Umeå University
Department of Computing Science
Master Thesis

Autonomous Learning of Behavioural Mappings in Robotics using Association Rules

Thomas Nyberg
c00tng@cs.umu.se

6th June 2005

Abstract

The reactive paradigm of robotics is based on the relatively simple behaviours found in nature, where complex actions are performed through the interaction between simple behaviours. Each behaviour acts as a direct correspondence between sensory input and the actions performed by the system.

Association rules will be used to provide that mapping between stimuli and response. The rule discovery process requires that the input space is discretized, a requirement that offers two distinct problems; determining the optimal number of discrete partitions and the exact extent of each partition. A simple iterative approach to the first problem will be proposed, and genetic algorithms will be used to find the optimal set of partitions. This will allow for a system to autonomously create the set of association rules and thus learn the behavioural mapping.

As will be shown, the combined use of genetic algorithms and the iterative algorithm for determining the optimal number of discrete partitions is sufficient for the system to learn the behaviour.

By using the set of association rules to provide the stimuli/response mapping there is a large chance of multiple rules matching any given set of input data. To aggregate the resulting action of all matching rules, a method based on principles traditionally used in fuzzy logic controllers will be proposed. This method will be evaluated against a simpler majority vote-algorithm to determine which approach is better.

The results of the comparison strongly suggests that the aggregation of rules using fuzzy logic concepts, is not sufficiently superior to justify its increased complexity compared to the simpler majority vote-algorithm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotics has become an important field with many applications, ranging from industrial robots building cars to autonomous vacuum cleaners in our homes. The versatility of robots are hampered by the complexity required to assign new tasks for a robot to perform. Teaching an autonomous lawn mower not to run over a particular flower bed should not require the assistance of an expert programmer.

To facilitate this the "Programming by Demonstration" methodology was proposed, where the goal is to provide rather simple means for a teacher to instruct a robot to perform certain actions and tasks. The system responds by recording the actions of the teacher, analyses it and should then be capable of performing the relevant task.[1]

We will borrow concepts traditionally used in data mining applications, to construct a system that is capable of autonomously learning tasks demonstrated to it by a teacher using a remote control. The learning process works by extracting association rules from the data recorded when the system "observed" the teacher. This rule base then provides a mapping between the sensor input and the action the robot should perform as a response.[2]

To extract the association rules, the raw sensor data recorded during the demonstration has to be discretized into a finite number of partitions. These partitions corresponds to different distances, e.g. "long distance" or "very short distance". In order for the system to be able to autonomously learn the behaviour, it has to be capable of determining how many such partitions are required, together with the borders defining the partitions themselves. Solving this problem will be done by using genetic algorithms, since they perform well with problems of high dimensionality.

Since the association rules are created to be quite general, there is a high probability of multiple rules matching a given set of sensor readings. In an attempt at solving this problem (or at least to ameliorate it), concepts from fuzzy logic and

[1] Dillman et al. [8] provides a general overview of this technique.

[2] In this thesis we shall limit ourselves to only consider purely reactive behaviours, e.g. "follow wall" or "avoid obstacle".

fuzzy set theory will be used to allow for non-crisp partitions. This method will then be evaluated and compared against alternative methods proposed by Hellström [14].

1.1 Related Work

The basic approach of learning the underlying behaviour by extracting association rules from recorded data was originally proposed by Hellström [14]. In his work it was assumed that the partitioning scheme[3] was already known, which introduced a limit to the flexibility of the system. In reality only manually coded controllers were plausible (see Section 5.4).

We will extend his work by allowing the system to find and determine the partitioning scheme autonomously, something which allows for remote controlled demonstration of behaviours. Comparing the possibilities offered by this approach to those from manually coded controllers, should make the gains obvious.

Hellström outlined a solution to the multiple rules matching problem, by using a majority vote between all the matching rules. His method will be compared to our method (based on fuzzy logic) to see which approach that is more powerful.

1.2 Outline

This thesis is organised into two parts, the first pertaining to the theoretical framework underlying the concepts used in the second part.

In order to understand the basic approach, a short overview of association rules and their related concepts is given in Chapter 2. The association rules are dependant upon discretized partitions, and the finding of these partitions is done using genetic algorithms. The necessary information regarding genetic algorithms and how they work can be found in Chapter 3. The various aspects of fuzzy systems (a collective name for fuzzy logic, fuzzy set theory and fuzzy logic controllers) are covered in detail in Chapter 4.

The second part of the thesis, Chapter 5, concerns the various aspects of autonomously learning the behavioural mappings. This includes finding the optimal number of partitions, finding the optimal set of borders that defines the partitions and lastly finding and evaluating the fuzzy approach to handling the matching of multiple rules. A small summary of the notation used in Chapter 5 can be found in Appendix A. The conclusions that can be drawn from the results obtained in the previous chapters are gathered in Chapter 6.

[3] The number of partitions and the borders that define the partitions are collectively referred to as a partitioning scheme.

Chapter 2

Association Rules

Association rules are often used in data mining applications, where the system tries to extract information from huge data sets. In the case of a supermarket these data sets often consists of the items bought by its customers at different times (i.e. the *basket* data). The concept was originally developed as a means to determine trends and used for setting prices, discounts and how to place shelves (e.g. which items should be placed next to each other). Knowing which items are often bought together can be useful, an example of that could be "60 percent of all customers who bought bread, also bought cheese and butter at the same time". Association rules are capable of discovering these patterns automatically, which makes them very useful [1].

The association rule consists of two components, the antecedent and the consequent. In our example the antecedent consists of "bread" and the consequent is made up of "cheese and butter". An important concept is the notion of confidence, which in this case would be "60 percent". This means that in 60 percent of the cases when someone bought bread they also bought the other items (cheese and butter).

An association rule can be defined more formally as follows [2, 26]. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ be the universe of discourse, where each member is called an item. Let $\mathcal{D}$ be a set of transactions where each transaction $T \subseteq \mathcal{I}$ is a set of items. Then an association rule is an implication $X \Rightarrow Y$, where $X \subset \mathcal{I}$ is the antecedent, the consequent $Y \subset \mathcal{I}$ and $X \cap Y = \emptyset$ (i.e. no item in the consequent may appear in the antecedent and vice versa).

From this follows several important concepts and definitions, which are used for several different purposes. Some of these provides different measures as to the quality of a rule.

The *cover* of a subset of items X is defined to be the number of transactions containing all the items of X divided by the total number of transactions, or formally:

$$\mathit{cover}(X) = \frac{|\{T \mid T \in \mathcal{D} \wedge X \subseteq T\}|}{|\mathcal{D}|}.$$

Building on this definition, the *coverage* of an association rule $X \Rightarrow Y$ becomes:

$$coverage\,(X \Rightarrow Y) = cover\,(X)\,.$$

This definition simply says that the *coverage* of a rule is the ratio of occurrences of its antecedent.

A similar measure is called $support$[1] and measures the proportion of transaction that contains both the antecedent and the consequent of a rule.

$$support\,(X \Rightarrow Y) = cover\,(X \cup Y)$$

Perhaps one of the most useful measures, is the confidence or strength of a rule (in the example used the confidence was 60 percent).[2] The confidence is the ratio of transactions containing the antecedent that also contains the consequent, or (less confusing):

$$strength\,(X \Rightarrow Y) = \frac{support\,(X \Rightarrow Y)}{coverage\,(X \Rightarrow Y)}. \tag{2.1}$$

One of the classical measures of the quality of a rule is the lift. This measure takes into account the (in)dependence of the items in the rule. If the antecedent and consequent are independent they may still occur together often enough (by chance) that rules are generated with a high strength. These rules are of low quality since they do not contain any important information. The lift calculates the ratio between the support of the rule if the antecedent and consequent were dependant and the support if they actually were *independent*. This can be formulated as:

$$lift\,(X \Rightarrow Y) = \frac{support\,(X \Rightarrow Y)}{cover\,(X) \cdot cover\,(Y)}.$$

Whenever the $lift$ deviates from 1.0, it indicates that a significant dependency exists between the antecedent and consequent.

The final function which shall be mentioned is the notion of leverage (also known as the weighted relative accuracy in a rewritten form), which yields the number of additional transactions found compared to those expected if the antecedent and consequent were assumed to be independent.

$$leverage\,(X \Rightarrow Y) = support\,(X \Rightarrow Y) - cover\,(X) \cdot cover\,(Y)\,.$$

The following example will hopefully help to clarify these definitions and the terminology used. Let $\mathcal{I} = \{apple,\ orange,\ pear,\ banana\}$, and the basket data consists of:

$$\begin{aligned} \mathcal{D} \quad = \quad & \{\{apple,\ orange\}, \{apple,\ pear\}, \\ & \{apple,\ orange,\ banana\}, \{orange,\ banana\}\}. \end{aligned}$$

[1] In this thesis there might be a degree of confusion surrounding this term, since it is also introduced in fuzzy set theory (Chapter 4) albeit with a different meaning.

[2] This measure is the one mostly used within this thesis, for example acting as a threshold value to filter out low quality rules.

Consider the rule $apple \Rightarrow orange$, then the different measures is calculated to be (using the previous definitions):

$$\begin{aligned} coverage(apple \Rightarrow orange) &= 0.75 \\ support(apple \Rightarrow orange) &= 0.5 \\ strength(apple \Rightarrow orange) &\approx 0.67 \\ lift(apple \Rightarrow orange) &\approx 0.89 \\ leverage(apple \Rightarrow orange) &\approx -0.06. \end{aligned}$$

2.1 Generating Rules

In the preceding example, the rule that was considered was assumed to be known beforehand. Finding these rule may appear difficult, but it can be done automatically using several different algorithms. One of the simpler and popular algorithms is the Apriori algorithm designed by Agrawal et al. [2],[3] others make use of more advanced methods such as the Opus algorithm [25].

Even though there are many different algorithms to mine the association rules from databases, it should be pointed out that they all yield the same rule base. The main difference between different algorithms is their efficiency, i.e. the time required to create the rule base. Furthermore, the mining of association rules is not to be confused with other forms of rules (such as those commonly used in classification) [11].

When association rules are created they are constrained by two factors [1]:

1. Syntactic constraints: This limits the appearances of the rules, determining factors such as the number of items allowed to occur in either (or both) of the antecedent or consequent. It also determines which items may appear in which part of the rule, or which items that may appear together. An example of a syntactic constraint might be that "oranges may only appear in the consequent".

2. Support constraints: The support corresponds to the statistical significance of a rule. This constraint is used such that only rules above a certain threshold are deemed interesting, and others are ignored. Most commonly used during the actual rule generation, where only the rules above the threshold are considered.

All of the algorithms used to generate the association rules work by the same main principles. First they generate a possible rule that complies with the syntactic constraints, then this rule is compared against the support constraints. If the support of the rule is above the threshold value it is kept, otherwise ignored. When no more

[3]The software used (aptly named Apriori) in Chapter 5 to create the association rules uses the Apriori algorithm.

rules can be generated, the algorithm is finished. This rather simple method works, the trick is to do this in as an efficient manner as possible (remember that some databases might span billions of transactions).

To increase the efficiency of the rule generation, there is one important property which is used. Namely, that the support of a rule can never be higher than the support of any subset of that rule. This means that the rule $apple \wedge banana \Rightarrow orange$ can never have a support higher than $apple \Rightarrow orange$. So, if the latter rule fails to conform to the support constraint, then the former rule is guaranteed to also fail. This property is the main driving force behind the Apriori algorithm [2].

Chapter 3

Genetic Algorithms

Genetic algorithms are a part of a much wider field called evolutionary computation, which includes other areas such as evolutionary programming. The idea behind evolutionary computation comes originally from the biological equivalent, where a group of individuals are competing for resources against each other. Individuals with a high fitness will have a better chance at producing offspring than those that are unfit. Since the traits of the parents are passed on to their offspring, the children in turn will have a higher probability of survival than the children of less fit parents. In essence, this is "survival of the fittest".

It should be reasonably clear that the process of evolution is analogous to an optimisation algorithm. Each generation will force the individuals to become more and more adapted to the specific environment, and thus becoming better at solving the particular problems within that domain. Those individuals which are unable to adapt and compete will be unable to produce offspring, this creates a selective pressure against unfit individuals.

Genetic algorithms are based upon the same mechanisms that drives biological evolution. This includes selection, reproduction, mutation and competition. This approach to optimisation has several advantages compared to classical techniques (e.g. gradient descent or deterministic hill climbing), in that they are better at handling nonlinear problems with chaotic or stochastic properties. Genetic algorithms also work well in high dimensionality problems (i.e. a lot of dependant variables interacting in complex ways).

The terminology used when dealing with genetic algorithms and evolutionary computation borrows heavily from biology. For genetic algorithms to be of any use in solving a problem, the problem must be possible to encode in a suitable form. The variables describing the problem and possible solutions are combined to form a chromosome, each part of the chromosome that corresponds to a variable is often referred to as an allele. The exact appearance of the chromosome is dependant on the representation chosen. In the case of a binary representation the chromosome consists of a binary string and each allele is a substring of binary digits.

The population consists of a number of individuals, and each individual corre-

sponds to a chromosome. Through the mechanisms of evolution the population will gradually converge towards the global optimum of the problem. This algorithm is typically made up of the following steps [5]:

1. Selection: A number of individuals are chosen from the population, where individuals with a high fitness has a higher probability of being chosen.

2. Recombination: A certain number of those chosen by the selection algorithm, are submitted to recombination. Where new individuals are produced by combining aspects of the old individuals.

3. Mutation: Each part of the chromosomes have a certain low probability of being subjected to random changes.

The details surrounding these concepts will be covered in the following sections.

3.1 Representation

The choice of representation should not only be based on the problem in question, but also take into account the search method that will be used. In most types of optimisation algorithms the problems and solutions are represented by real valued vectors. However, in genetic algorithms the most popular representation has been (and still remains) the binary representation, where the proposed solution is encoded as a string of binary digits.

Encoding the problem as a binary string suggests that only discrete problems may be considered. This limitation of the binary representation may be worked around though, by mapping the range of the respective real variables onto the range of the binary equivalents. That is, consider a variable that may assume values between a and b, then 000_2 maps to a and 111_2 maps to b. The intermediate values of the real variable can then be mapped to the other possible values of the three digit binary code. It should be obvious that there is a loss of precision corresponding to the length of the binary string [5].

Besides the possible problems of precision that may arise by using a binary representation, there is also the so called “Hamming cliff problem”. This is caused by the fact that two consecutive numbers may differ in many places, e.g. the number three is coded in binary as 011_2 and the following number four would then be 100_2 thus differing in three places (the two representation is said to have a Hamming distance of three). This lessens the strength of the mutation-operator since the probability that making a small change in the string would cause a large change in the output becomes quite high [7].

The solution to this problem is to use a Gray code. This is a transformation of the binary code which ensures that two consecutive values always have a Hamming distance of one. In the preceding example the two numbers becomes $3 \rightarrow 010_{Gray}$ and $4 \rightarrow 110_{Gray}$, which now differ in only one place.

Besides the binary representation there are several other possible representations, some of which are highly problem dependant (such as the finite-state and parse trees). Others are more general, including using real values directly [5].

3.2 Selection

Selection is one of the important concepts of genetic algorithms, in fact the entire field of evolutionary computation, used to provide the driving force behind the mechanism. The purpose of selection is to favour individuals with a high fitness score, allowing them to reproduce and create more children than compared to those individuals with a lower fitness score. Individuals with a high fitness should have a higher probability of producing offspring, thus ensuring that the population strives towards the optimum solution.

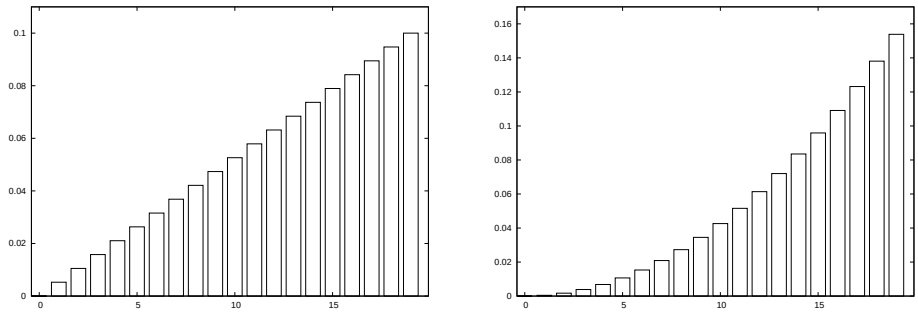
If only those individuals with the highest fitness is allowed to reproduce, there is a risk that the population converges to a local optimum instead of the global. To combat this problem, it is important that there is a chance for those individuals that are less fit to reproduce. This ensures a higher genetic diversity in the population, which is especially important when the population size is small since otherwise the offspring of the most fit individual will dominate the population after only a few generations.

The objective function defined by the user is the function the system is trying to optimise (i.e. minimise or maximise). It is important that this function is chosen well and that it reflects the problem. Genetic algorithms have a strong tendency to exploit "weaknesses" in the objective function, causing the population to converge to non-optimal solutions. This problem is especially prevalent when the objective function only provides an approximation to the *real* function, e.g. when the real function is too complex or unknown.

The fitness function $\Phi : A_x \rightarrow \mathbb{R}^+$, where A_x is the objective variable space, maps the objective function into a non-negative real value. This is required by many different selection and scaling algorithms, even though some such as the rank scaling works quite well with negative fitness functions (see Section 3.2.1).

The selection algorithms can be divided into two main classes, those that are deterministic and those that are stochastic in nature. These two groups have different strengths and weaknesses. Many stochastic selection mechanisms suffers from a very high variance in the number of offspring assigned to each individual, while deterministic selection tends to favour individuals with high fitness too much. Thus lowering the genetic diversity of the population.

For this thesis, an algorithm called stochastic universal sampling (SUS) will be used. This algorithm is as the name suggests, stochastic in nature. However, it does not suffer from the problem of high variance mentioned earlier. For this reason SUS is one of the more commonly used selection algorithms. The low variance comes from the fact that SUS only requires one draw from the population to determine the number of offspring to assign to each individual [5].



(a) (b)

Figure 3.1: (a) The resulting probability of selection when the population is submitted to linear rank scaling. (b) Nonlinear rank scaling is applied, favouring more heavily those individuals with high rank.

3.2.1 Scaling

One recurrent problem with genetic algorithms arises when the population is very close to converging to one solution. When this happens all the individuals of the population will have a very similar fitness score, and the likelihood of producing offspring becomes equal for all individuals. The selection pressure is therefore removed, and the algorithm will take a very long time to converge.

By scaling the fitness of the population this problem can be lessened or removed. The idea is to base the selection not on the fitness score itself, but rather on a function of the fitness score. By choosing the scaling function appropriately, it is possible to continue to provide selective pressure.

There are many different forms of scaling, some are based on the mean fitness of the population as it changes over time. Others attempts to scale by using the difference between the best and worst individual of the population. The method used in this thesis is called rank scaling, and is independent of the actual fitness of each individual. Instead, it only looks at the relative rank of each individual compared to the others within the population. An individual with a high rank corresponds to an individual with a high fitness and vice versa. Figure 3.1 shows both a linear and nonlinear version of rank scaling for a population size of 20. From this figure it should become evident that individuals with a high fitness have a higher probability of reproducing, which is exactly the purpose of scaling and selection [5].

3.3 Operators

A population consists of a number of individuals, where each individual consists of a number of variables encoded with a specific representation (even though hybrids are possible, for example mixing binary and real numbers). For most representa-

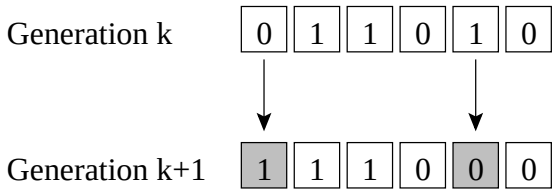


Figure 3.2: Mutation is a simple operator which works by inverting some of the bits, i.e. a one becomes a zero and vice versa.

tions one or both of the two basic operators mutation and recombination is defined. Just because the operators exists and provides similar functionality, does not mean that they work in the exact same way. In fact the inner mechanism for each operator is highly dependant upon the representation, e.g. mutation works by inverting bits in the binary representation, and it should be self evident that this method does not work for real numbers.

Below follows a description of the two main operators (besides the selection mechanism) used in genetic algorithms; mutation and recombination, with the focus on their binary versions.

3.3.1 Mutation

The mutation operator is one of the two basic operations defined for most representations. A new individual is produced from an old, by changing individual bits within the bit string making up the individual (see Figure 3.2). Each bit has a certain low probability of being altered, independent of the fate of the other bits within the string. Each bit a_i is altered according to the equation:

$$a_i' = \begin{cases} a_i & u > p_m \\ 1 - a_i & u \leq p_m \end{cases}$$

where $u \sim U([0,1))$ (u is drawn from a uniform random distribution). The mutation rate p_m governs the probability of a bit being affected. This parameter has a theoretical lower bound of $p_m = \frac{1}{l}$ where l is the length of the string. Instead of manually setting this parameter, there are methods by which it can be determined automatically through self-adaptation (where this parameter becomes part of the optimisation problem) [5].

The mutation operator is most often used together with recombination, where mutation is used mainly to provide genetic diversity to the population and recombination acts as the driving force. This does not hold for all problem domains, it has been shown that using only the mutation operator the resulting performance is on par with, or better than using a combination of mutation and recombination. Thus the mutation operator is an important mechanism for genetic algorithms [5, 10].

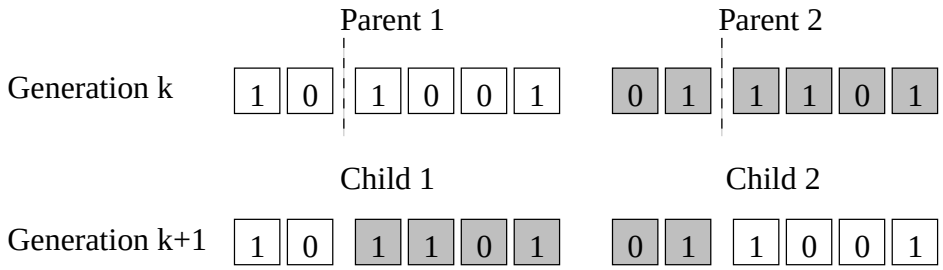


Figure 3.3: One-point crossover works by splitting the parents into two disjunct parts. The corresponding parts switches places to generate the offspring. The children now consists of parts of both of their parents.

3.3.2 Recombination

Recombination (or crossover) is a binary operator that combines the information contained in two individuals, the parents, to produce two new individuals, called the offspring or children. Assuming that the chromosome is represented by a string of arbitrary alphabet (e.g. binary or real numbers), then the recombination is performed by exchanging substrings between the two parents. More exotic representations, such as permutations or tree-structures, requires different methods (and the semantics may be different too). As before, it is assumed that the variables encoded in the chromosome uses a binary representation.

The recombination procedure works in three distinct steps.

1. Two individuals are chosen randomly from the set of individuals chosen for reproduction by the selection operator (see Section 3.2 for more details about how selection works). These two individuals are called the parents.

2. One or more positions along the chromosome are chosen as breakpoints, also called crossover points. At these crossover points the chromosome will be split forming the substrings. It is important to note that the same positions are chosen for both parents.

3. Finally, the substrings are exchanged between the two parents forming the offspring, consisting of two new individuals.

The number of individuals chosen for recombination is controlled by a parameter p_c, the crossover rate. A high value means that there is a high probability for each individual to undergo recombination (i.e. many children will be created).

Recombination exists in many versions, differing mainly in the number of crossover points they make use of. Originally the one-point crossover was used, as seen in Figure 3.3 it uses one breakpoint to split the two strings. This method has been extended to form two-point and N-point crossover operators (working in the obvious way).

A more interesting variant is the uniform crossover operator, which does not use breakpoints but rather determines if an exchange of bits should take place on a bit-by-bit basis. For each position in the parent string, there is a certain probability p_x that the bit will be exchanged with the other parent. Typically, $p_x = 0.5$ making all possible combinations equally likely [5].

Which of these recombination algorithms performs best is highly problem dependant. Determining which performs best therefore requires some experimentations. In this thesis the two-point crossover will be used, mainly because of its simple nature. The uniform crossover requires more computations and therefore slows down the time (measured in seconds) for convergence.

Chapter 4

Fuzzy Systems

The term fuzzy systems is often used to collectively refer to areas such as fuzzy set theory, fuzzy logic and fuzzy control. Fuzzy systems have been deployed in a multitude of applications, ranging from home appliances and cars to controlling trains and heavy machinery. The principles of fuzzy systems have been applied to (among others) expert systems, databases, medical applications and economics.

Traditional approaches often have difficulties when dealing with incomplete or inaccurate information. When dealing with complex problems approximations of the underlying models are often required (e.g. using a linear approximation for a nonlinear problem). In these cases fuzzy systems often outperforms their non-fuzzy counterparts [21].

Fuzzy systems allows rules to be described in relative terms such as “fast” or “slightly below”, which causes rules to more closely resemble natural language [27]. Because of this, expert knowledge is more easily encoded into a rule base.

Unfortunately, there are some shortcomings associated with fuzzy systems. They lack both learning capabilities and memory, one of the reason why hybrid systems have been developed (for instance combining neural networks and fuzzy logic). Determining and tuning the parameters and rules of a fuzzy controller is difficult, so genetic algorithms are often used for this purpose (see [23] for an example). Perhaps its greatest weakness is the question of stability, there are no guarantees that a fuzzy system will not descend into a chaotic state. Although experience suggest that the risk for such behaviour is very small [21].

Fuzzy logic and fuzzy sets have the same relationship and similarities as ordinary logic and set theory.

4.1 Fuzzy Sets

Classical set theory concerns the notion of a crisp set. In such a set every element is either a member or a nonmber, and determining which is the case is always a more or less simple matter (at least for well-defined sets). The characteristic function

Table 4.1: This table represents a possible membership function for the set of old people. It is clear that the different membership grades are very subjective (example taken from [17]).

Age	5	10	20	30	40	50	60	70	80
Old	0.0	0.0	0.1	0.2	0.4	0.6	0.8	1.0	1.0
Young	1.0	1.0	0.8	0.5	0.2	0.1	0.0	0.0	0.0

$\mu_A(x)$ assigns a value to any element such that:[1]

$$\mu_A(x) = \begin{cases} 1 & \text{if and only if } x \in A \\ 0 & \text{if and only if } x \notin A. \end{cases} \tag{4.1}$$

Crisp sets are a powerful and useful tool in many areas, but certain limitations still exist. Dealing with sets that are not well-defined introduces difficulties. For example, consider the set of old people. To create this set it would be necessary to partition all people into two sets, those below and those above a certain age. Deciding such a crisp boundary will almost certainly be quite difficult (a child might think that anyone older than thirty is old, while an adult would disagree).

To more efficiently deal with this problem, the idea of a crisp set can be extended into a fuzzy set theory (classical sets are in fact a subset of fuzzy sets). Instead of the binary relationship where elements are either members or nonmembers (but not both), fuzzy sets assigns a degree of membership. This can be done by expanding the notion of the characteristic function in Equation 4.1 from the mapping $\mu_A : \mathcal{U} \rightarrow \{0,1\}$ to the more general case $\mu_A : \mathcal{U} \rightarrow [0,1]$. This means that each element belong to a given set to a certain degree (with the two extremes of zero being no membership and one being full membership) [17].[2]

Using this membership function it becomes easier to deal with the aforementioned problem of determining who is old or not. Instead of saying that a person of forty is old, that person might belong to the set of old people with a membership grade of 0.4 (a more detailed example is found in Table 4.1). Another example might be someone at the age of ninety having full membership. It is easy to see the increased expressiveness offered by fuzzy sets over the more traditional crisp sets.

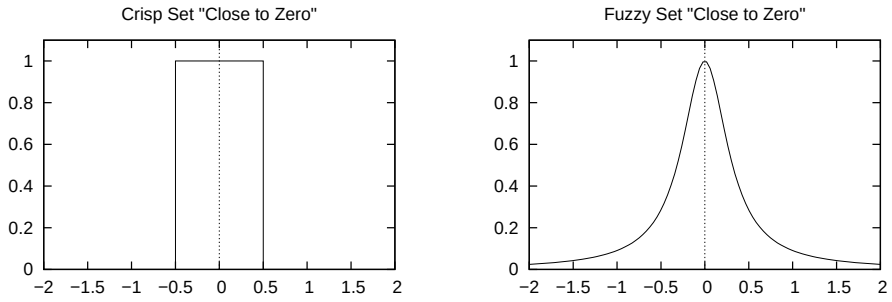
The difference between crisp and fuzzy membership functions is shown in Figure 4.1, where the set of real numbers that are "close to zero" is modelled. For the fuzzy set the chosen membership function in this case is:

$$\mu(x) = \frac{1}{1+10x^2}. \tag{4.2}$$

Comparing the two different representations in Figure 4.1 it should become clear as to the difference between fuzzy sets and the traditional crisp sets.

[1] This definition holds for *crisp* sets, as will be seen later this will be extended in the case of fuzzy sets.

[2] Fuzzy sets and degrees of membership should not be confused with probabilities (even tough they both deal with the same domain of values.)



(a) (b)

Figure 4.1: Above can be seen the two different ways to represent the set of real numbers "close to zero". Figure (a) shows the difficulties in defining what constitutes a number close to zero, in this case 0.5 was selected as border. In Figure (b) the corresponding fuzzy set is displayed, the membership function is easier to model.

The support[3] of a fuzzy set A is defined as the crisp set of all elements of the universal set $\mathcal{U}$ with a nonzero degree of membership in A, or:

$$\text{support}(A) = \{x \in \mathcal{U} \mid \mu_A(x) > 0\}.$$

Let x_i denote an element of the support of A and $\mu_i = \mu_A(x_i)$, then the support of A is often written with the special notation:[4]

$$A = \mu_1/x_1 + \mu_2/x_2 + \ldots + \mu_n/x_n.$$

For example, consider Table 4.1 which represents the set A of "old people". The support of this set would then be:

$$A = 0.1/20 + 0.2/30 + 0.4/40 + 0.6/50 + 0.8/60 + 1.0/70 + 1.0/80.$$

This notation can also be written in a more concise form:

$$A = \sum_{i=1}^{n} \mu_i/x_i,$$

or if the fuzzy set A is continuous rather than a finite countable set:

$$A = \int_{\mathcal{U}} \mu_A(u)/u.$$

[3]The support of a fuzzy set does not possess the same meaning as the support of association rules (defined in Chapter 2).

[4]Here the use of $+$ denotes the (crisp) union rather than the arithmetic sum.

The concepts of subsets and equality have both been transfered to fuzzy set theory. Two fuzzy sets A and B are equal if:

$$\mu_A(x) = \mu_B(x), \qquad \forall x \in \mathcal{U} .$$

Similarly, both subsets and proper subsets can be defined. Let A and B be two fuzzy sets, then A is said to be a subset of B if and only if:

$$\mu_A(x) \leq \mu_B(x), \qquad \forall x \in \mathcal{U} .$$

The two sets are proper subsets when $A \subseteq B$ and $A \neq B$.

The basic operations defined for crisp sets (i.e. union, intersection and complement), along with the "laws" have been defined for fuzzy sets as well. These operations have been defined in various ways, giving them different properties. However, the variants definition 4.1 provides are fully compatible with their crisp equivalents thus producing the same results (remember that fuzzy set theory is a superset of the classical theory) [17].

The union of two fuzzy sets A and B is the maximum grade of membership in either set. This is equivalent to the resulting set being "the smallest fuzzy set containing both A and B" [30]. The intersection between two fuzzy sets A and B is the minimum of either membership functions, while the complement of A is defined to be $1 - \mu_A(x)$ [17]. This means that if a person belongs to the set of old people with a membership grade of 0.3, he or she would be a member of the set of "not old" people (not necessarily identical to the set of young people) with a degree of 0.7.

Definition 4.1 *Let A and B be two fuzzy sets, then for every $x \in \mathcal{U}$:*

- *The union $A \cup B$ is:* $\mu_{A \cup B}(x) = \max[\mu_A(x), \mu_B(x)]$
- *The intersection $A \cap B$ is:* $\mu_{A \cap B}(x) = \min[\mu_A(x), \mu_B(x)]$
- *The complement $\overline{A}$ is:* $\mu_{\overline{A}}(x) = 1 - \mu_A(x)$

4.2 Linguistic Variables

An important concept used by many types of applications, including fuzzy controllers and fuzzy expert systems, is the notion of a linguistic variable. A linguistic variable differs from other kinds of variables in that it takes words or sentences as values instead of numbers. These words can be expressed using a natural language or if more appropriate, an artificial language. For example, *Height* would be a linguistic variable if its values was of the kind *short*, *tall* or *very tall* instead of the corresponding numerical values 1.45, 1.85 or 1.90. The different linguistic values are defined as fuzzy variables [27].

Comparing the use of a numerical terms such as 1.90 with *very tall*, it becomes clear that the numeric value is precise. The linguistic value on the other hand is vague and offers a sense of ambiguity. This vagueness is what provides linguistic variables with their power and expressiveness, allowing easier capturing of expert knowledge or creating control mechanisms for complex systems [21].

A fuzzy variable is an extension of a regular variable, both of which can be defined with a triple $(X, \mathcal{U}, R(X;u))$ where X is the name of the variable, $\mathcal{U}$ is the universe of discourse and u is a generic name for the elements of $\mathcal{U}$. The term $R(X;u)$ is a subset of $\mathcal{U}$ which restricts the possible values of u with respect to X. For a normal variable (i.e. non-fuzzy variable) this is most often referred to as the range of the variable, while for a fuzzy variable it is called a fuzzy restriction [27, 28].

The assignment of a value u to a variable x is determined by:

$$x = u, \qquad u \in R(X;u). \tag{4.3}$$

An important concept regarding the preceding equation and fuzzy variables is the notion of compatibility. This refers to the degree to which Equation 4.3 is satisfied, defined as:

$$c(u) = \mu_{R(X;u)}(u), \qquad u \in \mathcal{U}, \tag{4.4}$$

where $\mu_{R(X;u)}$ is the membership function for $R(X;u)$ [28].

A linguistic variable is defined in a manner similar to that of a fuzzy variable. A linguistic variable is characterised by a quintuple $(x, T(x), U, G, M)$, where x is the name of the variable, $T(x)$ is the term-set of x, as before $\mathcal{U}$ is the universe of discourse, G is a syntactic rule for defining the different values of x and M is a semantic rule assigning meaning to the different values of the term-set [18, 28].

Instead of using a restriction (as is done with fuzzy variables), the term-set $T(x)$ defines all the possible values that the linguistic variable can assume. For the linguistic variable *Height* this might be:

$$T(Height) = tall + very\ tall + not\ very\ tall + \ldots + very\ short + not\ short + \ldots$$

For a finite term-set it is possible to specify every value as a list, however if the finite term-set is very large or if a linguistic variable has a infinite term-set it can become very cumbersome or even impossible to do. Therefore a syntactic rule or grammar G might be specified and associated with the linguistic variable, specifying how the values are generated

A name generated by G is referred to as a term. A term constructed by either a single word or a group of words functioning as a unit is called an atomic term. Composite terms are constructed from atomic terms (also called sub terms). Words such as “very”, “somewhat” or “completely” are also sometimes referred to as hedges.

The semantic rule M assigns a meaning to the term X by use of the restriction defined for the value u as imposed by the fuzzy variable X [28], or:

$$M(X) = R(X;u). \tag{4.5}$$

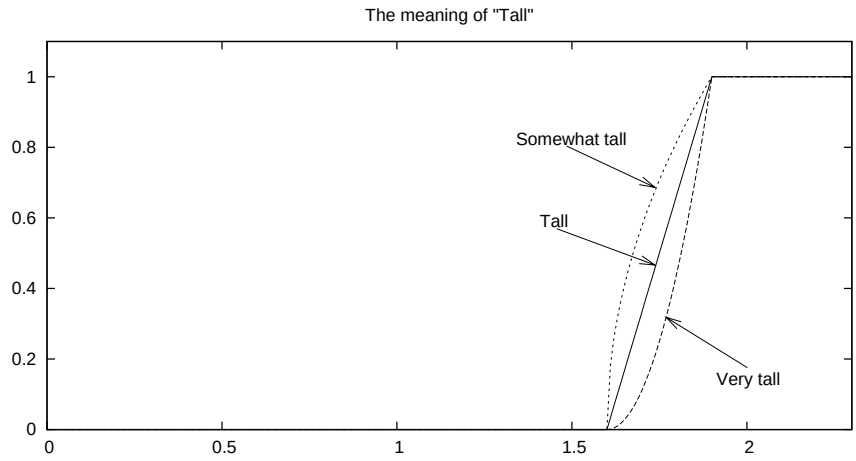


Figure 4.2: Applying the two hedges *very* and *somewhat* (defined as Equation 4.7 and 4.8) to the linguistic value $tall$ would create the membership function seen in the figure above.

Consider the following example. Let *Height* be a linguistic variable with $u = [0, 2.5]$. Then a linguistic value of *Height* might be $tall$, another *short* or *very* $tall$. The final value *very* $tall$ is a composite term, made up of the atomic term $tall$ and the sub term (or hedge) *very*. Let the meaning of $tall$ be defined as:

$$M(tall) = \begin{cases} 0 & x \leq 1.60 \\ \frac{10(x-1.60)}{3} & 1.60 \leq x \leq 1.90 \\ 1 & 1.90 \leq x \end{cases} \tag{4.6}$$

Applying the hedge *very* to the above meaning of $tall$ can be done in several ways, one of the more commonly used definitions of *very* is the concentration operator:

$$\text{CON}(A) = A^2. \tag{4.7}$$

This resulting meaning of *very* $tall$ then becomes:

$$M(very\ tall) = \begin{cases} 0 & x \leq 1.60 \\ \left(\frac{10(x-1.60)}{3}\right)^2 & 1.60 \leq x \leq 1.90 \\ 1 & 1.90 \leq x \end{cases}$$

Similarly, the hedge *somewhat* could be defined as the dilution operator:

$$\text{DIL}(A) = A^{\frac{1}{2}}. \tag{4.8}$$

The result of applying these two hedges to the linguistic value $tall$ is shown in Figure 4.2.

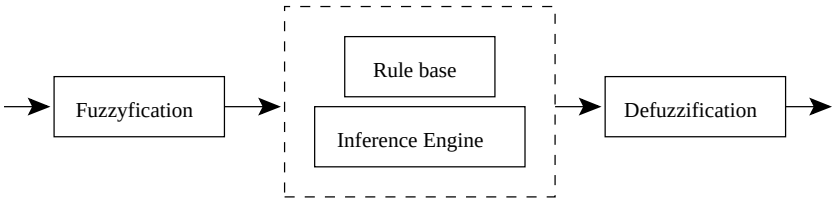


Figure 4.3: A fuzzy controller consists of four different main components. These are responsible for converting incoming values into linguistic values, inferring a fuzzy control action and finally converting this fuzzy action into a non fuzzy counterpart.

4.3 Fuzzy Controller

One of the most successful areas where the concepts of fuzzy systems have been applied is that of the fuzzy controllers (also called fuzzy logic controllers). Creating automatic control systems for various processes is a very difficult and complex task, especially when the system has to handle uncertainties, imprecision or missing information. Traditional control techniques often rely upon defining a precise mathematical model of the process, but that is not always possible. In many cases human operators are capable of controlling very complex systems, without knowledge of their dynamics or true workings.

Fuzzy control is an attempt at overcoming the difficulties encountered by the traditional control mechanisms, and creating a system which is capable of approximate reasoning. Expert knowledge is easy to translate into fuzzy rules using linguistic variables, and capturing the actions performed by a human operator allows poorly understood processes to be translated into a fuzzy controller [21]

The four main components of a fuzzy controller is seen in Figure 4.3, and consists of a fuzzification interface, a knowledge base combined with a inference engine and a defuzzification interface. The fuzzy controller converts the incoming values into linguistic values which is then used to create an outgoing control action. The inference engine combines the linguistic values created by the fuzzification interface with the knowledge base to infer a suitable fuzzy control action. To perform this task, the inference engine uses fuzzy implications and the rules of inference in fuzzy logic. Finally, the defuzzification interface converts the resulting fuzzy control action into a non-fuzzy equivalent [18].

The following sections will describe the different parts of the fuzzy controller in more detail.

4.3.1 Fuzzification

The data observed by the fuzzy controller is often crisp, which the fuzzification interface converts into linguistic values. These linguistic values will then be used

by the inference engine to infer an action. The choice of membership function of the fuzzy sets is application dependant. The simplest membership function is the fuzzy singleton [18], where the input value x_0 is mapped to the fuzzy set A as:

$$\mu_A(x) = \begin{cases} 1 & \text{if and only if } x = x_0 \\ 0 & \text{otherwise.} \end{cases}$$

Since the fuzzy singleton does not really provide much benefit over using a traditional non fuzzy controller, there are a large number of membership functions commonly used. Besides the fuzzy singleton, the triangular and trapezoidal membership functions are the simplest. The triangular membership function is defined to be:

$$\mu_A(x) = \begin{cases} \frac{x-a}{b-a} & a \leq x \leq b \\ \frac{c-x}{c-b} & b \leq x \leq c \\ 0 & \text{otherwise,} \end{cases}$$

where the parameters a and c specifies the start- and endpoint of the triangle, and b determines the peak. The trapezoidal function is defined in a similar manner. Other commonly used functions includes the Gaussian and sigmoidal, however these are more computationally expensive, which makes them unsuitable for some real time applications. Examples of these membership functions are displayed in Figure 4.4.

4.3.2 Knowledge Base

The knowledge base of a fuzzy controller consists of two main components. The database holds the definition for each fuzzy variable and the rule base contains the actual control rules used by the system. The fuzzy variables are defined by either membership functions or in a numerical form (this is done only if a discretized universe is used). A control rule is comprised of an antecedent and a consequent [18], and has the form

$$\text{IF } x_1 \text{ is } A_1 \wedge x_2 \text{ is } A_2 \wedge \ldots \wedge x_n \text{ is } A_n \text{ THEN } y \text{ is } B.$$

Database

The database is responsible for assigning a grade of membership to each possible linguistic value both the input and output can assume [6]. Each linguistic value is defined either numerically or by use of a membership function. If the universe of discourse is discrete, a table can be used to map each input to a corresponding membership grade (Table 4.1 is an example of this method). The advantage of this method is the speed of a look up compared to the evaluation of a (more or less) complicated function. The drawback is the loss of precision due to the discrete nature of the universe. The same method can be used for a continuous universe, since it can always be discretized [18]. The other method is by use of a membership function, which was discussed in section 4.3.1.

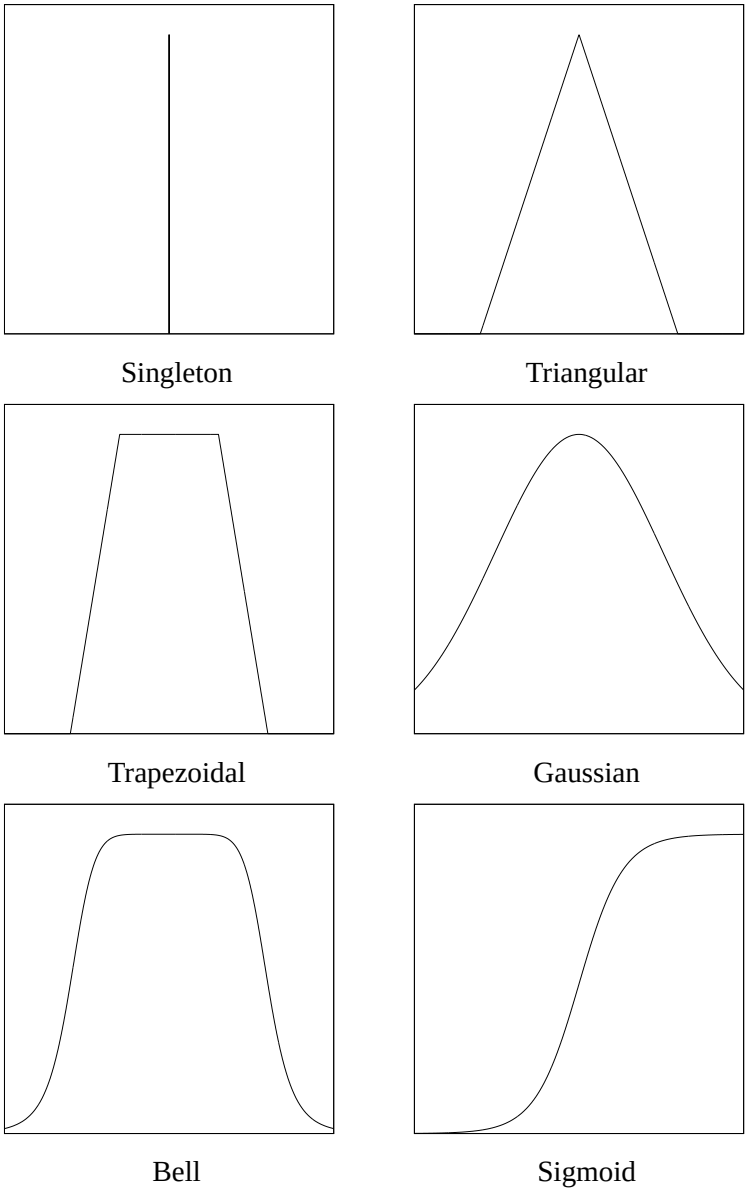


Figure 4.4: Some of the membership functions commonly used in fuzzy controllers are displayed above.

Determining which membership function to apply to which fuzzy variable is a difficult task. This can be done either by a quantisation process, where the continuous universe is discretized into a finite number of segments. Each segment is called a quantisation level, and for each quantisation level every term in the term set is assigned a membership grade. This of course means that the linguistic values are defined numerically (as mentioned above).

The other way to define the membership functions is through a learning or tuning process. The tuning process makes use of a previously defined database (defined through either learning or quantisation), and modifies the meaning of each linguistic value in order to improve performance. The learning process, on the other hand, does not require an initial database. Instead the process defines the membership functions through various means (e.g. watches a human operator performing the task). Learning the database is most often combined with learning the rule base [6, 18].

Rule base

The rule base contains the fuzzy control rules used by the fuzzy system. Each rule consists of an antecedent (with one or more terms), and a consequent. The different terms of the antecedent are linguistic variables, and their respective values are defined by the database. For the consequent, there are three main types. In the simplest form, the conclusion is another fuzzy variable. Another more complex method defines the consequent to be a polynomial of the linguistic values used in the antecedent, thus:

$$\text{IF } x_1 \text{ is } A_1 \wedge x_2 \text{ is } A_2 \wedge \ldots \wedge x_n \text{ is } A_n \text{ THEN } y = p_0 + p_1 x_1 + \ldots + p_n x_n$$

This approach can be generalised further, allowing the conclusion to be a function of the values in the antecedent as in:

$$\text{IF } x_1 \text{ is } A_1 \wedge x_2 \text{ is } A_2 \wedge \ldots \wedge x_n \text{ is } A_n \text{ THEN } y = f(x_1, \ldots, x_n).$$

Defining the actual rules of the rule base is a difficult task which depends upon the particular application. Since each application has its own data- and rule base, the performance of the resulting system depends on the quality of these components. A number of different methods can be used to create the rule base, and a high quality rule base might require a combination of several of these methods [6]:

1. Expert experience and control engineering knowledge: Most people find fuzzy rules to be a very intuitive way of expressing knowledge, considering the fact that linguistic values are more natural than crisp numerical values. This leads to the approach where a human expert manually creates the necessary rules for guiding the system, based on experience or a deep understanding of the process.

2. Modelling of the operator's control actions: Classical control theory is unable to model certain complex processes, even though a human operator can (more or less) easily control the system. By capturing the inputs and how the human operator responds to those inputs, fuzzy rules can be deduced.

3. Based on the fuzzy model of the process: By using a fuzzy model of the system, the optimal fuzzy control rules can be extracted. However, creating the fuzzy model is difficult since a linguistic description of the system and its dynamics is required. According to Lee [18] this approach will create a system with better performance and reliability, unfortunately this method is not yet fully developed.

4. Based on learning and self-organising: Finally, the rule base can be found by creating a system to automatically find and create the necessary rules, modifying them (during runtime) based on experience [18].

4.3.3 Inference Engine

The knowledge base provides the system with both rules and the semantic meaning of the terms used by the rules. However, this does not provide any advantage or benefit without a method to draw and form conclusions based on the inputs received by the system. In classical logic a rule might have the appearance of:

$$(a \wedge b \wedge c) \rightarrow q$$

If the truth values of the premises (a, b and c above) are known, it is possible to ascertain whether q holds or not. The rule of inference which is normally used for this is the *modus ponens* (sometimes also referred to as the rule of detachment):

$$\begin{array}{l} p \rightarrow q \\ \underline{p \qquad} \\ \therefore q \end{array}. \tag{4.9}$$

This rule states that if p implies q and p is true, then it is possible to infer the truth of q. An example of this might be "if it is snowing then it is cold" and "it is snowing", then the conclusion that "it is cold" can be made. Besides the modus ponens, there are several other rules (e.g. modus tollens and the law of syllogism) used for inference [13].

In fuzzy controllers the consequent of one rule is never used in the antecedent of another rule. This removes the need for other inference mechanisms besides the modus ponens. The basis of the fuzzy equivalent of the modus ponens (called the compositional rule of inference or generalised modus ponens) is the notion of fuzzy implication.

As Table 4.2 shows, there exists several different definitions of both conjunction and disjunction. The two most commonly used variants are the intersection/union pair of fuzzy set theory, as well as the algebraic sum and product.

Table 4.2: There are a large number of definitions of the fuzzy equivalents of disjunctions and conjunctions.

Conjunctions	
Intersection	$x \wedge y = \min(x,y)$
Algebraic product	$x \cdot y = xy$
Bounded Product	$x \odot y = \max(0, x+y-1)$
Disjunctions	
Union	$x \vee y = \max(x,y)$
Algebraic sum	$x+y = x+y-xy$
Bounded sum	$x \oplus y = \min(1, x+y)$
Disjoint sum	$x \Delta y = \max(\min(x, 1-y), \min(1-x, y))$

Depending on which definition is used, it is possible to derive several forms of implications (each having distinct advantages and disadvantages) [19]. Commonly used implication-functions are:

- Zadeh's maxmin rule of fuzzy implication:

$$\begin{aligned} R_m &= (A \times B) \cup (\neg A \times V) = \\ &= \int_{U \times V} (\mu_A(u) \wedge \mu_B(v)) \vee (1 - \mu_A(v)) / (u,v) \end{aligned} \tag{4.10}$$

- Larsen's product operation rule of fuzzy implication:

$$\begin{aligned} R_p &= A \times B \\ &= \int_{U \times V} \mu_A(u)\, \mu_B(v) / (u,v) \end{aligned} \tag{4.11}$$

- Mamdani's mini-operation rule of fuzzy implication:

$$\begin{aligned} R_c &= A \times B \\ &= \int_{U \times V} \mu_A(u) \wedge \mu_B(v) / (u,v) \end{aligned} \tag{4.12}$$

By comparing Zadeh's maxmin rule of fuzzy implication (Equation 4.10) to the definition of implication used in classical logic:

$$p \rightarrow q \quad \Leftrightarrow \quad (p \wedge q) \vee \neg p,$$

the similarities are obvious. Lee states in [19] that the other two variants of implication (Equation 4.11 and 4.12) are the most suitable for use in fuzzy controllers.

The fuzzy equivalent of the modus ponens (Equation 4.9) used in fuzzy systems is called the generalised modus ponens, which is based on the compositional rule of inference as defined by Zadeh [29]. The generalised modus tollens is used in

expert systems where backwards chaining is important, but as mentioned before a fuzzy controller is only dependant on forward driven inference (i.e. no consequent appears in the antecedent of another rule).

The compositional rule of inference allows the generalised modus ponens to be defined in the following manner. Let A and B be two fuzzy sets in the universes U and V, and the fuzzy relation $R : U \times V \rightarrow [0,1]$ define a fuzzy implication. Then the generalised modus ponens can be expressed as:

$$B = A \circ R.$$

Using the membership functions for the respective symbols, the above expression translates into:

$$\mu_B(v) = \sup_{u \in U} T\left(\mu_A(v), \mu_R(u,v)\right), \tag{4.13}$$

where T is any conjunction (see Table 4.2 for examples) [12, 19].

Being able to form a conclusion based on a single rule is not particularly useful for use in fuzzy controllers, so the fuzzy implication has to be extended to cover multiple rules. Consider the general rule base below:

$$\begin{array}{rl} \text{input:} & x \text{ is } A' \text{ and } y \text{ is } B' \\ R_1: & \text{if } x \text{ is } A_1 \text{ and } y \text{ is } B_1 \text{ then } z \text{ is } C_1 \\ \text{also } R_2: & \text{if } x \text{ is } A_2 \text{ and } y \text{ is } B_2 \text{ then } z \text{ is } C_2 \\ & \vdots \\ \text{also } R_n: & \text{if } x \text{ is } A_n \text{ and } y \text{ is } B_n \text{ then } z \text{ is } C_n \\ \hline & z \text{ is } C' \end{array}.$$

Lee [19] shows that the resulting conclusion from performing the GMP on each successive rule and aggregating the result, is equivalent to combining all the rules and inferring the resulting conclusion from the entire rule base.

An important special case arises when the inputs are fuzzy singletons, which is most often the case in fuzzy controllers. Then the Equations 4.11 and 4.12 can be rewritten and simplified into:

$$R_c: \quad \mu_{C'} = \bigcup_{i=1}^{n} \alpha_i \wedge \mu_{C_i} \quad \text{where } \alpha_i = \mu_{A_i}(u_0) \wedge \mu_{B_i}(v_0) \tag{4.14}$$

$$R_p: \quad \mu_{C'} = \bigcup_{i=1}^{n} \alpha_i \cdot \mu_{C_i} \quad \text{where } \alpha_i = \mu_{A_i}(u_0) \cdot \mu_{B_i}(v_0) \tag{4.15}$$

It is clear that the above equations are simple to implement in software, and inferring conclusions from the rule base requires a relatively low computational effort. Something which is important for real time low-cost applications.

4.3.4 Defuzzification

The inference engine produces a single linguistic value as the conclusion or consequent. However, most applications requires crisp control actions. The final part of

a fuzzy controller is the defuzzification interface, which converts the fuzzy output into a non fuzzy equivalent. This is accomplished by applying a defuzzifier, which acts as a function that maps each linguistic value into a non-fuzzy output.

Several techniques have been developed to facilitate this mapping, unfortunately different applications benefits from different defuzzifiers. The defuzzification strategies that are commonly used includes [19, 9, 16, 24]:

- The max criterion method: This defuzzifier simply chooses the point where the membership grade of the control action reaches its maximum.
- The Mean of Maxima (MOM): Determines the resulting control action by calculating the mean of all the maximum values.
- The Center of Area[5] (COA): The control action is determined by computing the centroid of the area of the membership function. For the continuous case this is:

  $$u = \frac{\int \mu_i(x)\, x \, \mathrm{d}x}{\int \mu_i(x) \, \mathrm{d}x}.$$

  When the output is a fuzzy singleton (as opposed to a more complex membership function) the above equation can be rewritten into:

  $$u = \frac{\sum_i \mu(x_i)\, x_i}{\sum_i \mu(x_i)}. \tag{4.16}$$

Lee [19] mentions that the mean of maxima produces fuzzy controllers with better performance than those using the max criterion, and that similar results applies to the difference in performance between the COA and the mean of maxima.

COA has the nice property of always producing smoothly varying control actions, and is therefore often used in fuzzy controllers. However, it is computationally expensive and therefore can be unsuitable for certain real time applications. Several attempts to address this problem have been made. For instance, Runkler et al. [24] proposes an iterative approximation named DECADE suitable for implementation directly in hardware.

[5] Center of Gravity (COG) is another name for this method.

Chapter 5

Learning Behavioural Mappings

In this thesis the "Programming by Demonstration" paradigm is used together with association rules to allow a user to teach the robot a simple behaviour. Hopefully, the robot will be able to perform this behaviour autonomously thereafter. The allowed behaviours are limited to those that are purely reactive in nature (i.e. follows the reactive paradigm).

The reactive paradigm was inspired by biological systems such as the behaviours of insects and simple animals, where complex actions are performed through the interaction between simple behaviours. Examples of such simple behaviours might be "follow wall", "avoid light" or "do not eat red things". The advantage of this method is the relatively low computational effort required to implement and execute the behaviours. Unfortunately the absence of planning and memory introduces severe limitations. To overcome this, a hybrid approach is often used, where reactive behaviours are superseded by more complex systems allowing planning, memory and other goal oriented methods [22].

The system is based on the notion of a stimuli/response mapping guiding the underlying behaviour. This is expressed as the mapping $B : S(t) \rightarrow R(t)$ which determines the appropriate response for the robot as a direct function of its sensor inputs. In a purely reactive system only the stimuli received at the current time t is considered. The function B will be implemeneted as a set of association rules providing the actual function $S \Rightarrow R$.

The data (henceforth referred to as samples) gathered by the system as the user demonstrates the behaviour, is stored as a database. The association rules are then extracted from this database using an arbitrary (rule discovery) algorithm.[1] The resulting set of association rules is then used by the system to guide the robot.

In order for the association rules to be created the raw sensor data has to be converted into a more appropriate form using a *partitioning scheme*. A set of borders (which defines a number of disjunct partitions), together with the actual number of borders forms a partioning scheme. These partitions corresponds to different

[1] We shall make use of the 3rd party software Apriori [4], which is based on the Apriori algorithm [2].

ranges, such as “medium distance” or “very long distance” (see Section 5.3 for more detailed information).

5.1 Previous Work

As mentioned in Section 1.1 this thesis strives to build upon research previously done by Hellström [14]. In his work it was assumed that the partitioning scheme was known beforehand. If the partitioning scheme was inappropriate, given the underlying behaviour, the resulting performance became very low. The first part of this chapter attempts to solve this problem (see Section 5.5 and 5.6).

Hellström also identified the following three cases which could occur as the stimuli was matched against the set of association rules:

1. Exactly one rule fires: This is the trivial case, the control action specified by the one rule is used.
2. Multiple rules fires: A more complex task, Hellström proposed performing a majority vote to determine the combined action.
3. No rule fires: Since no rule triggered, it becomes difficult to determine the resulting action. To solve this problem, Hellström created a simple nearest-neighbour method, were the rule with the antecedent closest to the current stimuli $S(t)$ was chosen.

The second part of this chapter regards an attempt at providing an alternative method to treat the case when multiple rules are triggered. Instead of a majority vote, a fuzzification of the borders will be done using concepts from both fuzzy logic (Section 5.7) and genetic algorithms. The resulting controller will resemble that of a fuzzy controller (Chapter 4 contains an overview of fuzzy systems). This approach will hopefully improve performance beyond that given by the majority vote algorithm.

5.2 Hardware

The robot used in all the experiments is the Khepera robot developed by K-Team, Figure 5.1 shows a simplified overview. It uses eight infrared sensors (labelled $IR_0, IR_1, \ldots, IR_7$) to determine the distance to any obstacles in its vicinity, as well as eight sensors to detect ambient light. The range of each infrared sensor is around 50mm, depending on the properties of the surface the light bounces off of (i.e. a reflective surface is easier to detect than a black surface). Each proximity sensor uses a 10 bit analog to digital (A/D) converter, returning an integer value between zero (no obstacle within range) and 1023 (an obstacle detected very close) [15].

The Khepera has two motors, each driving one of its wheels. The motor speeds can be independently controlled, allowing for motions such as rotation or sharp turns.

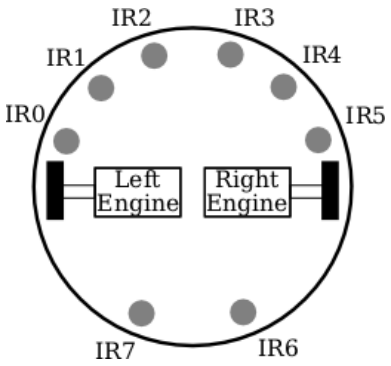


Figure 5.1: The Khepera robot has two motors and eight infrared sensors. The infrared sensors are sensitive up to a distance of about 50mm. Each of the two motors can be set independently, allowing for a range of different motions (including rotation and sharp turns).

5.3 Extracting Rules

The data from which the association rules will be extracted is a pair consisting of a stimuli and a response. The general form of the data is

$$ir_0, ir_1, \ldots, ir_7, y.$$

Where the stimuli consist of a collection of sensor readings (ir_i in the above equation), and the response y is the motor control action issued at that point in time. If the underlying behaviour is covered by the reactive paradigm, it is possible to determine this behaviour from these data pairs.

In order for this data to be used for extracting association rules, the raw data has to be converted into symbols using some form of a mapping. One such example of a partitioning scheme might be:

$$ir_i = \begin{cases} 0 & \text{if} \quad 0 \leq IR_i \leq 399 \quad \text{(long distance)} \\ 1 & \text{if} \quad 400 \leq IR_i \leq 1023 \quad \text{(short distance).} \end{cases} \tag{5.1}$$

This mapping corresponds to dividing the sensor range into a number of disjunct parts. When the sensor readings have been translated into symbols, it is possible to create the association rules.

It should be clear that the choice of borders (and by extension how the sensor range is partitioned) will have a huge impact upon the quality of the resulting rule base. A rule base of poor quality is equivalent with failing to capture or learn the underlying behaviour, whereas a rule base of good quality will also have a good performance.

The training set used to create and tune the rules is created through one of two methods, either by a manually programmed controller or through the use of a remote control operated by a human (more detailed information in section 5.4).

When the database has been translated into symbols, a program called Apriori[2] is used to extract the association rules. In this thesis the association rules are restricted to only having a motor control action as the consequent and the antecedent may only contain sensor readings, thus:

$$ir_i = v_i \wedge ir_j = v_j \wedge \ldots \wedge ir_k = v_k \Rightarrow y = a.$$

This restriction is well suited for a reactive behaviour since it creates a clear mapping between stimuli and response. Relaxing this restriction would allow for more powerful systems to be created. However, the reactive paradigm would not be well suited for this.

5.4 Controllers

To obtain the data required for training and validation two different types of controllers were used. The first case was a series of three different manually programmed controllers of increasing complexity, and the second case is a remote control operated by a human teacher.

The manually programmed controllers are described in Figure 5.2, and they all describe the same underlying behaviour. The main difference between the three variants is the number of borders used, e.g. the controller in Figure 5.2(a) uses one border while the controller Figure 5.2(c) uses three borders.[3] Using manually programmed controllers somewhat defeats the purpose of locating the borders, but it also means that this approach can more easily be evaluated.

With each controller comes an associated partitioning scheme, mapping the observed inputs to symbols for the creation of association rules. The partitioning scheme in Equation 5.1 corresponds to the controller in Figure 5.2(a). For the two-border-controller in Figure 5.2(b) and the one in Figure 5.2(c) the following two equations provides the partitioning scheme:

$$ir_i = \begin{cases} 0 & \text{if} \quad 0 \leq IR_i \leq 299 & \text{(long distance)} \\ 1 & \text{if} \quad 300 \leq IR_i \leq 899 & \text{(medium distance)} \\ 2 & \text{if} \quad 900 \leq IR_i \leq 1023 & \text{(short distance),} \end{cases} \tag{5.2}$$

$$ir_i = \begin{cases} 0 & \text{if} \quad 0 \leq IR_i \leq 249 & \text{(very long distance)} \\ 1 & \text{if} \quad 250 \leq IR_i \leq 499 & \text{(long distance)} \\ 2 & \text{if} \quad 500 \leq IR_i \leq 749 & \text{(medium distance)} \\ 3 & \text{if} \quad 750 \leq IR_i \leq 1023 & \text{(short distance).} \end{cases} \tag{5.3}$$

The problems posed by the second type of controllers, i.e. the remote control, are far more challenging. The idea is to use a remote control in order to demonstrate

[2]This software is based on the Apriori-algorithm as presented by Agrawal et al. in [1]. More information regarding this software can be found at [4].

[3]Most types of behaviours requires few borders and creating controllers that utilises a large number of borders is quite difficult.

Table 5.1: The different motor control actions are represented by integers, and their meaning is explained below.

Action (y)	Description	Action (y)	Description
-3	Clockwise rotation	3	Counter clockwise rotation
-2	Hard clockwise	2	Hard counter clockwise
-1	Soft clockwise	1	Soft counter clockwise
		0	Straight ahead

to the robot a specific behaviour, and through the use of association rules it will (hopefully) learn the behaviour. Using a remote control (especially operated by a human) introduces both benefits and drawbacks.

One of the clear benefits is that no knowledge of how to actually implement any given behaviour is required, as long as it is reactive in nature. Showing the robot a specific behaviour should take far less time than manually programming it. The drawback is the added noise and the mistakes done by the operator. The more mistakes and inconsistencies introduced during training, the lower the quality of the resulting rule base.[4]

5.5 Determining Optimal Borders

Since the association rules requires a partitioning scheme to effectively learn and map a behaviour, this is the first problem that has to be solved. The obvious method that could be used for determining this partitioning scheme is to mearly guess. However, it should be apparent that this method is suboptimal, and therefore a more intelligent solution will be proposed.

The partitioning scheme is determined by two factors, the number of borders used and the borders themselves. Section 5.6 discusses the problem of finding the optimal number of borders, while this section focuses on determining the actual borders. We will assume that the optimal number of borders is known and that for the general case it is $\widehat{\eta}$ (the optimal number of borders).

Before continuing, a short note on the notation used throughout this section. The set of borders will be denoted by $\beta_{\widehat{\eta}}$ and the set of *optimal* borders will be referred to as $\widehat{\beta}_{\widehat{\eta}}$. Both of these sets are ordered such that

$$\beta_{\widehat{\eta}} = \left\{b_1, b_2, \ldots, b_{\widehat{\eta}}\right\}, \quad 0 \leq b_1 \leq b_2 \leq \ldots \leq b_{\widehat{\eta}} \leq 1023.$$

To clarify the notation, the controller in Figure 5.2(b) would be expressed as:

$$\beta_2 = \{300, 900\},$$

[4]The old adage that the accomplishments of the student reflects the competence of the teacher holds true for this case.

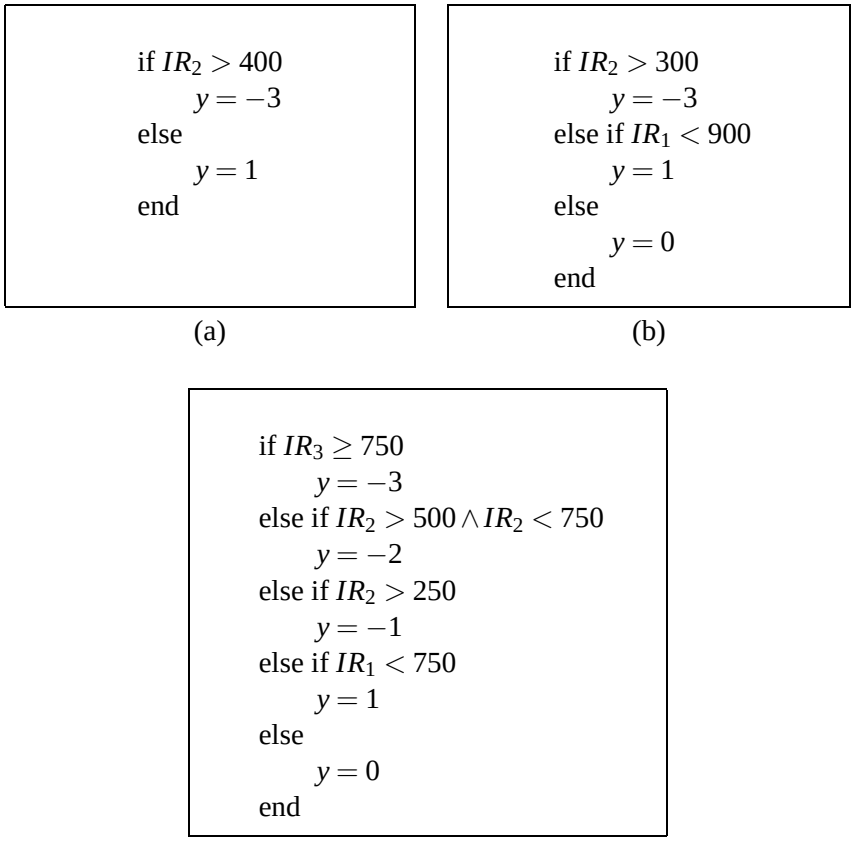


Figure 5.2: (a) This is a single border controllers, creating two partitions and utilising two separate motor control actions. Figure (b) is a slightly more advanced controller, using two borders and three motor control actions. Figure (c) has three different borders, and a host of control actions. Common between all these controllers is that they exhibit the same behaviour (following the left wall), but uses more and more advanced methods to achieve it.

where the two edge borders zero and 1023 are implied and therefore left out.

The first case that shall be considered is when all the sensors is subject to a single set of borders, meaning that their input space is partitioned in the exact same way. This will yield the simplest controllers using the smallest partitioning scheme.

Let $\Phi : [0,1023]^{\eta} \rightarrow [0,1]$ be a function which to each set of borders associates a value corresponding to its fitness. The objective function (that is, the function the system is trying to minimise) is the ratio of samples for which the system yields the wrong motor control action (compared to the action stored together with the sensor data). A high fitness corresponds to a low error and vice versa. The problem of finding the set of optimal borders can then be expressed as:[5]

$$\widehat{\beta}_{\widehat{\eta}} = \arg\max_{\beta_{\widehat{\eta}}} \Phi\left(\beta_{\widehat{\eta}}\right), \qquad \text{where } \beta_{\widehat{\eta}} \in [0,1023]^{\widehat{\eta}}. \tag{5.4}$$

The second case is when each sensor is allowed its own set of borders, hencefort referred to as *individual borders*. This case is interesting to examine since intuitively it should allow the resulting association rules to become more flexible requiring a lower value of $\widehat{\eta}$. This problem could also be described as locating a set of optimal sets of optimal borders: $\widehat{\mathcal{B}}_{\widehat{\eta}} = \left\{\widehat{\beta}_{\widehat{\eta}1}, \widehat{\beta}_{\widehat{\eta}2}, \ldots, \widehat{\beta}_{\widehat{\eta}8}\right\}$, or:

$$\widehat{\mathcal{B}}_{\widehat{\eta}} = \arg\max_{\mathcal{B}} \Phi(\mathcal{B}), \qquad \text{where } \mathcal{B} \in \left([0,1023]^{\widehat{\eta}}\right)^8. \tag{5.5}$$

The task of finding the set of optimal borders $\widehat{\beta}_{\widehat{\eta}}$ is rather trivial for the case when $\widehat{\eta} = 1$, since there would be only 1024 possible solutions to consider. Unfortunately, the size of the solution space increases exponentially as $\widehat{\eta}$ increases. An even worse case occurs if each sensor is allowed its own set of borders, since the solution space then would increase (for the Khepera) eight times faster. Locating the global minimum is clearly a nontrivial task, requiring an efficient algorithm. In reality, the best solution that can be expected to be found in finite time is simply "good enough".[6] The complexity of the solution space is hinted at in Figure 5.3, where the fitness of a remote controlled behaviour is visualised (for all sets of borders of length two).

Since genetic algorithms have worked well in other multidimensional problems (especially when the dimensionality is high), they were chosen for the task of determining the set of optimal borders. Since the sensor range of a Khepera robot is only ten bits, it is natural to encode each border as a ten bit binary allele (see Figure 5.4). As often is done when binary encoding is used, the pure binary code is replaced by a Gray code.

[5] We are trying to maximise the fitness, which corresponds to minimising the error of the objective function (in this case the error for the training or test set).

[6] For the case when $\widehat{\eta} = 3$ and individual borders are sought for, the total solution space contains $1024^{8\widehat{\eta}} = 2^{240}$ possible solutions. This makes it close to impossible for any algorithm to consider all the points in finite time, which implies that locating the global minimum is impossible.

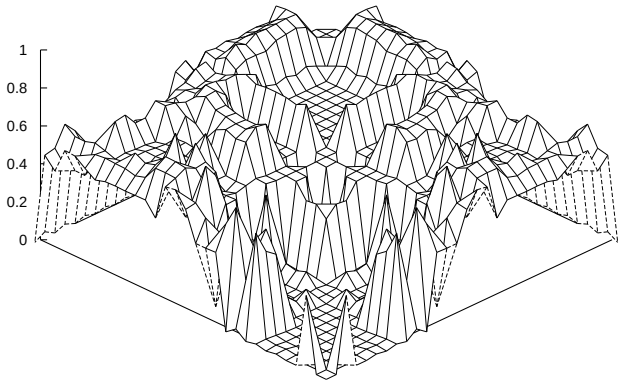


Figure 5.3: The solution space can be quite complex, in this case a two dimensional space is visualised. The x- and z-axis depicts the sensor range $0 \ldots 1023$ of the Khepera robot, and the y-axis corresponds to the fitness of the partitioning scheme defined by the values of the x- and z-axis. The mirroring which occurs along the center is caused by the repair mechanism used by the fitness function.

The set of borders is required to be ordered (so that the partitioning scheme makes sense) and there is no guarantee that the genetic optimisation process produces this property, therefore a means of repairing the defect chromosomes are needed. In this case a simple repair mechanism was implemented within the fitness function where the borders are simply sorted (their position is irrelevant, only their actual value is interesting).

The fitness function $\Phi\left(\beta_{\widehat{\eta}}\right)$ uses the partitioning scheme specified in $\beta_{\widehat{\eta}}$ to convert the raw sensor data into symbols, which is then used to create a set of association rules. This rule base is then used to determine the training error or fitness of this set.[7] The genetic algorithms, of course, uses this fitness score for selective pressure (more information in Section 3.2).

5.5.1 Experiment

The aim of the following experiment is to evaluate how well the proposed method of finding the optimal set of borders (as well as the optimal set of individual borders) using genetic algorithms works.

The Khepera robot is allowed to run for a period of time of 100 seconds and one sample is gathered every 0.1 seconds, resulting in a total of 1000 samples (where

[7] The reader is reminded that a low training error corresponds to a high fitness score.

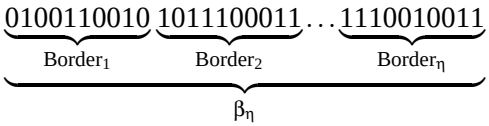


Figure 5.4: The chromosome is divided into a series of alleles, each coding for one border. Since each border is a value between zero and 1023, they are naturally expressed as ten bits binary numbers.

Table 5.2: The resulting performance for the manually programmed controller in Figure 5.2(b). From the high performance measured, it is clear that the set of borders found by the optimisation algorithm are (close to) the correct set.

Strength	#rules	e_{tr}%	e_{te}%	0rule%	1rule%	2rules%	>2rules%
1.00	5	0.0	0.4	0.0	85.3	14.7	0.0
0.95	7	0.1	0.5	0.0	82.2	14.8	3.0
0.90	10	1.2	1.8	0.0	0.7	25.9	73.4
0.85	10	1.2	1.8	0.0	0.7	25.9	73.4

each sample follows the general form specified in Section 5.3). This is repeated for each controller in Figure 5.2 as well as using the remote control to acquire more interesting data. The collected data is then split into two disjunct sets, the training set and the test set.[8]

The algorithm described in Section 5.6 is used to determine the optimal number of borders to use (the $\widehat{\eta}$ parameter) for each individual controller. Knowing the value of $\widehat{\eta}$ allows the optimisation algorithm to find the set of optimal borders for each data set.

Any set of borders may result in a large number of association rules, the strength attribute implies that some rules are more relevant than others. To remove and limit the number of rules a threshold value $\min_{strength}$ is selected, those rules with a strength value below the threshold are discarded.

For the first part of the experiment, the manually coded controllers found in Figure 5.2 are used. The result, after optimisation, for one case (namely the two border controller in Figure 5.2(b)) is summarised in Table 5.2 for several values of $\min_{strength}$.

The results include both the training error e_{tr} and the test error e_{te}, as well as a break down of the different cases occurring in the matching of the rules against the test set. The 0rule% summarises all the cases when no rule was found matching, causing the nearest neighbour algorithm to be invoked. The 1rule% are triv-

[8] It is important that the sets do not overlap or else the results will be biased.

ial cases, since the control action chosen only depends on one rule. The ratio of multiple matching rules are handled by the majority voting scheme proposed by Hellström.

The results are as expected, and matches quite well those obtained through a similar experiment by Hellström [14]. Keep in mind that in his experiments the set of optimal borders were all known before hand. The low training and test error indicates that the set of borders chosen were indeed the optimal set. Similar results are received for the other two controllers in Figure 5.2.

These results implies that our method of finding the set of optimal borders indeed works, and that it will be able to yield good results. The more interesting part of the experiment uses a remote control to allow an operator to directly guide the robot and showing it the behaviour it should learn.

In this case the operator tries to teach the robot a corridor following behaviour. Data is collected in the same manner as for the previous experiments, and the optimal number of borders $\widehat{\eta}$ is determined through the algorithm proposed in Section 5.6.

A subset of the resulting set of association rules for the corridor following behaviour is found in Table 5.3. Most rules relies heavily upon the rear sensors (see Figure 5.1) to determine the resulting control action.

The partitioning scheme found by the optimisation algorithm when $\widehat{\eta} = 3$ and $\min_{strength} = 1.00$ is:

$$ir_i = \begin{cases} 0 & \text{if} \quad 0 \leq IR_i \leq 352 & \text{(long distance)} \\ 1 & \text{if} \quad 353 \leq IR_i \leq 874 & \text{(medium distance)} \\ 2 & \text{if} \quad 875 \leq IR_i \leq 924 & \text{(short distance)} \\ 3 & \text{if} \quad 925 \leq IR_i \leq 1023 & \text{(very short distance).} \end{cases} \tag{5.6}$$

Table 5.4 shows the performance of the association rules for different threshold values of the strength attribute. The performance in this case is far lower than those obtained for the manually coded controllers. This is not surprising, since it is far more difficult for an operator to be consistent and always act in a reactive manner. In this case low performance is directly linked to the quality and skill of the operator, so low performance translates into a poor teacher. From the table it is easy to see that adding more rules does not seem to have any large effect on the test error (50 additional rules only manages to improve the performance by a small fraction).

Individual Borders

Intuitively the performance is expected to increase if each sensor is allowed its own set of borders, since the partitioning scheme of one sensor would not affect another sensor. This should also mean that fewer borders are required to attain high performance (that is a low fitness score).

These expectations are somewhat correct, as seen in Table 5.5. The previous experiment was repeated using individual borders and the performance recorded.

Table 5.3: Example of a constructed rule base for a remote controlled corridor following behaviour. This rule base was generated using a set of optimal borders with $\min_{strength} = 1.00$.

Rule	No.	Strength
$ir_0 = 3 \wedge ir_5 = 0 \wedge ir_7 = 0 \Rightarrow y = -1$	1	1.00
$ir_1 = 3 \wedge ir_5 = 0 \wedge ir_7 = 0 \Rightarrow y = -1$	2	1.00
$ir_0 = 1 \wedge ir_2 = 1 \wedge ir_4 = 0 \Rightarrow y = -1$	3	1.00
$ir_2 = 0 \wedge ir_4 = 1 \wedge ir_5 = 3 \wedge ir_7 = 0 \Rightarrow y = 0$	4	1.00
$ir_4 = 0 \wedge ir_5 = 0 \wedge ir_7 = 0 \Rightarrow y = -1$	5	0.98
$ir_3 = 0 \wedge ir_5 = 0 \wedge ir_7 = 0 \Rightarrow y = -1$	6	0.98
$ir_2 = 1 \wedge ir_5 = 0 \wedge ir_6 = 0 \Rightarrow y = -1$	7	0.97
$ir_1 = 1 \wedge ir_4 = 3 \wedge ir_6 = 0 \wedge ir_7 = 0 \Rightarrow y = 0$	8	0.95
$ir_1 = 1 \wedge ir_5 = 1 \Rightarrow y = -1$	9	0.94
$ir_4 = 0 \wedge ir_5 = 1 \wedge ir_7 = 1 \Rightarrow y = -1$	10	0.94
$ir_2 = 1 \wedge ir_5 = 0 \Rightarrow y = -1$	11	0.94
$ir_2 = 0 \wedge ir_3 = 0 \wedge ir_4 = 1 \wedge ir_7 = 0 \Rightarrow y = 0$	12	0.94
$ir_4 = 1 \wedge ir_5 = 3 \wedge ir_7 = 0 \Rightarrow y = 0$	13	0.94
$ir_2 = 1 \wedge ir_4 = 0 \wedge ir_6 = 0 \Rightarrow y = -1$	14	0.93
$ir_3 = 0 \wedge ir_4 = 0 \wedge ir_5 = 0 \wedge ir_6 = 0 \Rightarrow y = -1$	15	0.93
$ir_2 = 1 \wedge ir_4 = 0 \wedge ir_7 = 0 \Rightarrow y = -1$	16	0.93
$ir_0 = 1 \wedge ir_2 = 1 \Rightarrow -1$	17	0.93
$ir_3 = 0 \wedge ir_4 = 3 \wedge ir_6 = 0 \wedge ir_7 = 0 \Rightarrow y = 0$	18	0.93
$ir_1 = 1 \wedge ir_5 = 3 \wedge ir_6 = 0 \wedge ir_7 = 0 \Rightarrow y = 0$	19	0.93
$ir_0 = 3 \wedge ir_1 = 1 \wedge ir_4 = 3 \wedge ir_7 = 0 \Rightarrow y = 0$	20	0.93

Table 5.4: The table below shows the resulting performance for the set of optimal borders found for a remote controlled corridor following behaviour (in this case $\widehat{\eta} = 3$).

Strength	#rules	e_{tr}%	e_{te}%	0rule%	1rule%	2rules%	>2rules%
1.00	4	25.2	27.0	80.2	13.2	6.6	0.0
0.95	9	24.8	27.4	65.6	19.2	10.6	4.6
0.90	29	19.8	24.0	30.4	22.2	8.8	38.6
0.85	55	19.2	25.4	12.6	9.4	14.6	63.4

Table 5.5: Using individual borders yields similar performance as when using a single set for all sensors. It should be noted though, that the optimal number of borders decreases when using individual borders (in this case $\widehat{\eta} = 2$).

Strength	#rules	e_{tr}%	e_{te}%	0rule%	1rule%	2rules%	>2rules%
1.00	6	24.0	27.4	82.2	11.0	1.2	5.6
0.95	7	20.6	24.8	76.6	5.6	6.8	11.0
0.90	42	18.6	25.8	29.8	14.2	8.4	47.6
0.85	82	17.8	24.6	15.0	5.4	4.4	75.2

Comparing these results to those in Table 5.4 several things should be noted. The assumption that the number of borders should be lower using individual borders compared to one set of borders for all sensors is verified. Instead of three borders required for the previous experiment, using individual borders needs only two borders to reach maximum performance.

Furthermore, the training error is reduced and the test error is slightly better. These results are quite representative for all experiments done. However, it should be noted that using the first case has three borders to optimise while the second case has $8 \cdot 2 = 16$ borders. This causes the optimisation problem to become much harder and the resulting controller becomes much more complex.

5.6 Determining Border Count

The partitioning scheme (as mentioned before) is used to create the mapping between sensor readings and symbols which is later used in the rule discovery process. But in order for the partitioning scheme to be defined and found, the number of borders and therefore also the number of partitions have to be specified. In the case of hand crafted controllers (e.g. those listed in Figure 5.2) this may not be a problem, but in the case of very complex controllers or when using a remote control to guide the robot it is another matter.

Having too few borders will impede performance, since the resulting rule base will not be able to capture the details of the underlying behaviour. Consider the case of the controller in Figure 5.2(c) which depends on a partitioning scheme with four partitions. If only one border was used to discover the rules, it would mean that only part of its behaviour could be accurately described. This would also lead to poor performance. Therefore it is clear that the case of too few borders should be avoided.

However, simply using a "large" number of borders will not resolve this issue. Too many borders will result in a larger search space and will therefore require a higher computational effort to locate the set of optimal borders. Consider also the extreme case where we have one border for each possible sensor reading (i.e. 1024 borders for the sensor range of the Khepera), then an enormous rule base would be

required to capture all the possible combinations which could arise, as well as the need for a huge training set to create the rules.[9] The principle of Occam's razor[10] suggests that the number of borders used to create the rule base should be carefully considered.

To give the problem a more formal description, a suitable notation will be adopted. Define the function $\varphi : \mathbb{Z}^+ \rightarrow [0,1]$ as the function which determines the maximum fitness $\Phi(\widehat{\beta}_\eta)$ obtainable for a set of borders with size $|\widehat{\beta}_\eta| = \eta$ (i.e. the fitness for the optimal set with size η using Equation 5.4). The problem can then be formulated as:

$$\widehat{\eta} = \arg\max_{\eta \in \mathbb{Z}^+} \varphi(\eta) \tag{5.7}$$

Finding the solution to this problem can be done by utilising the following property of the fitness function:

$$\Phi(\widehat{\beta}_{i+1}) \geq \Phi(\widehat{\beta}_i), \qquad \forall i \in \mathbb{Z}^+. \tag{5.8}$$

Which we state and prove as a theorem:

Consider the set of borders $\widehat{\beta}_\eta = \{b_1, b_2, \ldots, b_\eta\}$ with the length η and the fitness $\Phi(\widehat{\beta}_\eta)$. Then a new set $\beta_{\eta+1}$ can be formed by adding a new element $b_{\eta+1}$ to the previous set β_η forming:

$$\beta_{\eta+1} = \widehat{\beta}_\eta \cup \{b_{\eta+1}\}. \tag{5.9}$$

If the new element $b_{\eta+1}$ is chosen such that $b_{\eta+1} \in \beta_\eta$ then Equation 5.9 will result in an empty partition,[11] which is equivalent to say that $\beta_{\eta+1} = \widehat{\beta}_\eta$, i.e. $\exists \beta_{\eta+1}\ \Phi(\beta_{\eta+1}) = \Phi(\widehat{\beta}_\eta)$, and maximising Φ with respect to $\beta_{\eta+1}$ can only yield greater values, that is $\Phi(\widehat{\beta}_{\eta+1}) \geq \Phi(\widehat{\beta}_\eta)$. Q.E.D.

By combining Equation 5.7 with Equation 5.4 (or Equation 5.5 for when individual borders are used), the total optimisation problem can be stated as:

$$\begin{aligned} \widehat{\eta} &= \min\left(\arg\max_{\eta \in \mathbb{Z}^+} \varphi(\eta)\right) \\ \widehat{\beta}_{\widehat{\eta}} &= \arg\max_{\beta_{\widehat{\eta}}} \Phi(\beta_{\widehat{\eta}}), \qquad \text{where } \beta_{\widehat{\eta}} \in [0, 1024]^{\widehat{\eta}}. \end{aligned} \tag{5.10}$$

In essence this problem consists of finding that set of borders $\widehat{\beta}_{\widehat{\eta}}$, of arbitrary length, that maximises the fitness function Φ.

[9] This is sometimes also referred to as "the curse of dimensionality", since the required size of the training set increases with the dimensionality of the problem.

[10] Occam's razor in this case states that using fewer borders is preferable when a more complex controller (using a higher number of borders) does not give any significant improvements upon performance.

[11] Consider the fact that if two identical borders exists, the partition has no extent and is therefore incapable of containing a sensor reading.

This problem may be solved by exploiting the fact that the theorem in Equation 5.8 is monotonically increasing, to construct a simple iterative algorithm to determine the optimal number of borders $\widehat{\eta}$.[12] By repeatedly training and evaluating the resulting fitness for increasing values of η, the optimal value $\widehat{\eta}$ together with the set of optimal borders $\widehat{\beta}_{\widehat{\eta}}$ can be determined as the value of η for which there is no additional increase in fitness, i.e. the maximum of the fitness function.

In practice this straightforward method should be modified slightly, since the optimisation algorithm is stochastic and is thus not guaranteed to find the global maxima. By aborting the iterative evaluation of η when the difference in fitness for two consecutive values of η is below a certain threshold τ, the algorithm will work as intended. This condition can more formally be specified as that the iterations are aborted when

$$\Phi(\widehat{\beta}_{\eta+1}) - \Phi(\widehat{\beta}_{\eta}) \leq \tau.$$

Determining which value of the threshold τ to use, should be based on the principle of Occam's razor. Setting the threshold to zero, is likely to lead to a (very) high value of $\widehat{\eta}$ and therefore a complex controller. Conversely, a higher threshold will lead to a smaller value of $\widehat{\eta}$ and a simpler controller. When using the error ratio as performance meassure it works quite well to set $\tau = 0.002$. For the general case, the value of τ is problem dependant.

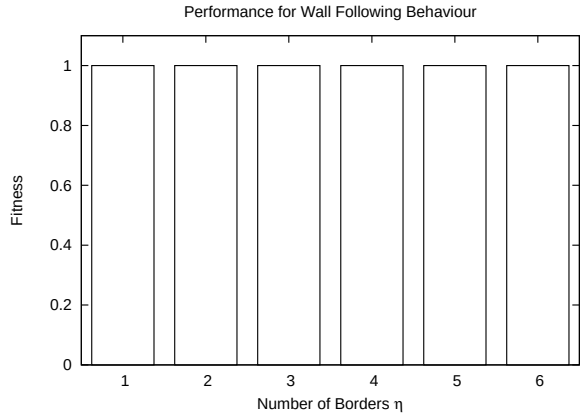
5.6.1 Experiment

To verify the claims made in the previous section regarding the choice of the optimal number of borders, the following experiment is performed. The Khepera robot is allowed to run for a period of time of 100 seconds, and one sample is gathered every 0.1 seconds, resulting in a total of 1000 samples (where each sample follows the general form specified in section 5.3). This is repeated for each controller in Figure 5.2, as well as using the remote control for additional data. The data collected is split into two sets, the training set and the validation set.

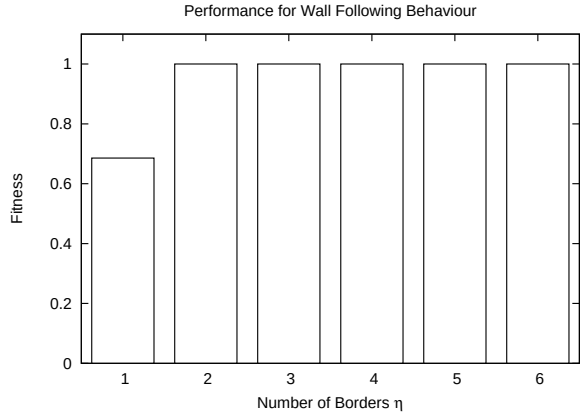
The algorithm proposed in the preceding section is used to determine the optimal number of borders for the controllers found in Figure 5.2. For increasing values of η the fitness of the set set of optimal borders is determined. The results are summarised in Figure 5.5. Since the optimisation algorithm is stochastic in nature, the performance is not guaranteed to be the same as the global optimum (causing the performance to vary a bit).

A more interesting example is that of a remote controlled behaviour, since the optimal number of borders $\widehat{\eta}$ is not known in advance. Figure 5.6 shows the resulting fitness for different values of η for a remote controlled corridor following behaviour (the same training and test set used in Section 5.5.1).

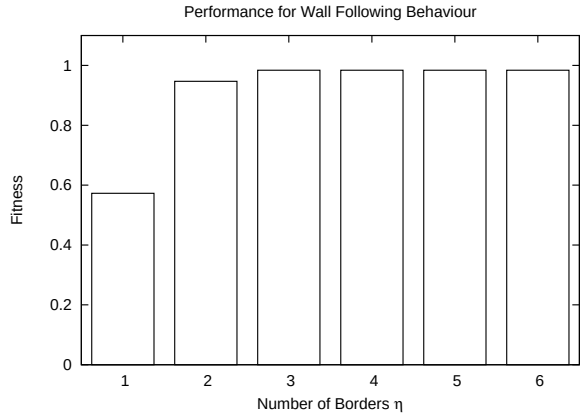
[12] It should be noted that, in our experience, relatively few borders, i.e. a low value of $\widehat{\eta}$, are required to maximise the fitness. Therefore a more advanced algorithm (e.g. branch and bound) is unlikely to provide any real benefit.



$\widehat{\eta} = 1$



$\widehat{\eta} = 2$



$\widehat{\eta} = 3$

Figure 5.5: The figures shows the fitness for the controllers in Figure 5.2. As expected the fitness varies with the number of border η as predicted by Equation 5.8. In the figure above $\widehat{\eta}$ denotes the correct number of borders to use (i.e. the number used in the actual controllers).

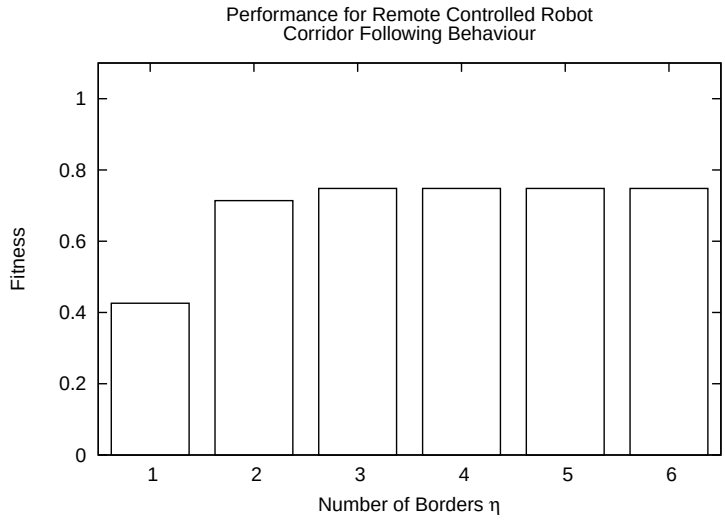


Figure 5.6: The fitness for a remote controlled behaviour (in this case a corridor following behaviour) for different values of η. In this case it appears that the optimal value is $\widehat{\eta} = 3$.

5.7 Fuzzified Borders

The set of association rules used to provide the behavioural mapping between the sensory input and the motor control action, has the unfortunate possibility of multiple rules triggering simultaneously. This raises the question of how the system should respond to this event. The obvious way to avoid this problem is simply to assign each rule a priority based on the strength and confidence of that rule. The control action of the rule with the highest priority is chosen.

A less naïve method was proposed by Hellström in [14], where a majority vote between the matching rules determined the final motor control action. Another means to handle this case would be to fuzzify the borders, causing the controller to more closely resemble that of a fuzzy controller. Depending on the membership functions that are used (e.g. a Gaussian membership function), this method would also provide a response for the case when no rules are triggered.

To relate this idea using the terms introduced in Chapter 4, what this system does is using the association rules to form the rule base used by the knowledge base. The database, containing the meaning of each term used in the rule base, is then modified to use fuzzy sets instead of the crisp sets used previous. This allows for each sensor reading to belong to several different partitions with a varying degree of membership.

There are a large number of considerations which are required in order for this to work, such as the choice of membership functions, the fuzzification and defuzzification functions used, the choice of disjunctions and conjunctions (and

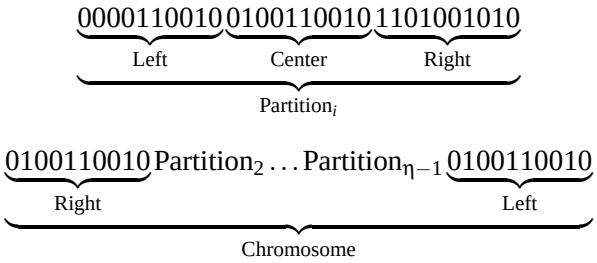


Figure 5.7: The chromosome structure used to encode triangular membership functions. Each allele corresponds to one edge- or centerpoint of a triangle. The two special cases for the edges where only one allele is required (since both zero and 1023 are implied) is also shown.

therefore also which implication function to use). The resulting performance of the system depend on the choices made regarding these functions and methods.

We shall continue to deploy genetic algorithms to perform the optimisation and tuning of the fuzzy sets. The encoding chosen will continue to be a binary Gray code, because the sensor range of the Khepera robot is limited to ten bits. Figure 5.7 shows the layout of the chromosome for triangular membership functions. Each fuzzy set is represented by a group of alleles corresponding to the respective parameters of the membership function. This means left, center and right for a triangular membership function and left, top left, top right and right for a trapezoidal membership function.

The position of each group of alleles corresponds to a particular partition. However, the position of each individual allele within a group has no special meaning. Repairing defect groups is important, since for a given sensor reading each membership function may only yield one degree of membership. Consider the case for triangular membership functions, where it is required that the center lies in between the left and right edges. To repair these defect groups, the individual alleles within a group will be sorted.

Depending on the defuzzification function chosen the resulting motor control action may be a real number (as opposed to an integer). This will invalidate the previous method of measuring performance, where the ratio of wrongly classified samples was used. For non-integer answers this measure does not apply. Instead the performance will be measured by the mean square error function as defined below.

Let $\tilde{y}_i$ denote the defuzzified result for the sample i, and let y_i be the correct motor control action as specified by sample i. Then the mean square error function is defined as:

$$e_{tot} = \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - y_i)^2, \tag{5.11}$$

Table 5.6: The two different performance measures yields slightly different results. The values can be said to be comparable, providing a means to understand how "good" the performance actually is.

Performance as mean square error

Strength	#rules	e_{tr}	e_{te}	0rule%	1rule%	2rules%	>2rules%
1.00	4	0.34	0.29	82.0	10.2	7.8	0.0
0.90	43	0.27	0.30	26.6	12.4	12.8	48.2

Performance as ratio of error

Strength	#rules	e_{tr}%	e_{te}%	0rule%	1rule%	2rules%	>2rules%
1.00	4	25.2	27.0	80.2	13.2	6.6	0.0
0.90	29	19.8	24.0	30.4	22.2	8.8	38.6

where n is the total number of samples.

To enable comparisons between fuzzified borders and Hellström's method (nearest neighbour when no rule matches and majority vote when multiple rules are triggered), new results will be computed using Equation 5.11 as performance measure.[13] Table 5.6 shows a summary of the results obtained from the same data sets (the remote controlled corridor follower) using the two performance measures. Since the results should be approximately equivalent, it allows for a more intuitive understanding of what a mean square error of 0.29 means (in this case this value corresponds to a error ratio of 27 percent).

Since the genetic algorithms seeks to maximise the fitness, and the objective function tries to minimise the mean square error (i.e. Equation 5.11) there is room for confusion. The fitness function, in this case, is simply the negative of the objective function. This can also be restated as trying to maximise $-e_{tot}$.[14] Because the sign of the fitness is always negative, it will be removed from the results.

5.7.1 Experiment

The purpose of the following experiment is to compare the performance of the fuzzified borders to that of the non-fuzzy controllers. Besides this comparison, we will also evaluate which combination of algorithms and methods (such as the choice of membership function and defuzzification method) that yields the best performance (i.e. the highest fitness).

In this experiment the data gathered previously, through the use of both the controllers in Figure 5.2 as well as the remote controlled behaviour, is reused. The

[13] Note that a different performance measure may result in different results, i.e. the set of optimal borders might become a different set.

[14] In general, the genetic algorithms do not work well with a negative fitness. However, since we are using rank scaling (see Section 3.2.1) the sign of the fitness will not matter.

data consists of 1000 samples, split into two disjunct sets, one set each for training and testing. The remote controlled behaviour is again the corridor following behaviour used before.

As was seen in Section 4 there are many different methods to choose from when setting up the fuzzy controller. Changing how disjunction and conjunction is interpreted or changing the defuzzification method will likely affect the resulting performance of the system. Another important aspect is whether the strength of the rules should be factored in or not, since intuitively it would seem that a low strength should be reflected in the magnitude of the control action. Choosing the right membership function is another important task.

In this experiment the following parameters are examined:

- Intersection/Union versus Algebraic Product/Algebraic Sum.[15]
- COA for singletons versus Max criterion method.
- Scaling by the strength of the rules versus no scaling.
- Choice of membership function between triangular, trapezoidal or Gaussian.

The reader should keep in mind that for many membership functions the problem of what should be done when no rule matches is not guaranteed to be removed. Using fuzzy membership functions only provides an alternative to majority voting between multiple matching rules (in fact it is a weighted voting). However, it should be pointed out that using the Gaussian membership function does indeed guarantee that there will always be multiple rules matching since the Gaussian is nonzero for every value.[16]

To evaluate the different methods (and combinations thereof) we shall look at both the maximum fitness obtained as well as the mean fitness of a large number of solutions. It can be argued that the "best" method is the one that consistently produces good solutions, i.e. has a high mean fitness. The reason for this is quite clear under the assumption that the samples are normally distributed, then there is always a certain (very low) probability of achieving good results irregardless of method used. A high mean fitness translates into an algorithm that does not require as much "luck" as the others to find a good solutions, which also implies that less computational effort is required to find good solutions.

To perform the evaluations of the methods the Student's t-test will be used.[17] This method compares the means of two distributions and is able to determine if they are equal. Unfortunately this test also requires the samples to be normally

[15] See Table 4.2 for a definition of these concepts.

[16] In practice this is not the case. Depending on the parameters of the Gaussian function, it is possible for the function to reach zero. This is caused by lack of precision in the calculation and round off errors. When this happens, the knn method introduced earlier will be used.

[17] Details regarding the Student's t-test can be found in most handbooks on statistics, for instance in [20].

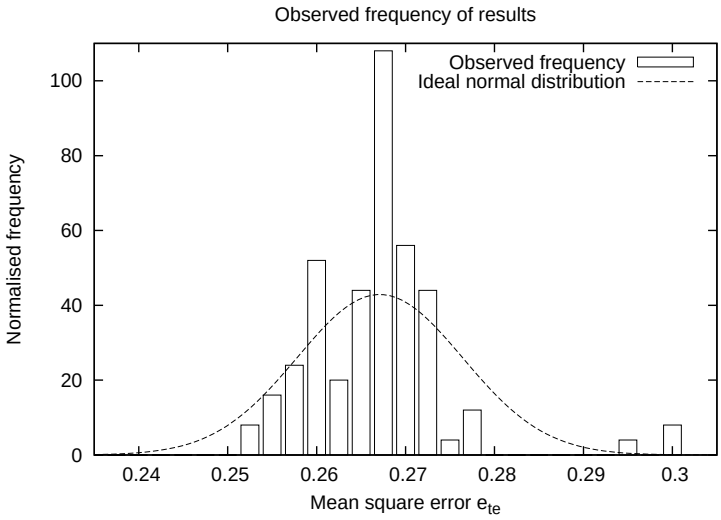


Figure 5.8: The histogram shows the normalised frequency of results obtained from the experiments. Even though the match between the histogram and the added ideal normal distribution is somewhat poor, the general shape of the histogram suggests that it is normally distributed.

distributed. However, according to the central limit theorem:

> "If a random sample of n observations, $y_1, y_2, \ldots, y_n$, is drawn from a population with finite mean μ and variance σ^2, then, when n is sufficiently large, the sampling distribution of the sample mean $\bar{y}$ can be approximated by a normal density function." – Mendenhall et al. [20]

That the above quote applies to this case is illustrated in Figure 5.8, where a histogram of the observed results are shown together with a curve representing the ideal normal distribution. The somewhat poor match between the histogram and the curve is likely caused by estimation of the parameters (i.e. the mean and standard deviation) from the measured samples (as opposed to the *real* distribution).

The non-fuzzy partitions can quite easily be interpreted as "long distance" or "short distance" depending on the sensor range they cover. When using fuzzified borders, the results after optimisation may not be as easily interpretable. An example of this is seen in Figure 5.9 where triangular membership functions have been optimised to yield the highest fitness. As the figure illustrates there exists a degree of ambiguity regarding which membership functions corresponds to a "medium distance" or a "long distance". This is an unfortunate consequence of the optimisation.

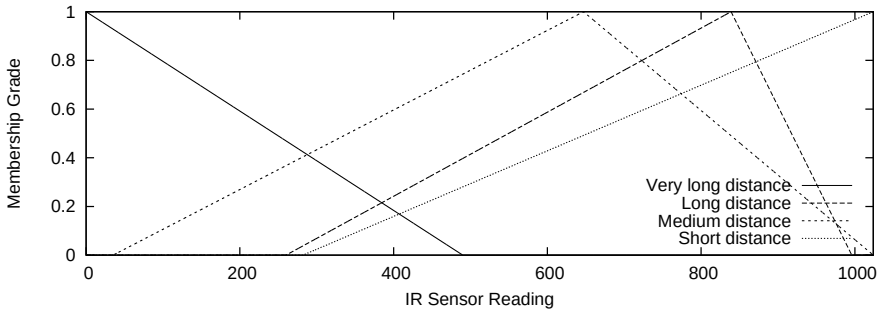


Figure 5.9: The resulting fuzzy partitions created by the triangular membership functions (in actuality any fuzzy membership function would suffice) may not be as readily interpretable as their non-fuzzy counterparts. There exists some ambiguity regarding which membership function should be regarded as “long distance” or “medium distance”.

Comparing Fuzzy Methods

Table 5.7 show the result of several combinations of the methods, both the optimal fitness e_{te} as well as the mean fitness $\mu_{e_{te}}$ and standard deviation $\sigma_{e_{te}}$ are shown. In this case the threshold for the strength attribute was set at 0.90, which creates a rule base consisting of 43 rules.

From a visual inspection the different methods seem relatively equivalent, but there is still some interesting results that can be obtained (these results are representative for all the different data tested).

Determining whether scaling the degree of membership by the strength of the respective rule is examined first. Our null hypothesis is $H_0 : \mu_1 = \mu_2$ and the alternative hypothesis becomes $H_1 : \mu_1 \neq \mu_2$. This means that the test will determine whether the mean fitness of the two methods are equal or not. To acquire a relatively high degree of certainty in our results, the significance level is chosen as $\alpha = 0.01$.[18]

The test is performed between each pair of methods that differ only in the usage of scaling or not (e.g. using a triangular membership function and intersection as conjunction, there is one entry where scaling is used and one when it is not).

For each of these six cases, the test clearly indicates that there is no significant difference between the two means. This forces us to accept the null hypothesis, and conclude that scaling the degree of membership with the strength of the rules does not affect the results (at least with a confidence of 99%).

Regarding using the algebraic product over intersection as the meaning of a conjunction, the results are more uncertain. For the case in Table 5.7 both the triangular and trapezoidal membership functions benefits from using the algebraic

[18] In one percent of the cases when the t-test asserts that the null hypothesis holds, it will be in error.

Table 5.7: The fitness for a large number of different sets of parameters are examined and shown below. As seen, the results are quite similar and on par with those obtained from using crisp borders and majority voting directly (compare with Table 5.6).

Scale	Intersect	Algebraic product	Membership function	e_{te}	$\mu_{e_{te}}$	$\sigma_{e_{te}}$
√	√		Triangular	0.2667	0.2880	0.0219
	√		Triangular	0.2627	0.2802	0.0183
√		√	Triangular	0.2591	0.2734	0.0170
		√	Triangular	0.2587	0.2730	0.0161
√	√		Trapezoidal	0.2713	0.2853	0.0079
	√		Trapezoidal	0.2712	0.2822	0.0063
√		√	Trapezoidal	0.2610	0.2802	0.0102
		√	Trapezoidal	0.2685	0.2801	0.0092
√	√		Gaussian	0.2634	0.2712	0.0092
	√		Gaussian	0.2639	0.2735	0.0141
√		√	Gaussian	0.2535	0.2671	0.0098
		√	Gaussian	0.2555	0.2671	0.0089

product, while for the Gaussian there is no statistical difference. However, these results do not seem to hold for the general case. Using a different threshold value for the strength attribute (changing the number of rules) produces different results.

Trying to determine which of the three membership functions yields the best results, has similar problems as trying to differentiate between the two forms of conjunctions. Inconsistent results depending on the use of the minimal value of strength allowed. It is therefore likely that some methods work better with a low number of rules, and others when the number of rules increases.

Fuzzy versus Crisp

The original method proposed by Hellström [14] used a combination of nearest neighbour and majority voting to handle the different cases that might arise. The comparison between the two approaches of using majority voting and using fuzzified borders will also make use of the Student's t-test. The null hypothesis is as before $H_0 : \mu_1 = \mu_2$, however since we are trying to determine which of the two methods are superior to the other the alternative hypothesis becomes $H_1 : \mu_1 < \mu_2$.[19]

Comparing the results from Table 5.6 with Table 5.7, it is obvious that the two methods tends to yield quite similar results for the maximum fitness. When using the non-fuzzy method the best result becomes $e_{te} = 0.30$ compared to $e_{te} \approx 0.26$ for the fuzzified borders.

[19] A single tailed test is used to determine if one mean is higher or lower than the other.

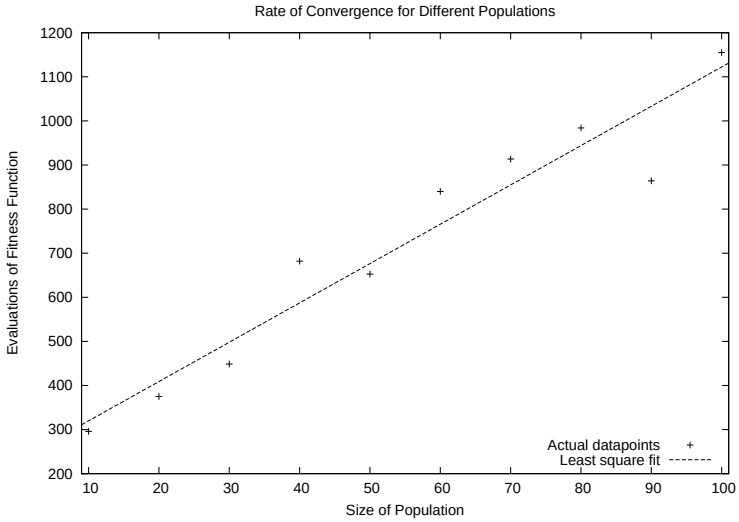


Figure 5.10: The rate of convergence, measured as the required number of evaluations of the fitness function to reach a certain threshold, increases as the population increases. A least square fitted line is also shown to help clarify the trend.

Even though the approach of using fuzzified borders produces an improvement, it may not be that important because of the arguments presented earlier. These arguments can be summarised as the method which consistently produces better results is superior, i.e. has a lower mean error (which is equivalent of a high mean fitness).

However, the conclusion that the fuzzified borders are better is supported by the significant difference between the two means. The non-fuzzy method yields a mean of $\mu_{te} = 0.3009$, whereas the fuzzified algorithms yields (according to Table 5.7) values around $\mu_{te} \approx 0.28$. This is also confirmed by the Student's t-test, since it rejects the null hypothesis in favour of the alternative one (the same $\alpha = 0.01$ as before is used).

Therefore using the fuzzified borders produces better results than using the crisp equivalent. Unfortunately, this result is skewed in favour of the method using the fuzzified borders. Since we are using the results from the non-fuzzy borders as a starting point, there exists a certain bias which makes it slightly easier to find a good set of fuzzified borders.

5.8 Choosing Algorithms and Parameters

To achieve a fairly high rate of convergence for the genetic algorithms a large number of possible sets of parameters were considered. For the first optimisation task, finding a optimal set of (crisp) borders, a good value of the mutation rate

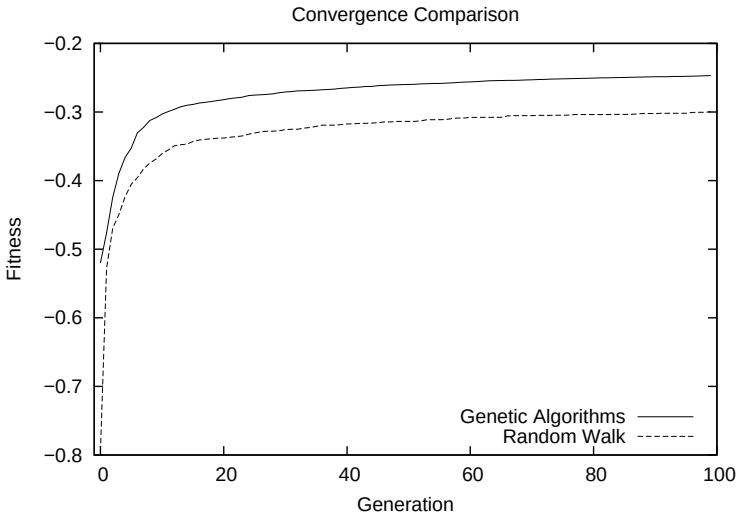


Figure 5.11: Comparing the rate of convergence between the genetic algorithms and a random search clearly shows that the genetic algorithms are superior. For the random search, one generation equals to ten evaluations of the fitness function.

was found to be around $p_m = 0.09$. Rather surprisingly, recombination did not significantly improve the rate of convergence for this particular problem.

The size of the population was set quite low, ten individuals seemed to produce the best results. Figure 5.10 shows that increasing the population size beyond this value did not improve the rate of convergence. The rate of convergence was measured as the number of evaluations of the fitness functions, while counting the number of generations is a possibility it is a (in our opinion) suboptimal measure. The reason is simply that the computational effort does not lie in producing new generations but rather in the evaluation of the fitness function, a low number of evaluations therefore translates into a quicker convergence.

Similar values of both the size of the population and the mutation rate was found to work well for the second optimisation problem (i.e. optimising the membership functions for the fuzzified borders). A high value of the crossover rate $p_c = 0.75$ gave consistently better results compared to other values.

These values are of course purely empirical and since it is very difficult to test all combination of values of the parameters together with the large number of possible algorithms, they can at best be described as “good enough”.

By using a Gray code representation instead of a pure binary one, the rate of convergence was again found to be clearly better. This result is not that surprising given the arguments in Section 3.1.

For the genetic algorithms there are a number of (sub) algorithms that had to be used. For the selection the SUS algorithm (see Section 3.2 for details) was found to work quite well, especially together with an elitistic strategy of combining the par-

ent population with the generated offspring. The scaling of the fitness values was performed with a linear rank scaling and a two-point crossover was used. These algorithms produced reasonably good results.

Comparing the rate of convergence and the results achieved by a purely random search, the genetic algorithm tended to perform better. This is illustrated in Figure 5.11, where the maximum fitness of each generation is shown for both the genetic algorithms and a random search. When the size of the problem increased (i.e. the number of dimensions increased) the genetic algorithm became far superior the the random search (this holds especially true for finding the fuzzified borders). Comparisons were also made against an implementation of the DIRECT-algorithm [3] for optimisation, and while it performed on par or better than the genetic algorithms for many problems it had sever difficulties handling cases when the dimensionality of the problem increased beyond 10 dimensions.

Chapter 6

Summary and Conclusions

Borrowing concepts of fuzzy logic controllers to enhance the association rules is a step that intuitively should improve the performance of the system. However, comparing the empirical results found in Chapter 5 with those obtained using a simple majority vote between all of the matching rules, several things can be determined.

The scaling of the membership grade by the strength of each rule, does not yield any change in performance. All of the different combinations of the parameters tested seems to yield similar results to each other, the improvement in performance is negligible. This can be ascertained with a high degree of certainty (the significance level of the tests were $\alpha = 0.01$).

A direct comparison between the fuzzified borders and the use of majority voting does indeed indicate that a higher mean performance results from the use of fuzzy set theory (again with a certainty of 99%). Unfortunately, the minimum fitness is very similar and from this the only conclusion that can be made is that the use of majority voting is superior to that of using fuzzy borders. Even though the performance appears *slightly* better it does not motivate the increased computational effort as well as the increased complexity of the resulting system.

The performance of the association rules is highly dependant upon how the input space is discretized. The discretization is problem dependant, and determining how this should be done (optimally) is an important task. Determining the partitioning scheme required for the mapping between the raw sensory information to the symbols used by the rule discovery process, is dependant on two factors; the optimal border count and the set of optimal borders.

By exploiting the properties of the fitness function (as it depends on the border count), a simple iterative method for determining the optimal border count was proposed. Both the theoretical proof and empirical experiments strongly suggests that this method works well in practice, allowing the system to determine the best value to use.

Finding the optimal border count is linked with the problem of finding the set of optimal borders, since they both requires a method of maximising the fitness function for any given number of borders. The solution to this optimisation prob-

lem made use of genetic algorithms, and from the experimental results in Chapter 5 we conclude that this approach works quite well. By utilising the known partitioning scheme of the manually coded controllers in Figure 5.2, it becomes obvious that the optimisation process quickly is capable of locating the correct set. This, of course, gives credence to quality of the borders found for the remote controlled behaviours.

By using these two methods, it is clear that the system is capable of autonomously providing the solutions required in order for the discretization of the input space to take place. By extension, this also allows the system to autonomously extract the association rules from the data recorded during the demonstration.

Chapter 7

Acknowledgements

I would like to thank my supervisor Thomas Hellström at the Department of Computing Science, Umeå University for his support during the completion of this project. I also want to thank Jonas Borgström for proof-reading this thesis.

Bibliography

[1] AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. N. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., 26–28 1993), P. Buneman and S. Jajodia, Eds., pp. 207–216.

[2] AGRAWAL, R., AND SRIKANT, R. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB* (12–15 1994), J. B. Bocca, M. Jarke, and C. Zaniolo, Eds., Morgan Kaufmann, pp. 487–499.

[3] BJÖRKMAN, M., AND HOLMSTRÖM, K. Global optimization using the DIRECT algorithm in matlab. *Advanced Modeling and Optimization 1*, 2 (1999).

[4] BORGELT, C. Apriori – Association Rule Induction / Frequent Item Set Mining. Webpage, nov 2004. `http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html`.

[5] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, T., Eds. *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Publishing Ltd., 2000.

[6] CORDONG, O., AND HERRERA, F. A general study on genetic fuzzy systems. In *Genetic Algorithms in Engineering and Computer Science* (1993), John Wiley & Sons, pp. 33–57.

[7] DEB, K. Encoding and decoding functions. In *Evolutionary Computation 2: Advanced Algorithms and Operators* (2000), T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., pp. 4–11.

[8] DILLMAN, R., ROGALLA, O., EHRENMANN, M., ZÖLLNER, R., AND BORDEGONI, M. Learning robot behaviour and skills based on human demonstration and advice. In *Proceedings of the 9th International Symposium of Robotics Research (ISRR '99)* (Oct 1999), pp. 229–238.

[9] EFSTATHIOU, J. Rule-based Process Control Using Fuzzy Logic. In *Approximate reasoning in intelligent systems, decision and control* (1987), E. Sanches and L. A. Zadeh, Eds., pp. 145–158.

[10] FOGEL, D. B., AND ATMAR, J. W. Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics 63*, 2 (1990), 111–114.

[11] FREITAS, A. A. Understanding the crucial differences between classification and discovery of association rules: a position paper. *SIGKDD Explorations Newsletter 2*, 1 (2000), 65–69.

[12] FULLER, R., AND ZIMMERMANN, H. On Zadeh's compositional rule of inference. In *Proceedings of Fourth IFSA Congress, Vol. Artificial Intelligens* (Brussels, 1991), R. Lowen and M. Roubens, Eds., pp. 41–44.

[13] GRIMALDI, R. P. *Discrete And Combinatorial Mathematics*, fourth ed. Addison-Wesley, 1999.

[14] HELLSTRÖM, T. Association rules for learning behavioral mappings in robotics. Technical report UMINF 03.12. ISSN-0348-0542, Department of Computing Science, Umeå University, SE-901 87 Umeå, Sweden, sep 2003.

[15] K-TEAM. Khepera user manual. Webpage/PDF, feb 2005. `http://www.k-team.com/download/khepera/documentation/-KheperaUserManual.pdf`.

[16] KLEMENT, P., AND SLANY, W. Fuzzy Logic in Artificial Intelligence. *Christian Doppler Laboratory Technical Reports 67* (1994).

[17] KLIR, G. J., AND FOLGER, T. A. *Fuzzy Sets, Uncertainty, and Information*. Prentice Hall, 1988.

[18] LEE, C. C. Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part I. In *IEEE Transactions on Systems, Man and Cybernetics* (March/April 1990), vol. 20, pp. 404–418.

[19] LEE, C. C. Fuzzy Logic in Control Systems: Fuzzy Logic Controller — Part II. In *IEEE Transactions on Systems, Man and Cybernetics* (March/April 1990), vol. 20, pp. 419–435.

[20] MENDENHALL, W., AND SINCICH, T. *Statistics for engineering and the sciences*, fourth ed. Prentice-Hall, 1995.

[21] MUNAKATA, T., AND JANI, Y. Fuzzy systems: an overview. *Communications of the ACM 37*, 3 (1994), 68–76.

[22] MURPHY, R. R. *Introduction to AI Robotics*. MIT Press, 2000.

[23] PHILLIPS, C., KARR, C. L., AND WALKER, G. W. Helicopter flight control with fuzzy logic and genetic algorithms. In *Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives* (1997), E. Sanches, T. Shibata, and L. A. Zadeh, Eds., World Scientific Publishing Co. Pte. Ltd., pp. 1–18.

[24] RUNKLER, T., AND GLESNER, M. DECADE – fast centroid approximation defuzzification for real time fuzzy control applications. In *SAC '94: Proceedings of the 1994 ACM symposium on Applied computing* (1994), ACM Press, pp. 161–165.

[25] WEBB, G. I. OPUS: An efficient admissible algorithm for unordered search. *Journal of Artificial Intelligence Research 3* (1995), 431–465.

[26] WEBB, G. I., AND ZHANG, S. Beyond association rules: Generalized rule discovery. Submitted for publication, 2003.

[27] ZADEH, L. The concept of a linguistic variable and its application to approximate reasoning–I. In *Fuzzy Sets and Applications* (1987), R. Yager, S. Ovchinnikov, R. Tong, and H. Nguyen, Eds., Wiley, pp. 219–269.

[28] ZADEH, L. The concept of a linguistic variable and its application to approximate reasoning–II. In *Fuzzy Sets and Applications* (1987), R. Yager, S. Ovchinnikov, R. Tong, and H. Nguyen, Eds., Wiley, pp. 271–327.

[29] ZADEH, L. The concept of a linguistic variable and its application to approximate reasoning–III. In *Fuzzy Sets and Applications* (1987), R. Yager, S. Ovchinnikov, R. Tong, and H. Nguyen, Eds., Wiley, pp. 329–366.

[30] ZADEH, L. Fuzzy sets. In *Fuzzy Sets and Applications* (1987), R. Yager, S. Ovchinnikov, R. Tong, and H. Nguyen, Eds., Wiley, pp. 29–44.

Appendix A

Notation

p_m	The rate of mutation.
p_c	The probability of recombination taking place.
η	The number of borders, i.e. the size of β_η
$\widehat{\eta}$	The optimal number of borders, i.e. the size of $\widehat{\beta}_{\widehat{\eta}}$
β_n	The set of borders, where $\lvert\beta_n\rvert = n$.
$\widehat{\beta}_n$	The optimal set of borders with size n.
$\mathcal{B}$	The set of sets of borders.
$\widehat{\mathcal{B}}$	The set of sets of *optimal* borders.
$\Phi(\beta_\eta)$	The fitness associated with the given set of borders of length η.
$\varphi(\eta)$	The optimal fitness for any set of borders of length η.
y	The (crisp) motor control action.
$\widetilde{y}$	The defuzzified motor control action.
e_{te}	The mean square error of the test set.
$\mu_{e_{te}}$	The mean value of a large number of e_{te}.
$\sigma_{e_{te}}$	The standard deviation of a large number of e_{te}.