated by Figure 4.8. This graph includes all classifiers whose quality is above the inadequacy threshold, θ_q, reward prediction parameter is above reliability threshold, θ_r, and experience above the experience threshold θ_e.

Finally, from the recorded trajectory of the robot in the environment, one can observe the fact that the robot maintains a fairly random navigation path and does not tend to converge to a repeated behavior such as maintaining a small loop path in a safe area.

4.2.4 Goal-oriented Behavior

Obstacle-avoidance tasks are only so useful by themselves. In order to achieve sensible behavior of a robot platform, it must be allowed to search for and carry out a particular objective. The number of possibilities is endless; tasks could involve mining for a mineral, locating a target, searching for mines, finding abnormal conditions, etc.

In order to perform simulation for this task, FALCS algorithm has been extended to deal with learning multiple behaviors. This is accomplished using successive iterations which train the system to generate a population of classifiers for one objective, then train a new population for the second objective. Finally, the classifiers are merged together into a single population where they are trained on the task involving multiple behaviors.

To allow for simulations to take place on goal-seeking behavior inside the utilized simulation environment, two additional values were passed from the environment to the controller, corresponding to the robot's relative coordinates. This gives the robot a means to localize its position and to sense when it has arrived at a goal position. This is functionally identical to a sensor providing distance information to a goal beacon as an alternative simulation scenario.

4.2.5 Performance on Goal-oriented Tasks

From Figure 4.11 one can truly appreciate the learning behavior for the obstacle avoidance task when the robot is trained for an extended period of time. With a single step being an equivalent to 0.25s in real time, the total training time illustrated encompasses over 17 hours of real-time simulation. After the goal seeking behavior is trained, it can be observed from Figure 4.12 that initially, the controller fails to find the goal destination in the allotted number of 2000 steps. However, once the goal position is located, the time to reach the goal position quickly diminishes. This can be attributed to the fact that a model is first build of the environment, and once a path to the goal is discovered, the model is then quickly exploited to improve the performance.

Figure 4.14 shows a sample recorded trajectory of well-learnt goal finding behavior for environment #3 (as shown in Figure A.3). Originally, the robot starts in the lower left hand corner of the sample world and then must autonomously navigate to the goal position in the upper right hand corner of the world.

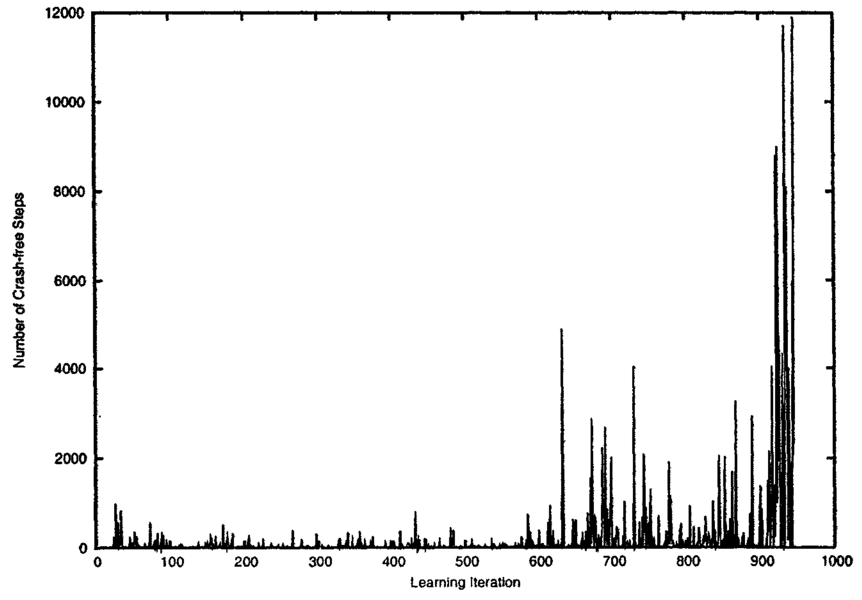


Figure 4.11: Number of steps between collisions during goal-seeking behavior training

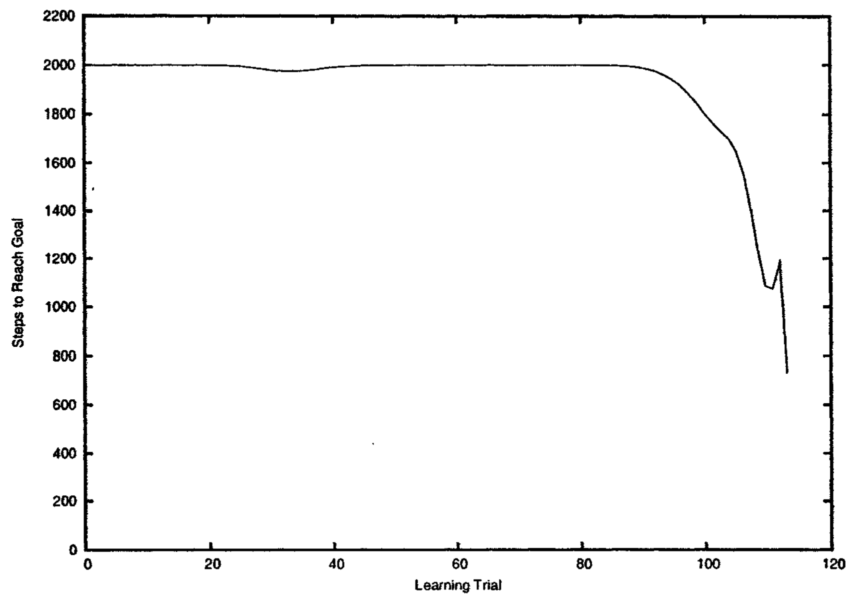


Figure 4.12: Time to seek goal position w.r.t. learning iterations during goal-seeking behavior training

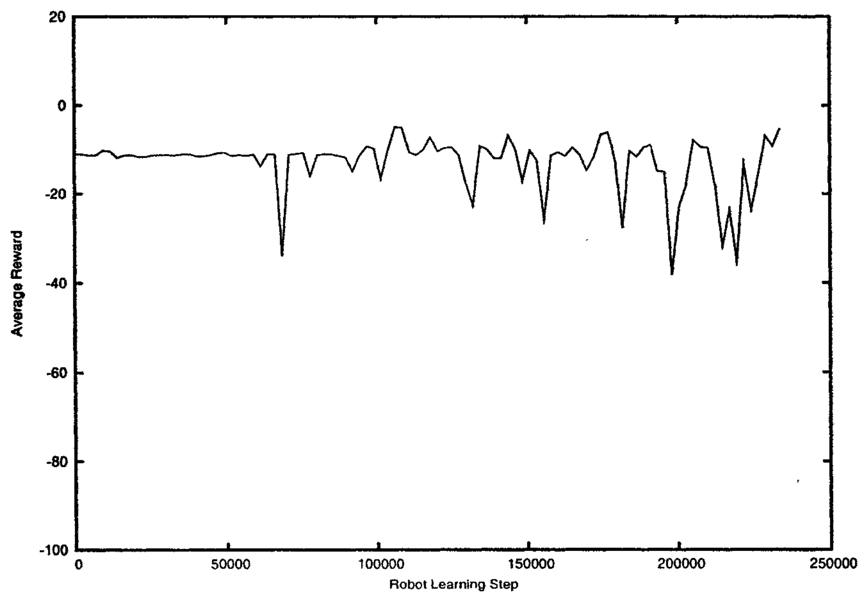


Figure 4.13: Reward function w.r.t. learning iterations for seeking goal position during goal-seeking behavior training

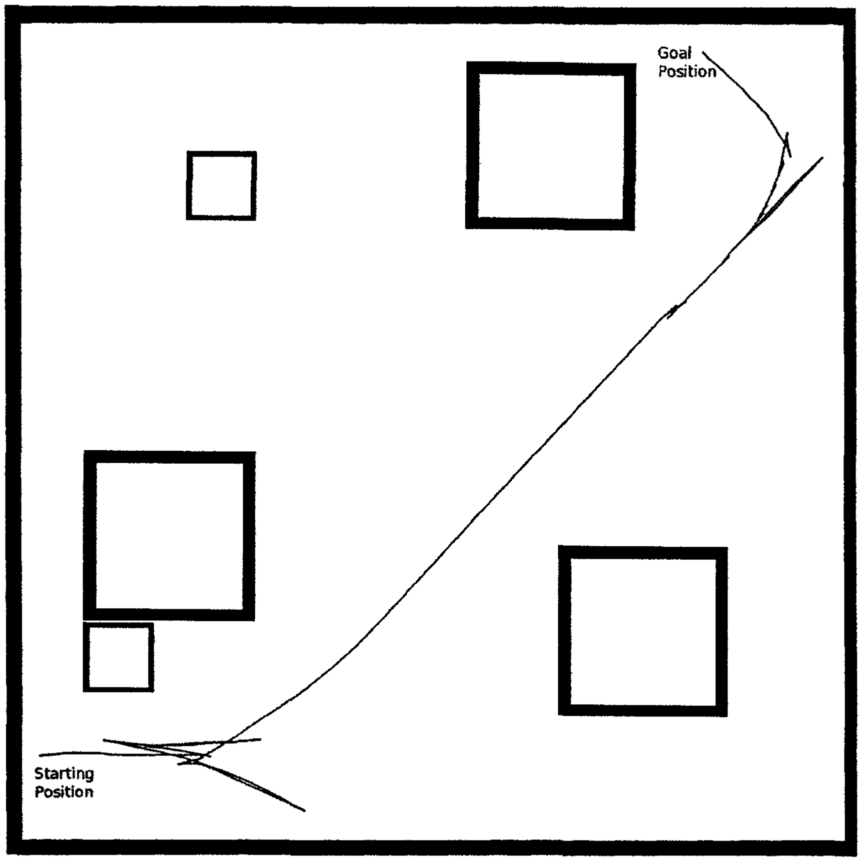


Figure 4.14: Sample trajectory in goal seeking objective with obstacles during goal-seeking behavior training

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis illustrates work performed upon the Anticipatory Learning Classifier System to allow it to be readily applied to a wider range of problems. Through the utilization of concepts borrowed from Fuzzy Logic, the rules in the learning classifier system have been enhanced to directly use real-valued inputs and provide real-valued outputs. Furthermore, the learning mechanisms present within the Anticipatory Learning Classifier System have been refined and expanded to handle and utilize the extra flexibility that the inclusion of Fuzzy Logic brings.

The Fuzzy Anticipator Learning Classifier System benefits from its constituent components in order to maximize its effectiveness. The parent Anticipatory Learning Classifier provides an elegant method to incorporate anticipatory learning processes observed in the psychology domain into a defined computational algorithm, thus allowing the system to benefit from the ability to perform latent learning. This is important for autonomous learning systems as it has been shown to be highly effective in nature.

By virtue of being a learning classifier, the system is able to evolve a set of rules which model the working environment and provide a means to choose the best course of action for a given situation. Furthermore, the learning classifier system is not constrained by a predefined structure as rules can be easily added or replaced to accommodate new environmental conditions. Thus the system does not require a redesign when more complexity needs to be trained into the system, which is a clear advantage compared to other approaches, such as neural networks. Due to the utilization of a Q-Learning like algorithm inside the system, it is capable of developing a solution to a given problem with minimal human interaction to be present. From this perspective, the major shortcoming of the system is the necessity to select a number of parameters prior to training which then proceeds with full autonomy.

Fuzzy Logic provides a means to use continuous, real-valued inputs into the system without the need to perform prior discretization. Furthermore, fuzzy logic provides a means to utilize partial knowledge gained through past learning immediately without the need to completely learn the input-action-output relationships. More so, the Fuzzy aspects allow a human expert to examine the individual rules generated in the system to gain insight into how the system solves a particular problem. Conversely, an expert can instantiate new rules from their own intuition and thus provide the system a better starting point from which to begin online training.

The implemented Fuzzy Anticipatory Learning Classifier System has been demonstrated to be effective through successful synthetic testing of a goal-oriented agent based upon a "virtual pet" game as well as well as through various simulated robotic navigation tasks. It was observed that the system does indeed aim to develop a model of the working environment in the form of a rule base, and that is able to take full advantage of this model. The anticipatory aspect of the system allows the system to learn with minimal reward to allow for better exploratory processes.

Through the contribution described in this thesis - the design and implementation of the Fuzzy Anticipatory Learning Classifier System, the boundaries of autonomous intelligent control systems, in particular those based on evolutionary computing principles, have been pushed another step towards the ultimate goal of building a truly autonomous self-learning robotic system.

5.2 Main Contributions

- This thesis illustrates a novel approach of applying Fuzzy Logic to the Anticipatory Learning Classifier System to allow it to handle real-valued inputs. This is accomplished through the replacement of discrete symbols in each classifier present in ALCS with a set of fuzzy membership functions utilizing generalized bell functions for both input matching and next state predictions as well as utilizing granulated singleton functions to specify each classifier's action.

- ALCS learning algorithms were expanded and augmented to handle the extra complexity that the continuous functions bring. This was accomplished though additional learning procedures to adjust the membership functions to find a general encoding of *input-action-effect* sequences. Further algorithms were devised to apply concepts of reinforcement learning to multiple classifiers contributing to a generated output action.

- ALCS action selection mechanisms were also improved to utilize the benefits of fuzzy logic to provide continuous valued outputs or discrete outputs with a confidence value. This was done through the implementation of classifier selection methods which choose a larger number of candidates that contribute to the final solution through a defuzzification procedure.

- This thesis also illustrates the success of the proposed system through the implementation of two test scenarios. In the first, a simulated agent scenario was implemented which illustrates that FALCS is capable of evolving a controller that can keep the agent alive in its virtual environment through the optimization of its parameters. In the second scenario, FALCS was used as a for a mobile robot navigation task, where a simulated Pioneer P2DX robot was allowed to explore its surroundings while at the same time avoiding obstacles. An extension to this system was done to introduce goal-seeking behavior though successive training on different tasks leading up to the final behavior objective.

5.3 Limitations and Future Work

Despite the many successes attained in this first revision of the Fuzzy Anticipatory Learning Classifier System, more work still needs to be performed on the system to further enhance its functionality.

The action selection mechanism, while already providing good performance, still has room for improvement. The most obvious enhancement would be to further utilize the anticipation strings i.e. predictions of the next state of a given environment, to provide a means of chaining the potential actions and then choosing a sequence of actions which would optimize for a higher level objective rather than focusing only on optimizing for the next state. However, despite its simplistic high level algorithm, the implementation of said method poses several challenges. First, a method needs to be devised which would appropriately choose the next state classifier for anticipations of broad possibilities. This is the case for classifiers which do not care about the outcome for a set of environmental conditions, because they have been deemed irrelevant, or because the outcome is so variable that it holds little co-relationship with the current task at hand. However, for the next state classifiers this may be a critical input variable which would need to be somehow predicted/recovered from the already present model. The second consideration that has to be taken into account is the fact that multiple classifiers can be utilized to arrive at the next state. Therefore, it is imperative that an accurate and repeatable method of selecting the acting classifiers and then coming up with the output value be devised. This is further complicated by the fact that it is not known how well the classifiers will match given environmental state which is crucial for defuzzifying the output.

Another area of improvement in the system is devising an accurate method of population control. The Anticipatory Learning Classifier System aims at providing an accurate model of all input-action-output combinations possible. However, in many situations, the entire model is not relevant. As Brooks has pointed out [48] "why bother re-creating an environmental model when the environment already provides the best model possible". Hence, a balance needs to be struck to preserve the optimal degree of modeling without effectively recreating the environment. It should be noted however, that this is already partially accomplished through the process of devising as general classifiers as possible. The area that needs to be addressed is how many classifier triplets need to be stored for the different actions and whether the best matching classifier procedure can be utilized to provide the knowledge without explicitly storing this information.

A different area of improvement that still needs to be taken into consideration is the selection of parameters. All intelligent systems require a certain degree of human intervention to choose an optimal set of parameters so that the system can efficiently interact with the environment. It has been experimentally found that the current implementation of FALCS provides good performange in two separate domains utilizing essentially the same set of operational parameters. However, it would be desirable to develop a method capable of finding the optimal set of parameters such as the learning rates, thresholds, etc., as opposed to human tuning through trial and error.

Furthermore, a limitation still present in the system is the fact that FALCS requires a fixed input and output dimensionality. While incomplete data input can be handled through the fuzzy aspects and the utilization of "don'tcare" bits, FALCS is not capable of handling inputs with variable dimensionality. Such a capability would provide an invaluable service to applications where sensing devices are changed on the go (such as using different robotic sensors depending on the operating environment) or to use the most applicable data without having to go through the time consuming process of learning which data being fed into the classifier system is most relevant for the current objective.

By developing and applying the suggested improvements, it is expected that the performance of the Fuzzy Anticipatory Learning Classifier System will be further refined thus being able to provide improved adaptive behavior, better runtime performance as well as increased transparency to allow a human to interpret the resulting model which enables the system to operate.

Bibliography

[1] T. Bäck and H. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[2] A. Eiben and J. Smith, *Introduction to Evolutionary Computing.* Springer, 2003.

[3] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.

[4] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, November 1859.

[5] D. Dasgupta, ed., *Artificial Immune Systems and Their Applications.* Berlin: Springer-Verlag, Inc, January 1999.

[6] L. DeCastro and J. Timmis, *Artificial Immune Systems: A New Computational Intelligence Approach.* 2001.

[7] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems.* 1999.

[8] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence.* Wiley & Sons, 2006.

[9] J. C. Spall, *Introduction to Stochastic Search and Optimization.* Wiley-Interscience, 1 ed., March 2003.

[10] D. Dasgupta and Z. Michalewicz, *Evolutionary Algorithms in Engineering Applications.* Springer, 1997.

[11] M. Ostertag, E. Nock, and U. Kiencke, "Optimization of airbag release algorithms using evolutionary strategies," in *Control Applications*, 4th IEEE Conference on Control Applications, pp. 275–280, 1995.

[12] A. K. Kordon, G. F. Smits, and M. E. Kotanchek, "Industrial evolutionary computing," in *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, (New York, NY, USA), pp. 3297–3322, ACM Press, 2007.

[13] M. Butz, D. E. Goldberg, and W. Stolzmann, "The anticipatory classifier system and genetic generalization," Tech. Rep. 2000032, 2000.

[14] E. Turban, J. E. Aronson, and T.-P. Liang, *Decision Support Systems and Intelligent Systems.* Prentice Hall, 7 ed., April 2004.

[15] F. Hoffmann, M. Köppen, F. Klawonn, and R. Roy, eds., *Soft Computing: Methodologies and Applications.* Springer, 1 ed., September 2005.

[16] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[17] C. Watkins, *Learning from Delayed Rewards.* PhD thesis, King's College, University of Cambridge, England, 1989.

[18] C. J. C. H. Watkins and P. Dayan, "Technical note: q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, 1992.

[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA.: Kluwer Academic Publishers, 1989.

[20] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA.: MIT Press, 1996.

[21] J. H. Holland, *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press, 1975.

[22] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[23] M. Marra and B. Walcott, "Stability and optimality in genetic algorithm controllers," in *Intelligent Control*, IEEE International Symposium, (Dearborn, MI, USA), pp. 492–496, September 1996.

[24] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," *SIGART Bull.*, no. 63, pp. 49–49, 1977.

[25] M. Dorigo, "Message-Based Bucket Brigade: An Algorithm for the Apportionment of Credit Problem," in *Proceedings of European Working Session on Learning '91, Porto, Portugal* (Y. Kodratoff, ed.), no. 482, pp. 235–244, Springer-Verlag, 1991.

[26] L. Bull and T. Kovacs, eds. Springer, June 2005.

[27] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994.

[28] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[29] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," *Lecture Notes in Computer Science*, vol. 1996, pp. 253–273, 2001.

[30] S. W. Wilson, "State of XCS classifier system research," *Lecture Notes in Computer Science*, vol. 1813, pp. 63–81, 2000.

[31] M. Butz, D. E. Goldberg, and W. Stolzmann, "New challenges for an ACS: Hard problems and possible solutions," Tech. Rep. 99019, Urbana, IL, 1999.

[32] Martin V. Butz, David E. Goldberg and W. Stolzmann, "Introducing a genetic generalization pressure to the anticipatory classifier system - part 1: Theoretical approach," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (D. Whitley, D. Goldberg,

E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, eds.), (Las Vegas, Nevada, USA), pp. 34–41, Morgan Kaufmann, 10-12 2000.

[33] M. Butz, O. Sigaud, and P. Gérard, eds., *Anticipatory Behavior in Adaptive Learning Systems, Foundations, Theories, and Systems*, vol. 2684 of *Lecture Notes in Computer Science*, Springer, 2003.

[34] B. Widrow and M. Hoff, "Adaptive switching circuits," *IRE WESCON Convention Record*, pp. 96–104, 1960.

[35] W. Stolzmann, M. Butz, J. Hoffmann, and D. E. Goldberg, "First cognitive capabilities in the anticipatory classifier system," pp. 287–296, 2000.

[36] I. P. Pavlov, *Conditioned reflexes.* London: Routledge & Kegan Paul, 1927.

[37] E. C. Tolman and C. H. Honzik, "Insight in rats," *University of California Publications in Psychology*, 1930.

[38] J. P. Seward, "An experimental analysis of latent learning," *Journal of Experimental Psychology*, vol. 39, pp. 177–186, 1949.

[39] W. Stolzmann, "Latent learning in khepera robots with anticipatory classifier systems," in *2nd International Workshop on Learning Classifier Systems* (P. L. Lanzi, W. Stolzmann, and S. W. Wilson, eds.), (Orlando, Florida, USA), pp. 290–297, 13 1999.

[40] G. Chen and T. T. Pham, *Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems.* CRC, November 2000.

[41] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice Hall PTR, 1 ed., May 1995.

[42] L. Zadeh, "Fuzzy sets," *Information Control*, vol. 8, pp. 338–353, 1965.

[43] R. Babuška, *Fuzzy Modeling for Control.* Boston: Kluwer Academic Publishers, 1998.

[44] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, p. 48. Addison-Wesley, 1984.

[45] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian Conference on Robotics and Automation*, ACRA 2005, December 2005.

[46] B. Gerkey, K. Støy, and R. T. Vaughan, "Player robot server," Tech. Rep. IRIS-00-392, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, November 2000.

[47] R. T. Vaughan, "Stage: A multiple robot simulator," Tech. Rep. IRIS-00-394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 2000.

[48] R. A. Brooks, "Intelligence without reason," in *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (R. Myopoulos, John; Reiter, ed.), (Sydney, Australia), pp. 569–595, Morgan Kaufmann, August 1991.

Appendix A

Appendix

A.1 Experiment Setup Parameters

The Fuzzy Anticipatory Learning Classifier Systems includes many tunable parameters which have considerable effect on the outcome of the solution to the problem at hand. However, the majority of these values do not require any modification, and are suitable for a wide range of different problems.

A.1.1 FALCS Parameters

These parameters listed in Table A.1 alter the learning characteristics of the FALCS system. The tuning of parameters is to allow the user to optimize the system for fast learning while avoiding local minima as well as to minimize overtraining, to ensure optimal generality for the given environment, and to ensuring a compact classifier population.

Notation	Description	Value
η_{rl}	Reinforcement learning rate	0.15
η_{rl_κ}	Reinforcement learning rate for Contribution Set	0.05
β	Reinforcement learning rate coefficient for Chain Set	0.8
γ	Reinforcement learning discount factor	0.9
η_{fl}	Function learning rate	0.05
η_{bl}	Function bounds learning rate	$2 \times \eta_{fl}$
θ_r	Inadequacy threshold	0.1
θ_q	Reliability threshold	0.8
θ_e	Experience threshold	25
ϵ	Exploration probability	0.2
χ_{ga}	GA application rate	25
χ_c	Crossover probability	0.7
χ_m	Mutation rate	0.3
i_q	Initial untrained classifier / clone quality	0.5
i_r	Initial untrained classifier / clone reward prediction parameter	1.0
n_{ip}	Initial population size	50
w_C	Initial condition membership function width	0.1
w_E	Initial effect membership function width	0.05
s_C	Initial membership function sharpness modifier	3
s_E	Initial membership function sharpness modifier	10
i_{w_C}	Initial random classifier condition membership function width	0.25
i_{w_E}	Initial random classifier effect membership function width	0.07
min_q	Minimum predictive quality threshold	0.15
min_r	Minimum classifier reward prediction parameter threshold	-20
max_r	Maximum classifier reward	50
g	Action singleton granularity	8
min_κ	Minimum contribution threshold	0.2
α^T	Match acceptance threshold	0.7
Δ_r	Max mergable reward prediction parameter difference	20
Δ_q	Max mergable quality difference	0.1
$n_{[M]}$	Match Set size threshold	10
n_{act}	Acting Set size	5
n_{ch}	Number of chained classifiers	15

Table A.1: Critical FALCS parameters

A.1.2 FALCS Configuration

The options listed in Table A.2 influence the overall configuration and structure of operation. Different environments and objectives may require different behavior and the user may wish to optimize the learning characteristics for the particular application, e.g. wishing to develop an extensive model or just develop a control system to exploit the best action as possible.

Description	Options
Compact population	$\{n_{iter}$, no$\}$
Generalize classifiers during ALP	{yes, no}
Prune bad classifiers	$\{n_{iter}$, no$\}$
Effect "#" equals "no change"	{yes, no}
Utilize best matching classifier	{yes, no}

Table A.2: FALCS configuration switches

A.2 Robot Environments

A set of environments were created to provide an obstacle course for the mobile robot learning simulations. While simplistic in nature, they allow for a full range of different sonar readings to be encountered as well as obstacles with varying profiles that a mobile robot could encounter in a real environment. These are shown in Figures A.1 to A.4.

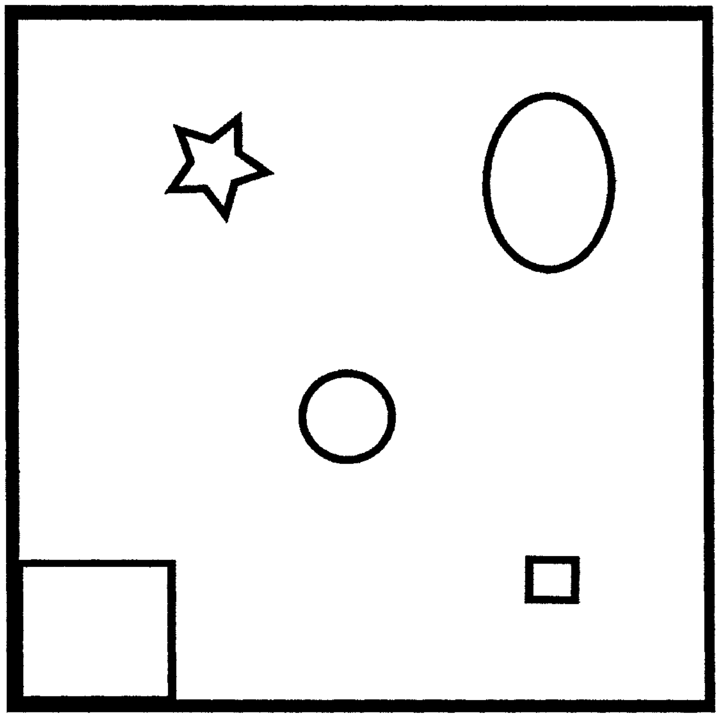


Figure A.1: Environment #1 for robot navigation

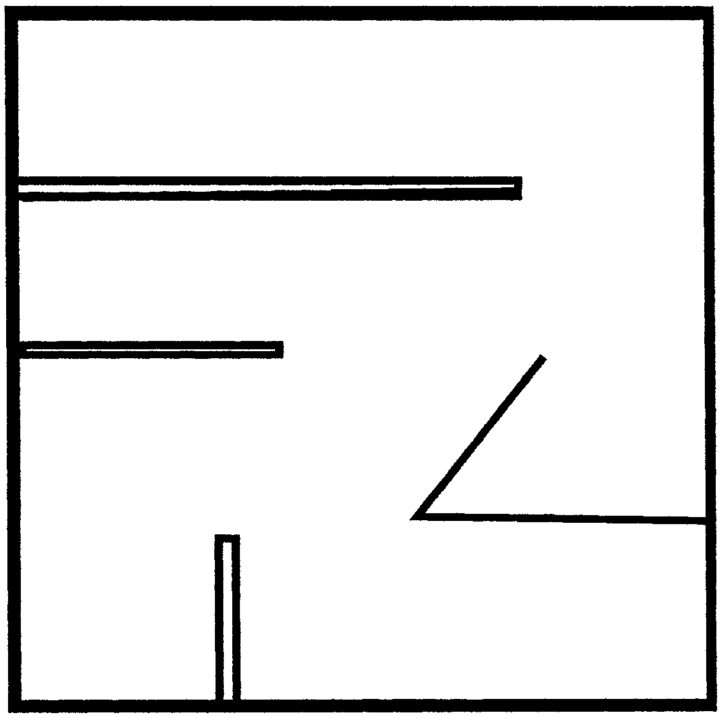


Figure A.2: Environment #2 for robot navigation

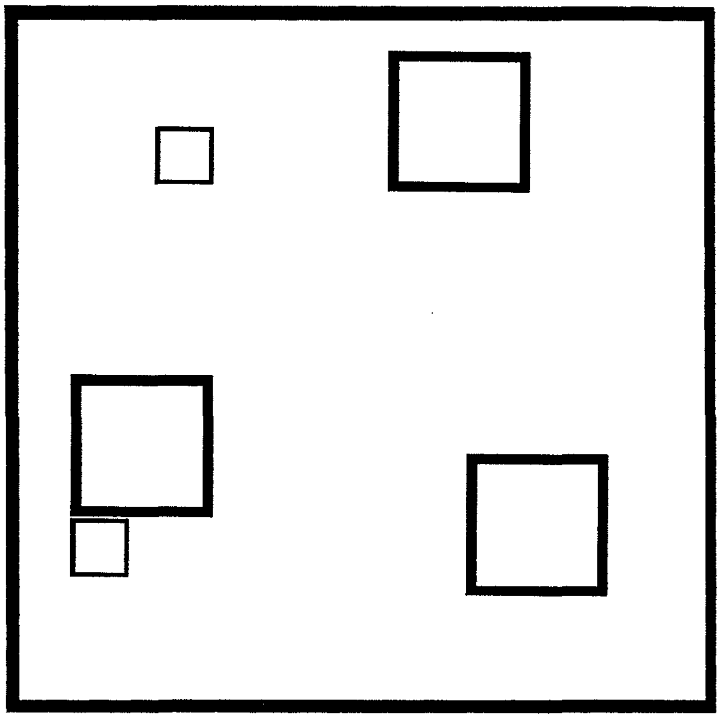


Figure A.3: Environment #3 for robot navigation

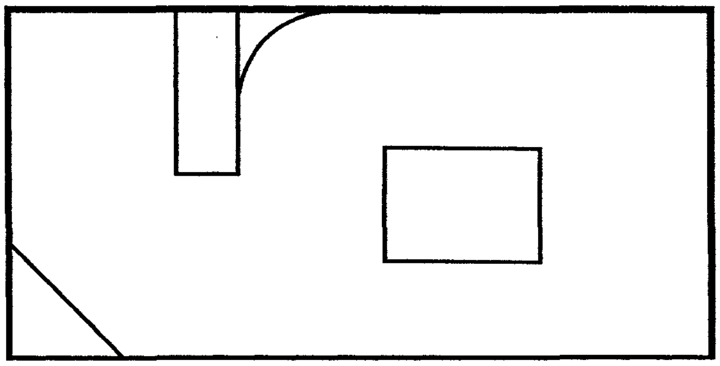


Figure A.4: Environment #4 for robot navigation

A.3 Additional Test Scenarios

This section provides examples of additional results obtained during trial runs of the FALCS-based mobile robot obstacle avoidance system.

A.3.1 Run A

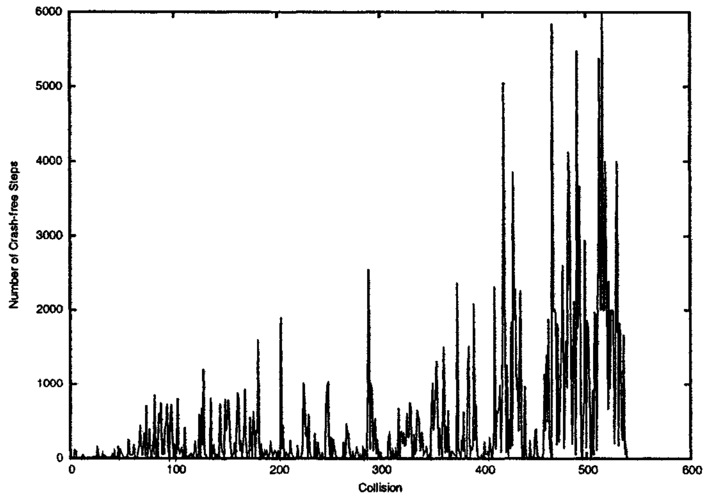


Figure A.5: Number of steps between collisions in Environment # 1

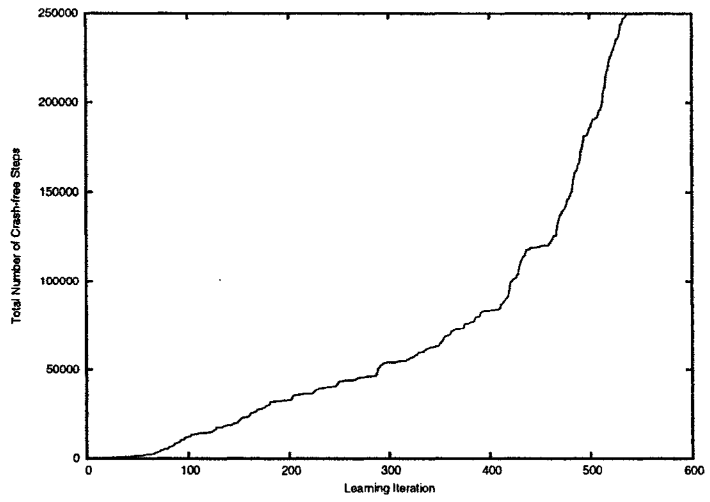


Figure A.6: Total number of crash-free steps in Environment # 1

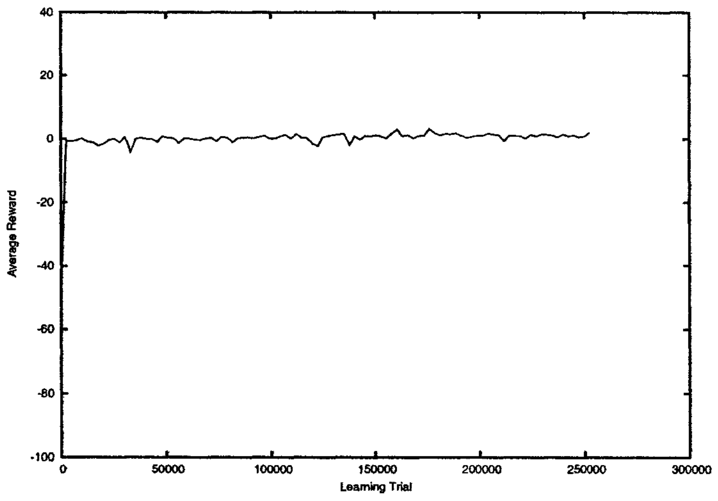


Figure A.7: Average reward given in Environment # 1

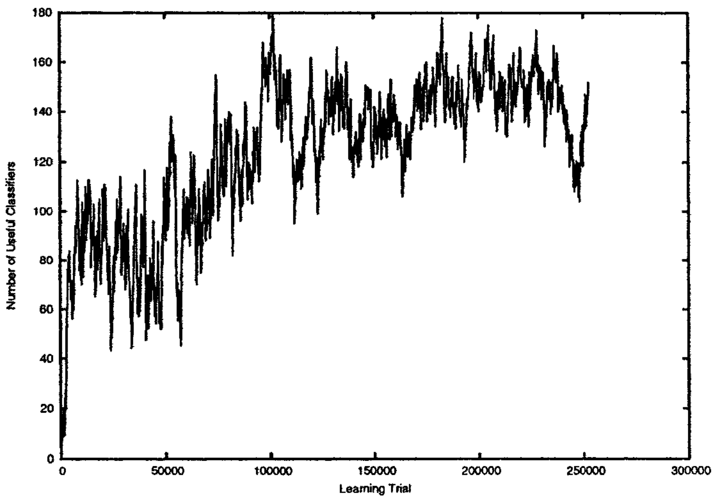


Figure A.8: Number of useful classifiers in Environment # 1

A.3.2 Run B

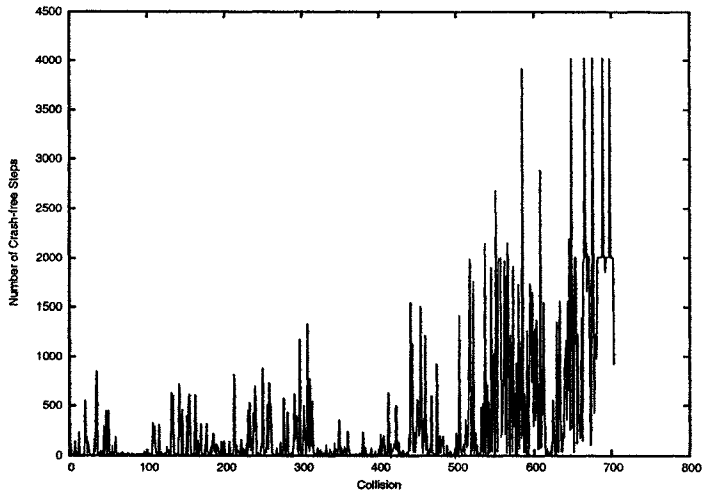


Figure A.9: Number of steps between collisions in Environment # 2

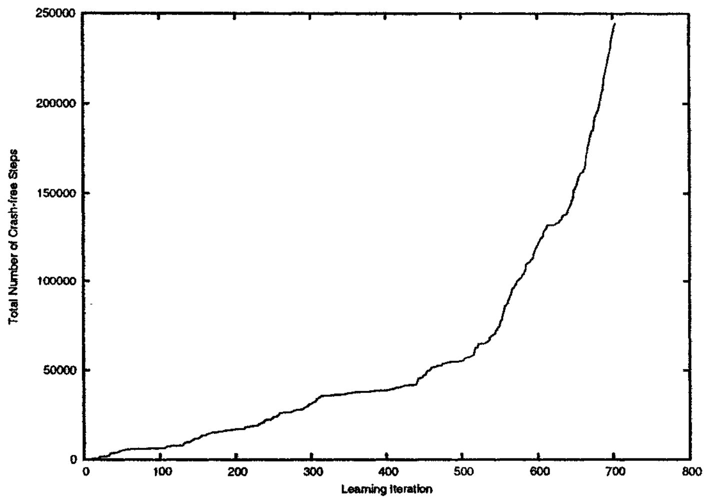


Figure A.10: Total number of crash-free steps in Environment # 2

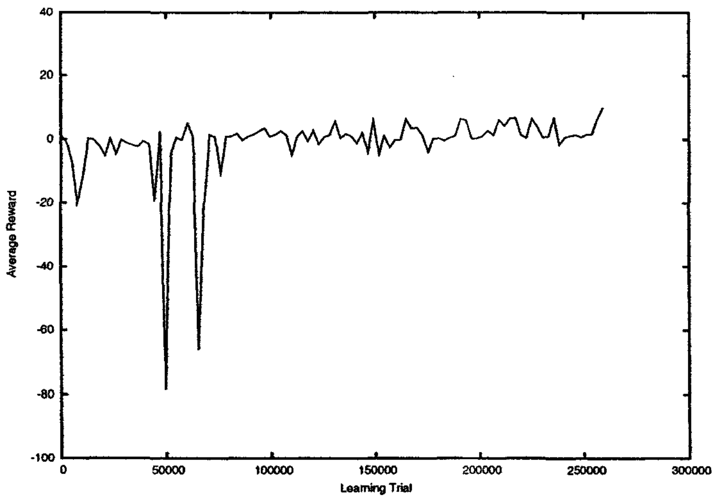


Figure A.11: Average reward given in Environment # 2

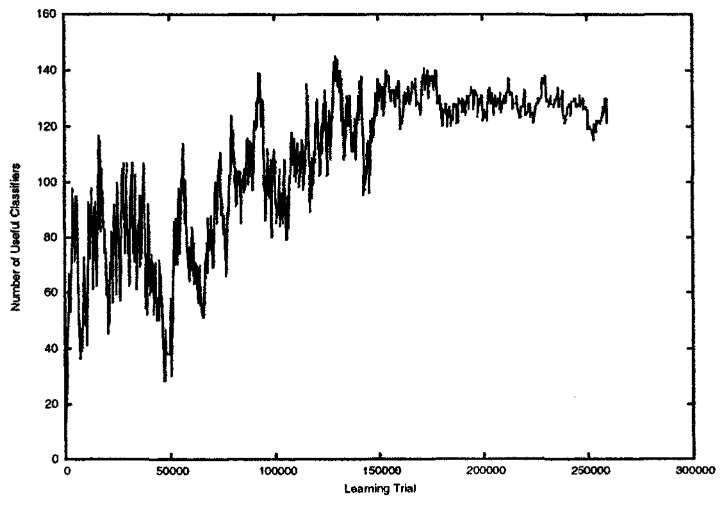


Figure A.12: Number of useful classifiers in Environment # 2

A.3.3 Run C

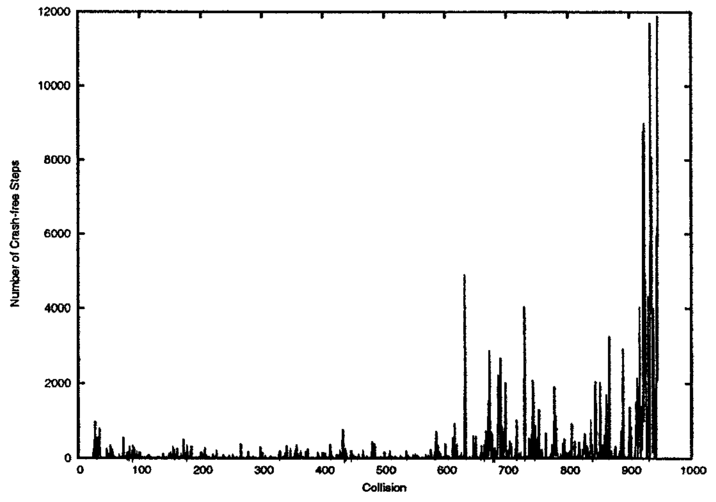


Figure A.13: Number of steps between collisions in Environment # 2

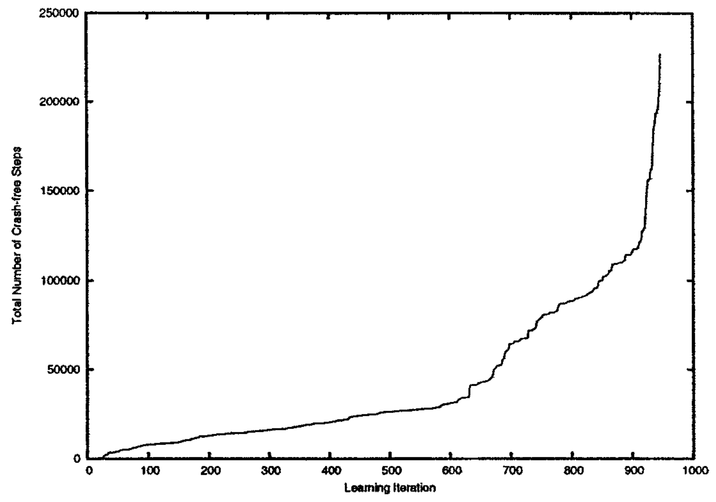


Figure A.14: Total number of crash-free steps in Environment # 2

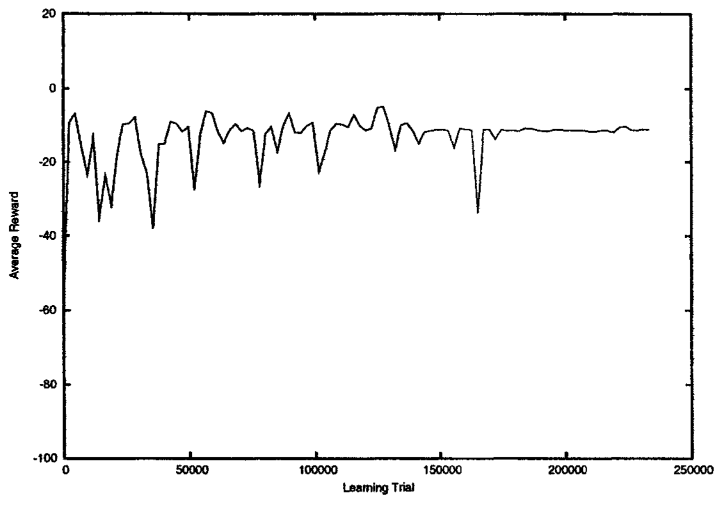


Figure A.15: Average reward given in Environment # 2

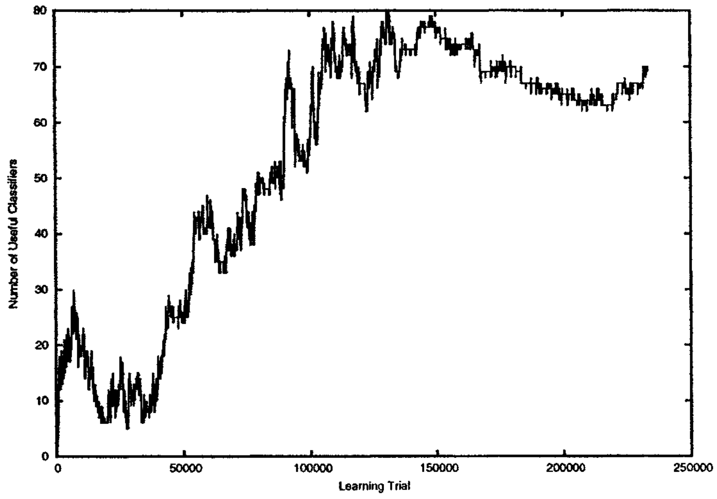


Figure A.16: Number of useful classifiers in Environment # 2

A.3.4 Run D

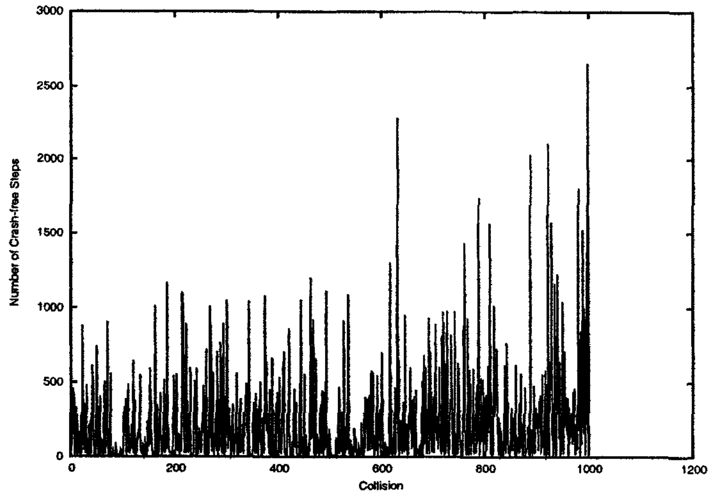


Figure A.17: Number of steps between collisions in Environment # 3

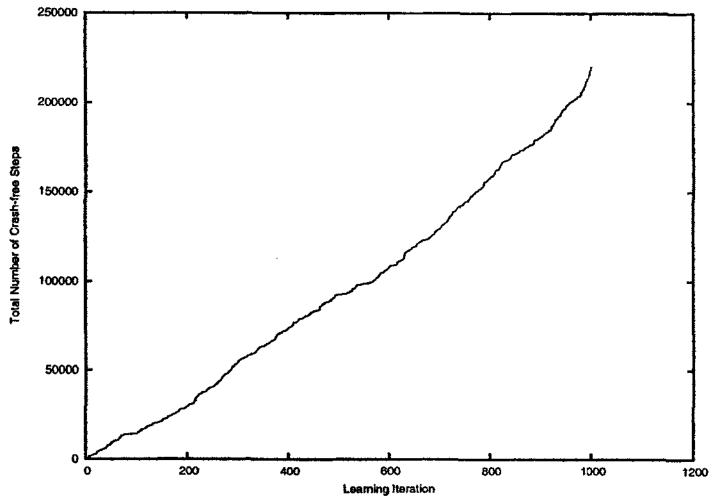


Figure A.18: Total number of crash-free steps in Environment # 3

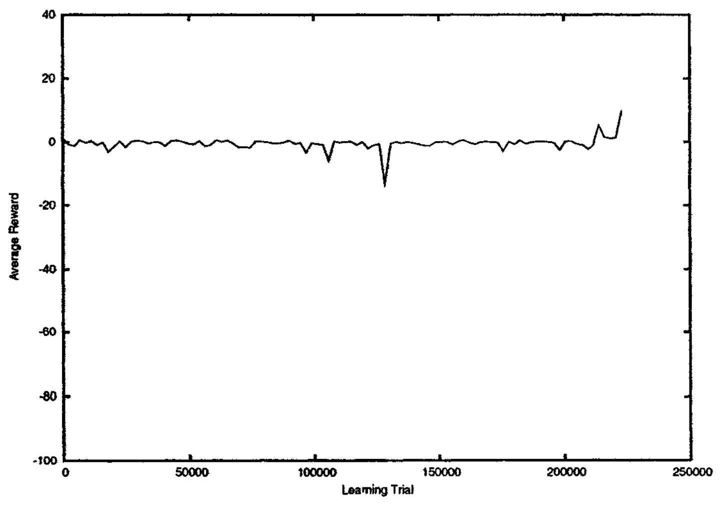


Figure A.19: Average reward given in Environment # 3

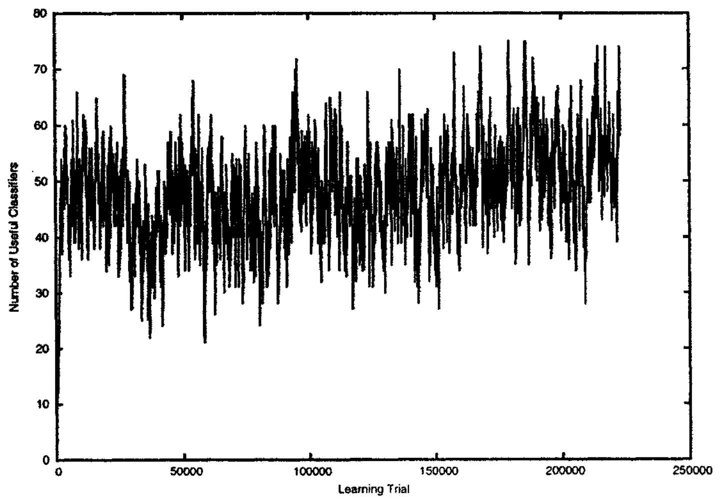


Figure A.20: Number of useful classifiers in Environment # 3

A.3.5 Run E

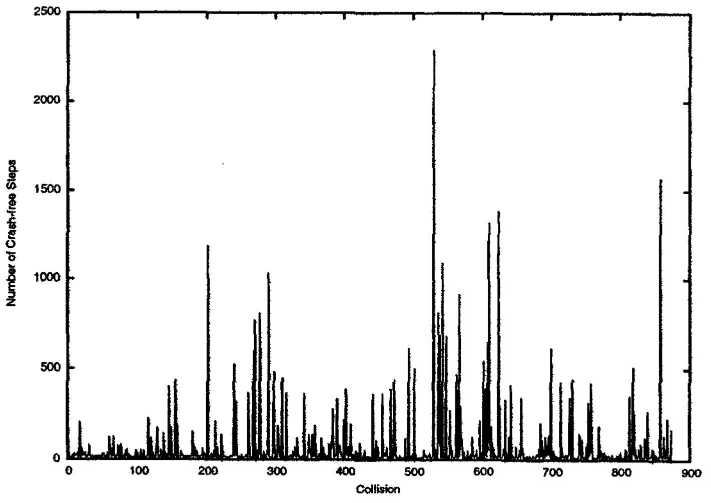


Figure A.21: Number of steps between collisions in Environment # 4

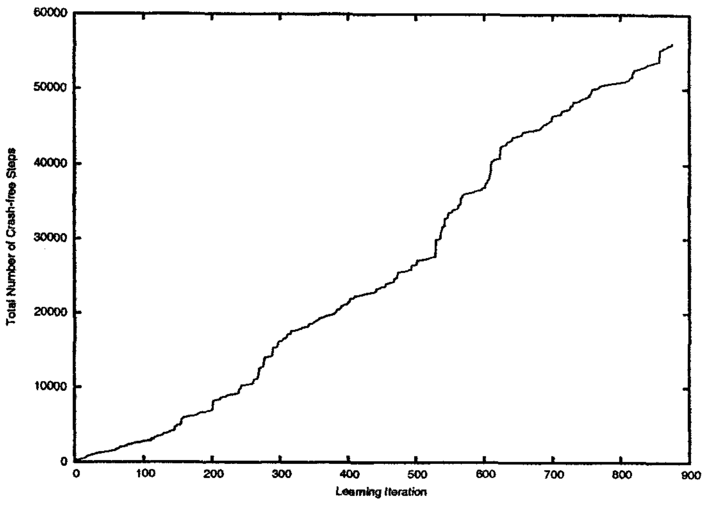


Figure A.22: Total number of crash-free steps in Environment # 4

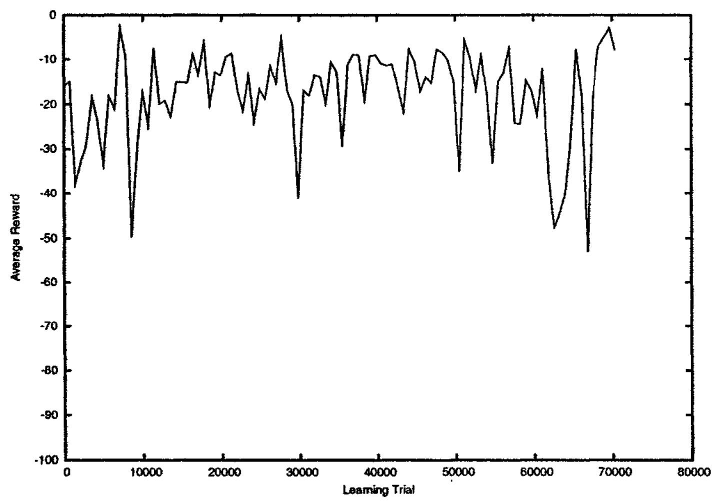


Figure A.23: Average reward given in Environment # 4

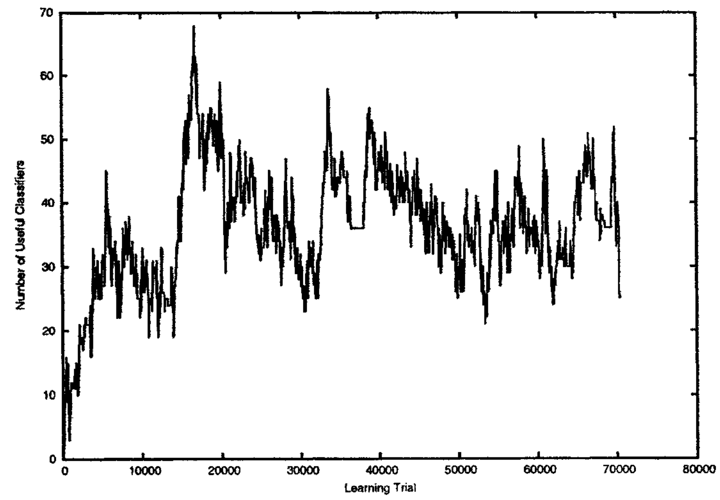


Figure A.24: Number of useful classifiers in Environment # 4

A.3.6 Run F

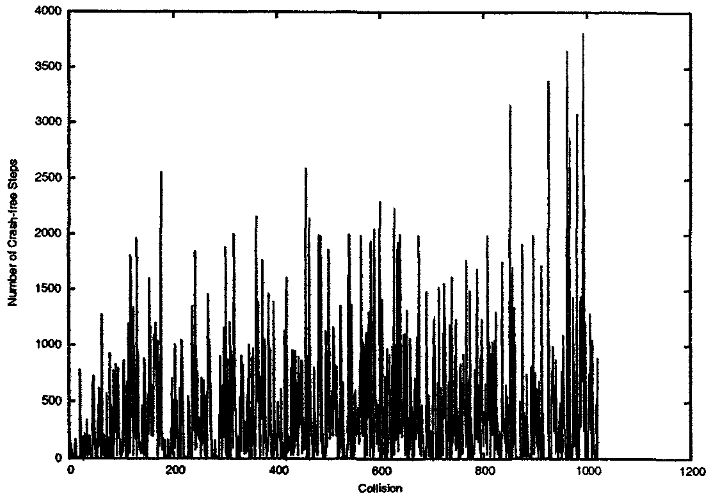


Figure A.25: Number of steps between collisions in learning environment #4

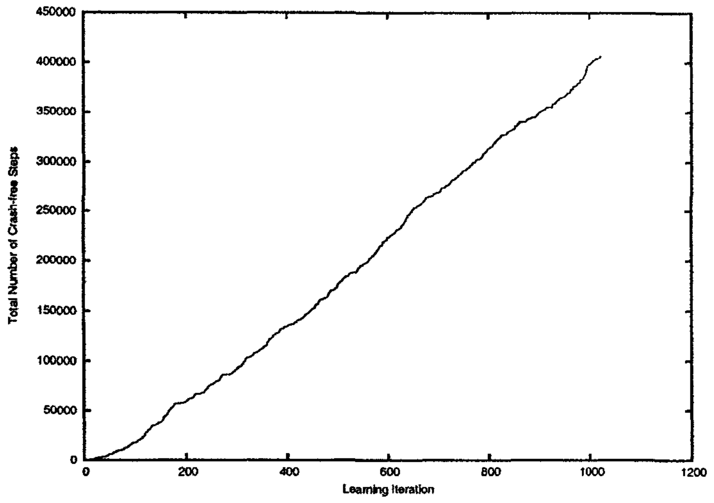


Figure A.26: Total number of crash-free steps in learning environment #4

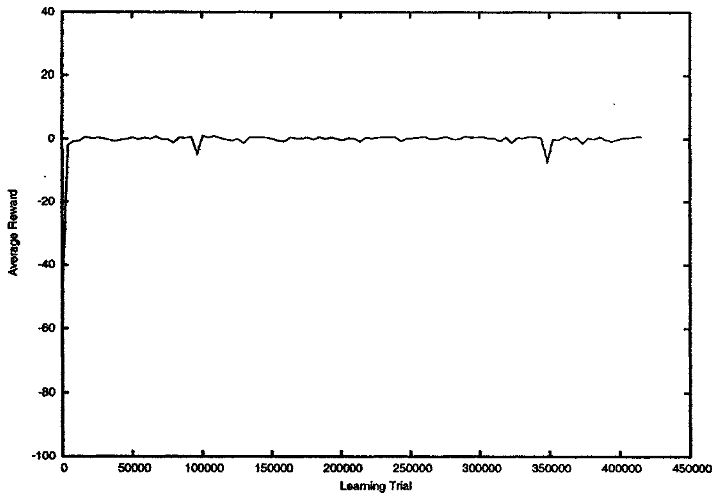


Figure A.27: Average reward given w.r.t. learning iterations in learning environment #4

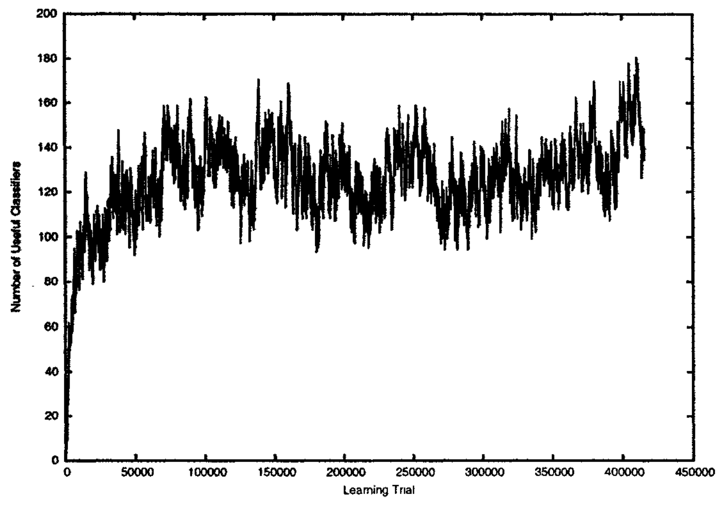


Figure A.28: Number of good classifiers generated in learning environment #4