Bar-Ilan University
Department of Computer Science

ANY-TIME FUZZY CONTROLLER

by

Edward Shmukler

Advisor: Dr. Gal Kaminka

Submitted in partial fulfillment of the requirements for the Master's degree
in the Department of Computer Science

Ramat-Gan, Israel
June 2006

This work was carried out under the supervision of

Dr. Gal A. Kaminka

Department of Computer Science, Bar-Ilan University.

Abstract

Fuzzy logic has been successfully applied in various fields. However, as fuzzy controllers increase in size and complexity, the number of control rules increases exponentially and real-time behavior becomes more difficult. This thesis introduces an any-time fuzzy controller. Much work has been done to optimize and speed up a controlling process, however none of the existing solutions provides an any-time behavior. This study first introduces several constraints that should be satisfied in order to guarantee an any-time behavior. These constraints are related to aggregation and defuzzification phases of fuzzy control. Popular aggregation and defuzzification methods (max-min, sum-product, MOM and COG) are first shown to satisfy these constraints, and then three linearization methods are presented. Linearization methods are used to reorder fuzzy rules base such that a reordered rule base would result in any-time behavior. Finally, several approximation methods are described, that do not break any-time behavior, while causing the intermediate result of an any-time controller to come closer to the final (full calculation) result in a shorter time. The exact influence and worthiness of approximation methods should be further researched.

Acknowledgments

I would like to thank my advisor Dr. Gal A. Kaminka for giving me the opportunity to be part of the MAVERICK lab. Without his brilliant guidance, this thesis could not have been written. He accompanied me every step of the way, supported my research with his vast knowledge and helped me formulate my ideas into coherent research.

Dr. Kaminka is a great research partner with whom it was always a pleasure to work.

I am also deeply grateful for the love and encouragement of my parents, Aleksandra and Solomon Shmukler, for their moral support along the way.

Last but not least Slava, my wife, and my son, Matan, for their love and patience and for giving me the courage to make this journey.

Contents

List of Figures

Chapter 1

Introduction

Fuzzy control is used in a wide variety of applications, ranging from controlling of oxygen delivery [17] to robotic navigation while avoiding obstacles [24]. A fuzzy controller relies on a set of conditional input-output rules, which are all fired in parallel.

All rules whose left-hand-side condition matches the input (through a process of *fuzzification*) are aggregated together and their collective right-hand-side determines the controller output (through a *defuzzification* process).

Many applications of fuzzy control can be sensitive to the time it takes for the controller to return a result. In the case of an obstacle-avoiding navigation control, for instance, a controller ideally has enough time to calculate the best way to avoid an obstacle, but in practice this is not always possible. Sometimes, when the robot moves too fast, there is not enough time to perform a full calculation. In such cases, the controller may fail to provide its function.

Existing approaches to handling this challenge include specialized hardware [7], [11], [16], [12] and [14], changing the environment to suit the controller speed (i.e., slowing down the robot in our case), reducing the number of rules in the controller (e.g., by re-designing the controller hierarchically [23]) and simplifying membership function or fuzzy set shape. Such re-design is not always possible, of course, and may degrade the performance of the controller. This is especially true in cases where the controller is automatically generated (e.g., [4] and [26]), or is optimized by learning (e.g. [8] and [17]).

A different approach is proposed here to handle real-time requirements, which is to change the controller such that it works in an any-time fashion. This means that for any given amount of time of operation, the controller is (1) guaranteed to provide a result, and (2) guaranteed to provide a monotonically-improving result (compared to results obtained using less time). Put a different way, the results of the controller improve monotonically with time.

In robot navigation the advantage of using an any-time controller is clear. For example, consider situation where there is insufficient time for performing a full calculation because of close obstacle. In that case any fast decision (even not an optimal one) that causes individuals to avoid an obstacle, is better then waiting for an optimal solution that may come too late.

Fuzzy control has three main phases: Fuzzification, implication with aggregation and defuzzification. Work described in this thesis addresses the two last phases — aggregation and defuzzification. In particular, this study presents a way to reorder and restructure fuzzy-control rule-bases such that these two steps result in an any-time fuzzy controller (FC), under the assumption of negligible fuzzification time. The re-structuring of the rule-base must take into account both defuzzification and aggregation. The thesis proposal provided results for max-min aggregation with center-of-gravity (COG) defuzzification, and proposed creating similar methods for popular alternatives: Sum-product aggregation and mean-of-maxima (MOM) defuzzification.

Chapter 2

Background and Motivation

2.1 Background

Fuzzy Set Theory is a popular extension of regular set theory [31]. In classical set theory an element X is or is not a member of a set. In fuzzy set theory, an element has graded membership. The element may partially belong to the set, where the degree of membership is determined by a real number, typically in the range $[0, 1]$; 0 denotes no membership in the set, while 1 denotes full membership.

For example, we may define the fuzzy set *cold* as follows: $0C^\circ$ and less with a membership of 1; $50C^\circ$ with a membership of 0; and for $t \in (0C^\circ, 50C^\circ)$, membership $\mu(t)$ defined by the function $\mu = 1 - t/50$. Now, what can be said about $20C^\circ$? One can assert that the membership degree of cold of $20C^\circ$, is 0.6. See figure 2.1.

Fuzzy control [15] is a control methodology based on fuzzy set theory. It

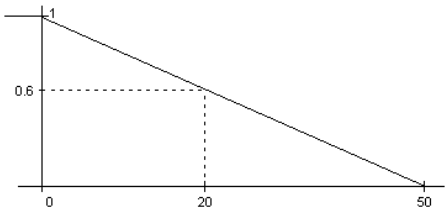


Figure 2.1: Graded membership example: The fuzzy set *cold*.

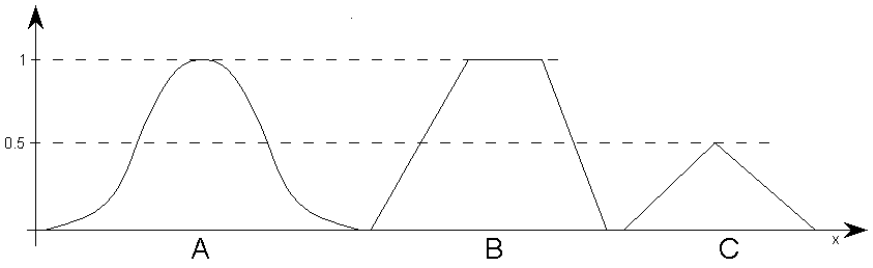


Figure 2.2: Fuzzy set example

is composed of three stages that rely on a set of control rules, stored in a rule-base. Each such rule tests an input for its membership degree in a set of fuzzy sets (the *left-hand-side*, or *LHS*), and associates with the result fuzzy sets that are defined on the output value (the *right-hand-side*, or *RHS*). All rules in the rule-base are matched in parallel, and their collective RHS is used to compute the final controller value. The process is then repeated for new inputs.

A short overview of fuzzy set theory and fuzzy control is going to be provided. Additional details can be found in [15] and in [1, 29, 30].

2.1.1 Fuzzy set

A fuzzy set A is given by a membership function $\mu(x)$ from universe of discourse U which is a non fuzzy set to the range $[0, 1]$

$$A = \{(x, \mu_A(x))|x \in U\}$$

where $\mu_A(x)$ is a membership function, which defines a membership degree of value x. See sample fuzzy sets in figure 2.2.

The *Height* of fuzzy set A is

$$h = \{\max_X(\mu_A(x))|x \in X\}$$

The height of sets A and B in figure 2.2 is 1, and height of set C is 0.5.

Like mentioned above fuzzy control has three main phases: Fuzzification, implication with aggregation and defuzzification.

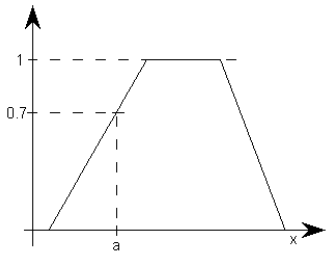


Figure 2.3: Fuzzification example

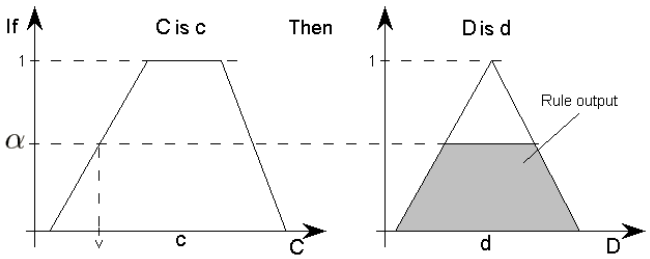


Figure 2.4: Implication example

2.1.2 Fuzzification

Fuzzification is a first phase in control sequence. At this phase crisp values from universe of discourse U are translated to values in range $[0, 1]$ that fuzzy controller can handle. That is done by looking up a membership value in the set. For example see figure 2.3. Here crisp value a is translated to 0.7.

2.1.3 Implication

At this phase result of each individual rule is determined. Assuming next example rule: If C is c, Then D is d (see figure 2.4). Assuming input crisp value v, α is a result of fuzzification. Output of fuzzification is used to tweak an RHS set of the rule. Output of the rule is a set represented by filled area in figure 2.4 and defined by:

$$\mu_{A'} = \min_{A}(\mu_A(x), \alpha)$$

In this example C is an input variable and D is an output variable.

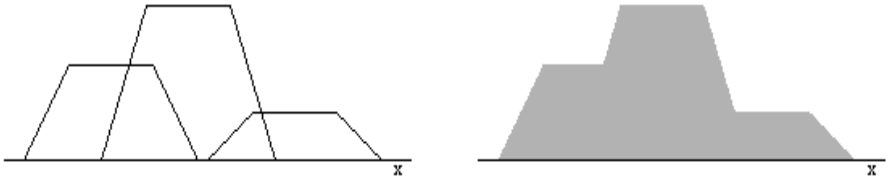


Figure 2.5: Max-min example

2.1.4 Aggregation

Aggregation is a phase where tweaked RHS output sets from various rules are combined for each output variable. Two aggregation methods are addressed in this thesis: Max-min (max — aggregation and min — implication) and sum-product (sum — aggregation and product — implication).

Max-min

In max-min aggregation, union of implication results is calculated. For example of max-min aggregation see figure 2.5. Formally according to [15], max-min aggregation method is defined by:

$$\mu_{A'}(x) = \overbrace{\max_{k}}^{aggregation} \underbrace{\min_{A_k}(\alpha_k, \mu_{A_k}(y))}_{implication}$$

This aggregation method is relatively simple for implementation, but in contrast, it looses information in intersections of fuzzy sets because of nature of union.

Sum-product

In contrast to max-min aggregation, sum-product does not lose information. It is implemented by summation of fuzzy sets instead of union. Sum-product aggregation is defined by:

$$\mu_{A'}(y) = \sum_{k} \alpha_k \mu_{A_k}(y)$$

For example of sum-product aggregation see figure 2.6.

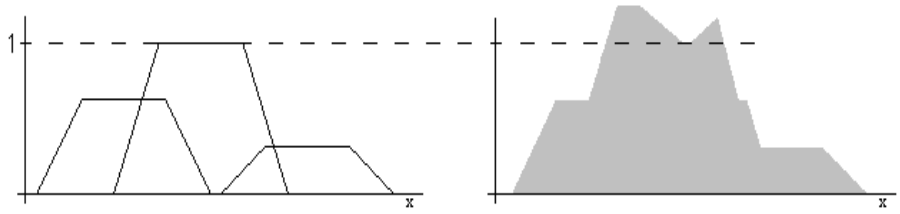


Figure 2.6: Sum-product example

2.1.5 Defuzzification

Defuzzification is a last phase of fuzzy control. In this phase, a crisp value for each output variable is calculated. Input of this phase is an output of aggregation. Two defuzzification methods are addressed in this thesis: mean-of-maxima (MOM) and center-of-gravity (COG).

Mean-of-maxima

MOM of set A' is a value, which is a mean of all values, where membership function $\mu_{A'}(y_i)$ $(i = 1, 2, ..., N)$ reaches its global maximum. MOM is defined by:

$$MOM(A) = \frac{\sum_1^N y_i}{N}$$

MOM is influenced by relatively small (usually one) fuzzy set.

Center-of-gravity

COG calculates a value which is a center of gravity of fuzzy sets. In case of two-dimensional fuzzy sets (our case) COG is equivalent to center-of-area. COG is defined by:

$$COG\mu(x) = \frac{\int_x x\mu(x)dx}{\int_x \mu(x)dx}$$

2.2 Motivation

In the theoretical model of fuzzy controller all rules are handled simultaneously and the final result is calculated. In practice, actually, rules are handled by small number of processors — usually one, as a result fuzzy rules are handled sequentially. To provide parallel processing of fuzzy rules, a processor for each fuzzy rule or specialized hardware should be supplied [14] that is capable of handling rules simultaneously.

The calculation time in sequential processing depends on the size of the fuzzy rules database and the amount of available resources. Thus a big rule base and lack of resources can cause a (relatively) long calculation time.

It is this process of matching the rules and aggregating the results that is addressed in this thesis. In general, the controller's operating speed is governed by the time it takes to match all rules, compute their individual RHS and then compute a final controller output. If insufficient time is given to the controller, the process will stop in the middle and an incorrect output value may result.

Example 1:

If in the next environment there are:

- Two output variables P_1, P_2
- Three rules R_1, R_2 and R_3 that modify these variables
- Three fuzzy values of each variable: for P_1 - V_{11}, V_{12} and V_{13}, for P_2 - V_{21}, V_{22} and V_{23}, such that $V_{ij} < V_{i,j+1}$ for all $1 \leq i, j \leq 3$

The rules are as follows:

- R_1: **If** A **then** P_1 is V_{11} and P_2 is V_{22}
- R_2: **If** B **then** P_1 is V_{12} and P_2 is V_{23}
- R_3: **If** C **then** P_1 is V_{13} and P_2 is V_{21}

Suppose, for a lack of time, that the controller is prevented from going over all the rules in this example, then only a subset of the rules would have been

computed. Would the partial result approximate the final result? Unfortunately, the answer, generally speaking, is no. In this example, in any order rules are fired the output values of both variables will not be monotonous (increasing or decreasing) with respect to the final result.

Suppose, for example, rules R_1, R_2 and then R_3 are fired. This will incrementally assign the values V_{11}, V_{12} and V_{13} to P_1 affording P_1 an increasing order of values. As for P_2, the order of values is V_{22}, V_{23} and then V_{21}. Here the monotony is broken by V_{21} ($V_{22} < V_{23}$ but $V_{22} > V_{21}$). In this example there is no possible order of rules R_1, R_2 and R_3 such that values of both P_1 and P_2 would be fired monotonously.

Any-time algorithms address such difficulties because the quality of their output improves gradually as computation time increases. Term any-time algorithm was coined by T. L. Dean in the late 1980's. Anytime algorithms offer a tradeoff between solution quality and computation time. An algorithm is called *anytime* if (1) it can be stopped at any time to provide a valid result of some quality, and (2) it is guaranteed that this quality monotonically increases if more time is given.

Chapter 3

Related Work

Considerable work has been done to optimize and speed-up controllers in general, and fuzzy controllers in particular. Overall, several approaches have been taken:

- Specialized hardware [7, 11, 16, 12, 14] was developed to speed up the processing of fuzzy control rules, for common fuzzy sets and aggregation/defuzzification methods. However, specialized hardware minimizes the time it takes to compute controller output—it does not address the inherent non-anytime performance of such a controller.

- There have been several explorations examining ways to reorder or decompose rules of controllers (and expert systems), in order to improve their performance [10, 28, 9, 32]. None of these have examined anytime performance in fuzzy control systems.

- Finally, there exist algorithms (such as the RETE matching algorithm) which are specifically geared towards efficient selection of the rules that should be recomputed when new inputs come in. Again, there is no guarantee of anytime performance.

These are described below in detail, in separate sections. In addition, other methods exist that are described in Sections 3.4, 3.5.

3.1 Specialized hardware

[12] describes a hardware design that performs fast inference of trapezoid-shaped membership functions, using the fact that membership functions have a known shape, and the shape is a set of linear functions. Analysis was performed of all possible intersections between two fuzzy sets. Based on cited points, an inference processor was designed, which, combined with pipelining the processor, achieves high computation speed. Two major disadvantages of this processor are the limited granularity of the degrees of membership — there are only 16 possible membership degrees — and the shape of membership functions — a trapezoid. Smooth functions are not available.

[14] describes a parallel fuzzy controller. It handles up to four input variables and one output variable with precision of eight bits. The number of fuzzy sets is limited to seven for each variable. The number of overlaps between fuzzy sets is up to two. All the limitations mentioned are related to certain applications described in the paper. It is possible to implement a controller with expanded capabilities, but limitations will remain. Implementing a chip with (almost) unlimited capabilities e.g. hundreds or thousands of variables, tens of overlaps and high granularity, significantly increases its complexity and cost.

In contrast to complete hardware fuzzy controllers there are programmable embedded micro-controllers. Such controllers are characterized in [16]. Embedded controllers are programmed using standard languages, so they may be programmed to do just about anything regular software controller does. Since the memory of such a controller is limited, many vendors restrict both the number and the shape of the permissible fuzzy sets for both inputs and outputs. The common shape of a membership function is a triangle. As defuzzification is a relatively complex task, singletons are often used to simplify calculations.

Hardware controllers are usually very fast, since hardware designed for certain purposes is faster then generic CPU that is capable of handling any problem. In addition, vendors usually limit inputs for their hardware: The number of fuzzy sets, fuzzy rules, membership function shape etc. All these cause the hardware controller be fast, but possible environments for using such controllers are also limited. Thus, the hardware controller with a limited number of fuzzy rules or

fuzzy sets is usually not usable in environments where fuzzy rules are generated as a result of learning because such controllers have limited scalability, especially when the learning process updates the fuzzy rules database online. The size the fuzzy rules database will reach remains, in this case, unknown.

To satisfy a scalability constraint hardware might be used that is designed to perform a certain step of fuzzy control for a single rule. This solution is not limited to a number of rules on the one hand and shortens calculation time on the other hand. An example for such hardware is described in [21]. A solution portrayed there is an algorithm that approximates centroid (COG). The algorithm is characterized by short calculation time, low approximation error and same static, dynamic and statistical properties like COG. The algorithm is called DECADE — Decreased Effort Centroid Approximation DEfuzzification. It may be implemented both in hardware and software, significantly reducing calculation time. Such a speedup algorithm may successfully be combined with any-time behavior of the controller described in this thesis.

Assuming a certain hardware fuzzy controller can handle the necessary number of fuzzy rules and fuzzy sets, even a fast hardware fuzzy controller may take a long time, when the necessary number of calculations is big, till all outputs are calculated. In such a situation the any-time controller may be suitable to guarantee that no matter how long calculations take, the controller's client will have its output from the fuzzy controller when needed.

3.2 Rule reordering and restructuring

Hicks [10] examines rule-ordering in non-fuzzy rule-bases and discusses its impact on accuracy and efficiency. He shows that in some cases, efficiency comes at the cost of accuracy. He proposes prioritizing fuzzy rules according to fuzzy rule type and environment. Hicks also mentions the fact that an increasing number of input variables in fuzzy rules may cause slow-down because of waiting for input (input/output is usually much slower then the calculation process). Mostly, a reordering according to fuzzy rule type and environment will prevent the fuzzy controller from being an any-time controller. Nevertheless, combining the reordering proposed in [10] with methods described in this thesis may cause a controller to

be any-time with a certain degree of granularity. It is possible to combine rules to units such that the intermediate results of units cause any-time behavior, while the intermediate result of fuzzy rules in the same unit do not have a monotonic character.

Another approach to fuzzy rules ordering is described in [28], where decomposing a monolithic fuzzy rules database to behaviors is suggested. The final controller is a hierarchy of behaviors, as a result of which a behavior-based controller is obtained. Each behavior in the controller can be seen as a stand-alone fuzzy sub-controller. This approach can easily be combined with the any-time approach by converting each behavior to an any-time sub-controller. The way such behaviors may be coordinated is also described in [28] while additional approaches for monitoring and coordinating any-time algorithms (sub-controllers) are described in [9, 32].

3.3 RETE and large-scale rule bases

A key step in activating rules involves the process of going through all rules in the rule-base, and matching each one against the inputs to the system. For each rule, we check its activation (in the non-fuzzy case) or degree of activation (defined by the fuzzification value in the fuzzy case). For small rule bases (say, a few dozen rules), a naive linear matching process is sufficient (checks all rules in each step).

However, when the number of rules is scaled up, it becomes computationally burdensome to go through all rules in each step. The RETE algorithm [6] is used to solve this problem. It remembers previous activations for each rule. As a result, only rules whose activation is affected by new inputs is fired (recomputed). Thus, the RETE algorithm significantly reduces number of rules that should be handled each time. The algorithm uses a data structure called the RETE network, Where each path in the network represents a left-hand-side of rule.

RETE has been adapted for use in fuzzy controllers [19]. This work presents a RETE network that consists of a cascade of three networks: The pattern network, the join network, and the evidence aggregation network. The first two layers are modifications of the regular RETE network. The third layer is a new concept.

RETE network addresses an inference, and aggregation steps of fuzzy control

and defuzzification have to be done afterwards. The main aim of this thesis is to order fuzzy rules such that the intermediate results will be monotonic. In the case of a RETE network it is hard to influence an order of rules firing, so detailed research is necessary to find the best way to integrate any-time behavior in a fuzzy controller based on RETE network. Nevertheless, a simple solution is possible and is detailed in section 9.2.3.

3.4 Learning methods

Fuzzy systems are generated using either expert or machine learning. Various automatic learning techniques pay attention to different aspects of the fuzzy system. Thus [26] and [27] present a technique that uses a genetic algorithm with a minimal fuzzy rules database. By reducing the database size this technique reduces calculation time. An Adaboost algorithm may be used to learn a fuzzy rule base described in [4].

[26] and [27] use a genetic algorithm combined with 'entropy' optimization to generate a fuzzy system. All membership functions generated by the genetic algorithm have the same general linearly approximated (by six line segments) form and Gaussian probability distribution function. The line approximation is used to speed up evaluation. Membership functions may be symmetric (same standard deviation on left and right sides) or asymmetric (various σ_{right} and σ_{left}). Entropy is used to obtain a minimal fuzzy rule base. All the results in a fuzzy rule base are described above and maximally suit an original domain while using the minimal number of fuzzy rules. Such fuzzy systems are comparable to systems designed by an expert but with a better I/O behavior.

Another approach to genetic algorithms for generating a fuzzy controller is proposed in [5]. The general idea of building (optimal) fuzzy expert system as presented there is to use clustering algorithms to divide an entire pattern space to subspaces. The center of each cluster is then mapped for a fuzzy rule. A problem represented that way is actually a search problem — finding the best representation of space by fuzzy rules. The way proposed to solve the search problem is a genetic algorithm. After representation of the fuzzy system is defined, the next question to be handled is the probabilities of crossover and mutation opera-

tors. Although these probabilities are often held constant or changed linearly for the entire run of a genetic algorithm, such an approach will not produce optimal results in many cases. Based on the experience of researchers, fuzzy rules for controlling probabilities of crossover and mutation operators are used.

Evolutionary fuzzy expert systems discussed in that paper evolve using a GA, membership function shapes, number of rules and other parameters. Genetic parameters of the evolutionary algorithm are adapted via the fuzzy expert system. The entire approach is scalable and can handle large and complex environments.

An Adaboost algorithm, and connections between it and iterative learning algorithms are described and analyzed in [4]. In contrast to iterative learning that removes correctly classified examples from training sets, Adaboost reduces the weight of such examples. Therefore, Adaboost always take into account all examples, preventing it from generating conflicting rules that are removed during a post-processing stage in a regular iterative learning algorithm. Experiments performed to compare Adaboost to other learning techniques show that Adaboost demands significantly less time to learn an environment. On the other hand, weighted fuzzy rules generated by Adaboost are less comprehensive and interpretable. Certain weighted fuzzy rule bases generated by Adaboost may be unsuitable for an any-time fuzzy controller because low-weighted rules may be handled before high-weighted ones. This may cause undesired behavior, where the first intermediate results of any-time algorithms are significantly far from the full calculation result (result that takes all fuzzy rules into account).

Modern learning methods return an optimal fuzzy system in terms of fuzzy rule base size, and complex environments still demand large fuzzy rule bases. Nevertheless, increasing the number of rules learned for the controller combined with any-time property may significantly improve the quality and usability of a fuzzy controller and its output, increasing the range of applications where such a controller may be integrated.

3.5 Non-fuzzy systems

Similar to the trade-off between accuracy and efficiency described in [10], another kind of trade-off is described in [3] between efficiency and generalization as a way

to combine two problem solvers. The first one is interpreted and uses a general knowledge representation, enabling the system to learn general rules. The second is compiled and uses specialized knowledge representation enabling the system to solve problems rapidly. A second solver decides when learning should occur.

Several mechanisms are used to speed up rules such as reordering rule conditions and deleting some useless conditions of the learned rules. These mechanisms do not modify learning. On the contrary, mechanisms that modify learning use cuts in the unification graph, specialization of multi-attributes predicates and compilation of the learned rules into C++ programs. The combination of two problem solvers, where each has its ways for speed-up, and the results of its application to Go game are shown in the paper.

In [13] there is a description of an algorithm that optimizes rules in rule-based systems that are utilize the RETE matching algorithm. There exist few major heuristics for optimizing rules but unfortunately they are conflicting, thus there is no guarantee that a particular heuristic always leads to optimization. To find the best optimization, all optimizations are applied and the best is chosen. Certain constraints are applied in order to reduce the number of checks performed by an algorithm. Optimization also takes into account a set of previous runs and the relevant statistics because the same set of rules in different domains may have different efficiency. This happens because heuristics conflict with each other. Thus rules used more often should be better optimized. Certain evaluation shows a reduction to a third in inter-condition tests and to half in CPU time.

Chapter 4

Monotony Background

As mentioned above, an any-time algorithm is an algorithm whose intermediate result improves with time, tending to a real result that demands full calculation time. It will be shown here that all intermediate results, including a final result, are monotonic. Such a monotony gives an any-time behavior. This thesis deals with two phases of fuzzy control: Aggregation and defuzzification. The aggregation methods addressed are max-min and sum-product, and the defuzzification methods are: MOM(mean-of-maxima) and COG (center-of-gravity).

This thesis will deal with all combinations of mentioned aggregation and defuzzification methods, i.e., any-time behavior will be proven for each combination:

- Max-min with MOM.
- Sum-product with MOM.
- Max-min with COG.
- Sum-product with COG.

Although this study describes only four pairs of aggregation and defuzzification, the main claim about monotony and any-time behavior is wider. In fact, the defuzzification method is the one that determines whether a certain fuzzy controller has any-time property or not. For this reason, fuzzy controllers that use other defuzzification methods, that can guarantee a monotony, may easily be converted to any-time controllers.

This thesis will prove that the defuzzifying output sets of a certain variable in a certain order (the increasing order of a set's own COG), will change a total COG of the output variable monotonically. This thesis further proves that the claim is also valid for MOM, COG (as defuzzification) and max-min, sum-product (as aggregation).

Chapter 5

Any-time in MOM and COG Defuzzification

An any-time algorithm is an algorithm whose intermediate results are monotonous. Therefore, in order to prove that a fuzzy controller based on MOM or COG can be converted to an any-time fuzzy controller, we are going to prove that both defuzzification methods combined with either max-min of sum-product aggregation methods can return monotonous intermediate results.

5.1 MOM

$$MOM(A) = \frac{\sum_1^N y_i}{N}$$

where membership function $\mu_{A'}(y_i)$ $(i = 1, 2, ..., N)$ reaches its global maximum.

5.1.1 Max-min

Monotony for MOM in case of max-min is based on the following claim:

Claim 1: For any two fuzzy sets S_1 and S_2, such that $height(S_2) \geq height(S_1)$:

$$MOM(S_2) \geq MOM(S_1) \Leftrightarrow MOM(S_1 \cup S_2) \geq MOM(S_1)$$

Note: $height(S_2) \geq height(S_1)$ is a reasonable assumption, because otherwise $height(S_2) < height(S_1)$ does not influence on $MOM(S_1 \cup S_2)$ and $MOM(S_1 \cup S_2) = MOM(S_1)$, which is a particular case of the claim.

General formulation of Claim 1 Let MOM_A be mean-of-maxima of A, MOM_B be mean-of-maxima of B and MOM_{AB} be mean-of-maxima of $A \cup B$, where $height(B) \geq height(A)$, then

$$MOM(B) \geq MOM(A) \Leftrightarrow MOM(A \cup B) \geq MOM(A)$$

Proof

Definition:

The next two cases can be discussed for any two fuzzy sets (A) and (B) where $height(B) \geq height(A)$:

1. $height(B) > height(A)$
2. $height(B) = height(A)$

In the first case $MOM(A \cup B)$ is $MOM(B)$ and $MOM(B) \geq MOM(A) \Leftrightarrow$

$$MOM(A \cup B) \geq MOM(A)$$

In the second case $MOM(A \cup B)$ is a value between $MOM(A)$ and $MOM(B)$ as it is known that $MOM(B) \geq MOM(A) \Leftrightarrow$

$$MOM(A \cup B) \geq MOM(A)$$

5.1.2 Sum-product

When aggregating fuzzy rules by sum-product, output sets are aggregated by summation (and not by taking max like in max-min). Summation of two sets may realize result values greater than 1, and does not conform to fuzzy set theory. Because of defuzzification, this is, however, a minor problem from the practical point of view. Monotony for MOM in case of sum-product is based on the next claim (that is similar to the previous one):

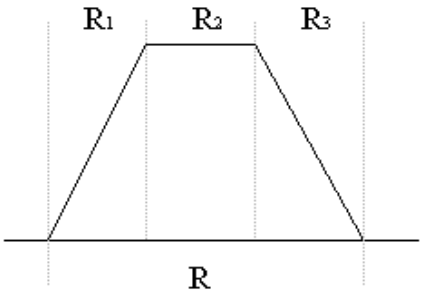


Figure 5.1: Fuzzy set ranges

Claim 2: For any two fuzzy sets A and B:

$$MOM(B) \geq MOM(A) \Leftrightarrow MOM(A \uplus B) \geq MOM(A)$$

where $\uplus$ is a sum-product aggregation of A and B.

Proof

Definitions:

Let A and B be two fuzzy sets such that $MOM(B) \geq MOM(A)$. Let f and g be functions that define sets A and B respectively, i.e. sets A and B are areas between functions f and g, respectively, and axis OX. Let R_A and R_B be ranges on which sets A and B are defined. Three ranges R_{A_1}, R_{A_2} and R_{A_3} (see figure 5.1) are assumed such that:

- $R_{A_1}, R_{A_2}, R_{A_3} \subseteq R_A$
- $R_{A_1} < R_{A_2} < R_{A_3}$
- $f'_{R_{A_1}} > 0$
- $f'_{R_{A_2}} = 0$
- $f'_{R_{A_3}} < 0$

Note: Each one among R_{A_1}, R_{A_2} and R_{A_3} may be a single point.

The same is assumed for R_{B_1}, R_{B_2}, R_{B_3} and R_B.

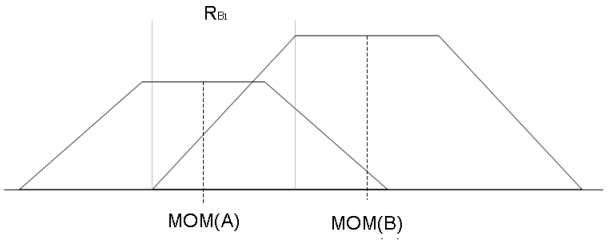


Figure 5.2: Claim 2, $MOM(A) \notin R_{B_2}, MOM(A) \in R_{B_1}$

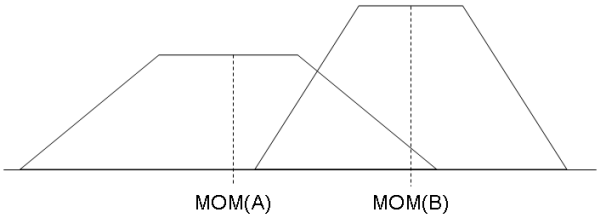


Figure 5.3: Claim 2, $MOM(A) \notin R_{B_2}, MOM(A) \notin R_{B_1}$

Let $h = f + g$. Let h' be the derivative of h. MOM of h should reach its maximum when h' is 0 (or h' changes its sign from positive to negative), i.e. f' + g' = 0. It is assumed that f and g have only one maximum which is a global maximum. The maximum may be either a single point or a range.

There are two major cases: $MOM(A) \in R_{B_2}$ and $MOM(A) \notin R_{B_2}$.

1. $MOM(A) \notin R_{B_2}$**:** $MOM(A) \notin R_{B_2} \Leftrightarrow MOM(A) < R_{B_2}$. See figure 5.2

If $MOM(A) \in R_{B_1}$ then $f'(MOM(A)) = 0$ because ($MOM(A) \in R_{A_2}$) and $g'(MOM(A)) > 0$ because ($MOM(A) \in R_{B_1}$). That is why $h'(MOM(A)) > 0 \Leftrightarrow h(R_{B_2}) > h(MOM(A)) \Leftrightarrow MOM(A \uplus B) \geq MOM(A)$.

If $MOM(A) \notin R_{B_1}$ then $\forall x < MOM(A)$ it is true that $h(x) \leq h(MOM(A))$ and $\forall x > MOM(A)$ it is true that $h(x) \geq f(MOM(A)) \Leftrightarrow MOM(A \uplus B) \geq MOM(A)$.

2. $MOM(A) \in R_{B_2}$**:** Here we have two possibilities: $MOM(B) \in R_{A_2}$ and $MOM(B) \notin R_{A_2}$.

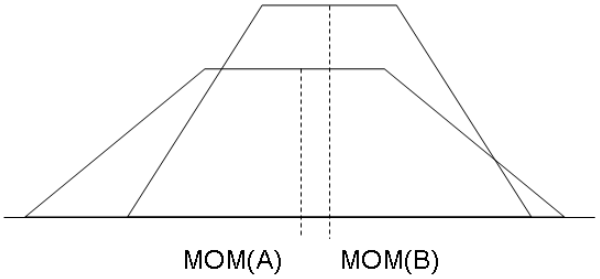


Figure 5.4: Claim 2, $MOM(A) \in R_{B_2}, MOM(B) \in R_{A_2}$

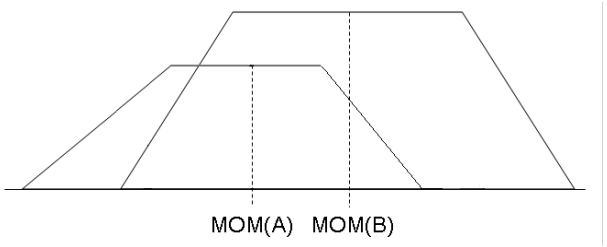


Figure 5.5: Claim 2, $MOM(A) \in R_{B_2}, MOM(B) \in R_{A_3}$

If $MOM(B) \in R_{A_2}$ then $h'(MOM(B)) = 0$ because $f'(MOM(B)) = 0$ and $g'(MOM(B)) = 0 \Rightarrow$

$$MOM(A \uplus B) = MOM(B)$$

When $MOM(B) \notin R_{A_2}$, $MOM(B) \in R_{A_3}$. In that case $h'(MOM(A)) = 0$ and $h'(MOM(B)) < 0 \Rightarrow$

$$MOM(A \uplus B) = MOM(A)$$

This proves that

$$MOM(B) \geq MOM(A) \Leftrightarrow MOM(A \uplus B) \geq MOM(A)$$

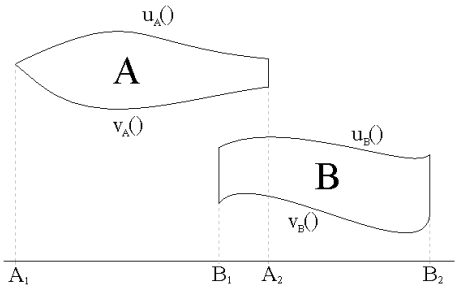


Figure 5.6: Pair of two-dimensional areas for COG monotony claim

5.2 COG

The monotony of COG is based on the claim described below. The claim deals with pair of two-dimensional areas A and B. Each of the areas is limited by functions $u_A()$ and $u_B()$ on the top and $v_A()$ and $v_B()$ on the bottom. In the case of fuzzy sets $\forall x : v_A(x) = v_B(x) = 0$. Functions $u_A()$ and $v_A()$ are defined on the range $[A_1, A_2]$ and $u_B()$ and $v_B()$ are defined on the range $[B_1, B_2]$ (see figure 5.6).

Claim 3 Let A and B be two areas in a two-dimensional space. Each of the areas is limited by functions $u_A()$ and $u_B()$ on the top and $v_A()$ and $v_B()$ on the bottom. Functions $u_A()$ and $v_A()$ are defined on the range $[A_1, A_2]$ and $u_B()$ and $v_B()$ are defined on the range $[B_1, B_2]$. We assume that pairs of functions u_A, v_A and u_B, v_B have no intersection points, but there may be intersection points at $(A_1, y_0), (A_2, y_1), (B_1, y_2)$ and (B_2, y_3) respectively.

Let COG_A be the center-of-gravity of A, COG_B be the center-of-gravity of B and COG_{AB} be the center-of-gravity of $A \cup B$, then

$$COG_B \geq COG_A \Leftrightarrow COG_{AB} \geq COG_A$$

Proof

Definition:

$$COGf(x) = \frac{\int_x xf(x)dx}{\int_x f(x)dx}$$

In this case

$$COG_A = \frac{\int x(u_A(x) - v_A(x))dx}{\int (u_A(x) - v_A(x))dx},$$

$$COG_B = \frac{\int x(u_B(x) - v_B(x))dx}{\int (u_B(x) - v_B(x))dx}$$

and

$$COG_{AB} = \frac{\int x(u_A(x) - v_A(x))dx + \int x(u_B(x) - v_B(x))dx}{\int (u_A(x) - v_A(x))dx + \int (u_B(x) - v_B(x))dx} \Leftrightarrow$$

$$\frac{\int xu_A(x)dx - \int xv_A(x)dx + \int xu_B(x)dx - \int xv_B(x)dx}{\int u_A(x)dx - \int v_A(x)dx + \int u_B(x)dx - \int v_B(x)dx} \Leftrightarrow$$

We wish to prove that

$$COG_{AB} \geq COG_A$$

then

$$\frac{\int xu_A(x)dx - \int xv_A(x)dx + \int xu_B(x)dx - \int xv_B(x)dx}{\int u_A(x)dx - \int v_A(x)dx + \int u_B(x)dx - \int v_B(x)dx} \geq \frac{\int xu_A(x)dx - \int xv_A(x)dx}{\int u_A(x)dx - \int v_A(x)dx} \Leftrightarrow$$

$$(\int xu_A(x)dx - \int xv_A(x)dx + \int xu_B(x)dx - \int xv_B(x)dx) * (\int u_A(x)dx - \int v_A(x)dx) \geq$$

$$(\int u_A(x)dx - \int v_A(x)dx + \int u_B(x)dx - \int v_B(x)dx) * (\int xu_A(x)dx - \int xv_A(x)dx) \Leftrightarrow$$

$$\int xu_A(x)dx \int u_A(x)dx - \int xv_A(x)dx \int u_A(x)dx + \int xu_B(x)dx \int u_A(x)dx$$
$$- \int xv_B(x)dx \int u_A(x)dx - \int xu_A(x)dx \int v_A(x)dx + \int xv_A(x)dx \int v_A(x)dx$$
$$- \int xu_B(x)dx \int v_A(x)dx + \int xv_B(x)dx \int v_A(x)dx \geq$$
$$\int u_A(x)dx \int xu_A(x)dx - \int v_A(x)dx \int xu_A(x)dx + \int u_B(x)dx \int xu_A(x)dx$$
$$- \int v_B(x)dx \int xu_A(x)dx - \int u_A(x)dx \int xv_A(x)dx + \int v_A(x)dx \int xv_A(x)dx$$

$$-\int u_B(x)dx \int xv_A(x)dx + \int v_B(x)dx \int xv_A(x)dx \Leftrightarrow$$

$$\int xu_B(x)dx \int u_A(x)dx - \int xv_B(x)dx \int u_A(x)dx$$
$$-\int xu_B(x)dx \int v_A(x)dx + \int xv_B(x)dx \int v_A(x)dx \geq$$
$$\int u_B(x)dx \int xu_A(x)dx - \int v_B(x)dx \int xu_A(x)dx$$
$$-\int u_B(x)dx \int xv_A(x)dx + \int v_B(x)dx \int xv_A(x)dx \Leftrightarrow$$

$$(\int xu_B(x)dx - \int xv_B(x)dx) \int u_A(x)dx - (\int xu_B(x)dx - \int xv_B(x)dx) \int v_A(x)dx \geq$$
$$(\int xu_A(x)dx - \int xv_A(x)dx) \int u_B(x)dx - (\int xu_A(x)dx - \int xv_A(x)dx) \int v_B(x)dx \Leftrightarrow$$

$$(\int xu_B(x)dx - \int xv_B(x)dx)(\int u_A(x)dx - \int v_A(x)dx) \geq$$
$$(\int xu_A(x)dx - \int xv_A(x)dx)(\int u_B(x)dx - \int v_B(x)dx) \Leftrightarrow$$

$$\frac{\int xu_B(x)dx - \int xv_B(x)dx}{\int u_B(x)dx - \int v_B(x)dx} \geq \frac{\int xu_A(x)dx - \int xv_A(x)dx}{\int u_A(x)dx - \int v_A(x)dx} \Leftrightarrow$$

$$COG_B \geq COG_A$$

Q.E.D.

When COG is used as a defuzzificator the same claim and the same proof can be used for max-min and for sum-product, because COG of the set is not affected by functions added to $u()$ and $v()$ — top and bottom borders of the set.

5.3 Monotony Summary

Three claims have been shown above that prove monotony of defuzzification for two fuzzy sets. These claims will now be used to prove monotony for an arbitrary number of fuzzy sets.

5.3.1 General formulation

Each of claims described above deals with a different aggregation method combined with a different defuzzification method. Generally, these claims may be combined into next lemma:

General pair-sets lemma. Let aggregation methods be max-min or sum-product. Let DEF be one of defuzzification methods COG or MOM. For S set of fuzzy sets, let $DEF(S)$ be the result of the calculation of certain defuzzification methods on S. For any two fuzzy sets A and B

$$DEF(B) \preceq DEF(A) \Leftrightarrow DEF(AB) \preceq DEF(A)$$

5.3.2 Main lemma

After a lemma for monotony of defuzzification for two fuzzy sets has been formulated, it may be extended to a general case with an arbitrary number of fuzzy sets.

Let $S = \{S_1, S_2, ..., S_n\}$ be fuzzy sets in certain order (increasing or decreasing) of self defuzzification values, i.e. $DEF(S_1) \preceq DEF(S_2) \preceq ... \preceq DEF(S_n)$. Let $S^k = \{S_1, S_2, ..., S_k\}$ for any $1 \leq k \leq n$.

Calculating a defuzzification value of S iteratively: An intermediate result $DEF(S^{i-1})$ at iteration i is obtained of previous iteration and is extended to take S_i into account.

Lemma.

Intermediate results of iterations described above

$$DEF(S^1), DEF(S^2), ..., DEF(S^n)$$

are monotonic.

Proof

In order to prove this lemma the following claim will be used:

Claim 4:

Let $S = \{S_1, S_2, ..., S_n\}$ be fuzzy sets in certain order (increasing or decreasing) of self defuzzification values, i.e. $DEF(S_1) \preceq DEF(S_2) \preceq ... \preceq DEF(S_n)$. Let $S^k = \{S_1, S_2, ..., S_k\}$ for any $1 \leq k \leq n$, then

$$\forall 0 \leq k \leq n, DEF(S_1) \preceq DEF(S^k) \preceq DEF(S_k)$$

Proof

Proof by induction.

The lemma is correct for one set, i.e. $DEF(S_1) \preceq DEF(S^1) \preceq DEF(S_1)$.

The lemma is correct for two sets according to a general lemma for pair of fuzzy sets: $DEF(S_1) \preceq DEF(S^2) \preceq DEF(S_2)$

Assumption: It is assumed that the lemma is correct for $k = i$, i.e.

$$DEF(S_1) \preceq DEF(S^i) \preceq DEF(S_i)$$

We would like to prove it is correct for $k = i + 1$, i.e.

$$DEF(S_1) \preceq DEF(S^{i+1}) \preceq DEF(S_{i+1})$$

Let us focus on S^i and S_{i+1}. According to assumption above, $DEF(S^i) \preceq DEF(S_i) \Rightarrow$

$$DEF(S^i) \preceq DEF(S_i) \preceq DEF(S_{i+1}) \Rightarrow$$

$$DEF(S^i) \preceq DEF(S_{i+1})$$

By definition $DEF(S^i, S_{i+1}) = DEF(S^{i+1})$. According to a general lemma for pair of fuzzy sets (S^i, S_{i+1})

$$DEF(S^i) \preceq DEF(S^{i+1}) \preceq DEF(S_{i+1}) \Rightarrow$$

$$DEF(S_1) \preceq DEF(S^i) \preceq DEF(S^{i+1}) \preceq DEF(S_{i+1}) \Rightarrow$$
$$DEF(S_1) \preceq DEF(S^{i+1}) \preceq DEF(S_{i+1})$$

Q.E.D.

Lemma proof. Continuation: The proof will be by induction.

The lemma is correct for one set, i.e. $DEF(S^1)$ is always monotonic.

The lemma is correct for two sets according to a general lemma for pair of fuzzy sets described above: $DEF(S^1) \preceq DEF(S^2)$

Assumption: The lemma is assumed to be correct for i-th step, i.e.

$$DEF(S^1) \preceq DEF(S^2) \preceq ... \preceq DEF(S^i)$$

It is necessary to prove the lemma is correct for $(i+1)$-th step, i.e.

$$DEF(S^1) \preceq DEF(S^2) \preceq ... \preceq DEF(S^i) \preceq DEF(S^{i+1})$$

It is known that $DEF(S_i) \preceq DEF(S_{i+1})$. On the other hand, according to Claim 4, $DEF(S^i) \preceq DEF(S_i) \Rightarrow$

$$DEF(S^i) \preceq DEF(S_{i+1})$$

By general lemma for two fuzzy sets (S^i, S_{i+1})

$$DEF(S^i) \preceq DEF(S^{i+1}) \preceq DEF(S_{i+1}) \Rightarrow$$

$$DEF(S^i) \preceq DEF(S^{i+1})$$

Combining this with the assumption of induction:

$$DEF(S^1) \preceq DEF(S^2) \preceq ... \preceq DEF(S^i) \preceq DEF(S^{i+1})$$

Q.E.D.

5.3.3 Summary

It has been proven in this section that the intermediate results of the defuzzification of any set of fuzzy sets are monotonic constraints:

- Using any of max-min and sum-product as aggregation
- Using any of MOM and COG as defuzzification

This proof may be extended to other defuzzification methods that meet conditions of the general lemma for pair of fuzzy sets, i.e., for the S set of fuzzy sets, let $DEF(S)$ be result of calculation of a certain defuzzification method on S, then for any two fuzzy sets A and B

$$DEF_B \preceq DEF_A \Leftrightarrow DEF_{AB} \preceq DEF_A$$

There are defuzzification methods that do not fulfill a general lemma for a pair of fuzzy sets, and so monotony may not be proven for them. An example for such a defuzzification method is $ArbitraryMax$. Unlike MOM, in this defuzzification method we return an arbitrary X-axis value among all values for which height is maximal. (see figure 5.7).

For a fuzzy set in figure 5.7, $ArbitraryMax$ will return any point in range $[A, B]$. Hence, in the example below (see figure 5.8) monotony may not be guaranteed. Three fuzzy sets — A, B and C — exist. Each set is defined on a different range, but range $[x_1, x_2]$, is common for all three. In this example, a triplet of points a, b, c (a possible result of $ArbitraryMax$) for any order of set A, B, C exists that breaks a monotony. Nevertheless, monotonic order may be found and assured when ArbitraryMax is used, if fuzzy sets do not intersect (trivial case). In that case each set may be treated as if MOM was used as the defuzzification method. The order of the set will be found accordingly.

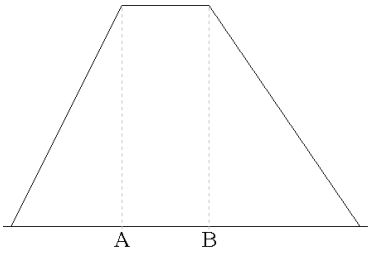


Figure 5.7: Arbitrary max fuzzy set

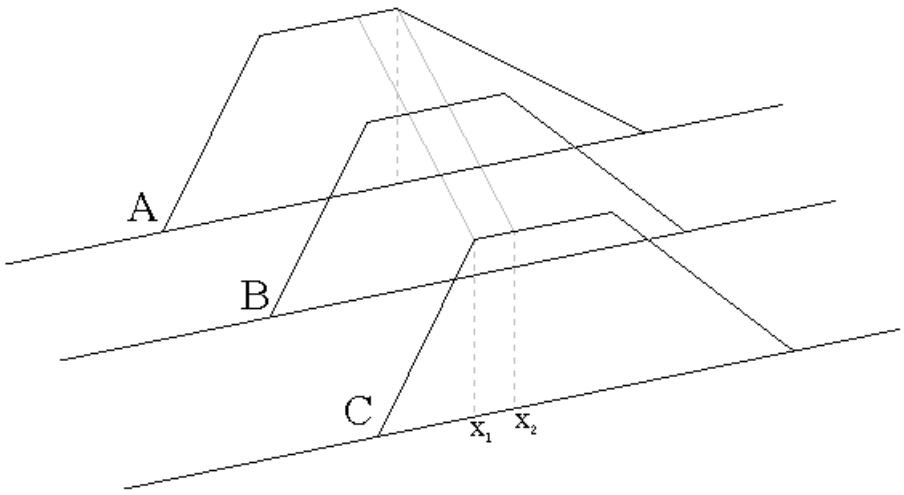


Figure 5.8: Arbitrary max example

Chapter 6

Any-Time Defuzzification in Practice

6.1 Overview

Till now the discussion has been of one output variable and a few fuzzy rules that modify it. This thesis addresses fuzzy controllers that return more than one variable hence a general controller with several output variables will now be discussed.

The thesis proposal pertains to a controller that uses max-min as the aggregation method and COG as defuzzification. That controller kept increasing the order of self COG for fuzzy sets of each output variable. Particularly, the controller used topological sorting for all fuzzy sets (each fuzzy set associated with fuzzy rule) in order to solve conflicts. For which reason that controller was actually an any-time controller.

It has been shown previously that keeping a certain (increasing/decreasing) order of self COG/MOM of fuzzy sets guaranties a monotonic intermediate result for each output variable. Thus the entire controller becomes an any-time controller for any aggregation-defuzzification pair among max-min, sum-product and MOM, COG.

Each fuzzy set is modified by a certain fuzzy rule. Interest is actually directed to the order in which fuzzy rules should be fired in order to obtain the desired

order of output of fuzzy sets. Some fuzzy rules modify fuzzy sets for more then one output variable. This may cause a conflict with monotony as in the example described above that entailed:

Example (reminder):

- Two output variables P_1, P_2
- Three rules R_1, R_2 and R_3 that modify these variables
- Three fuzzy values of each variable: for P_1 - V_{11}, V_{12} and V_{13}, for P_2 - V_{21}, V_{22} and V_{23}, such that $V_{ij} < V_{i,j+1}$ for all $1 \leq i, j \leq 3$

The rules are as follows:

- R_1: **If** A **then** P_1 is V_{11} and P_2 is V_{22}
- R_2: **If** B **then** P_1 is V_{12} and P_2 is V_{23}
- R_3: **If** C **then** P_1 is V_{13} and P_2 is V_{21}

If the controller is stopped, for lack of time, from going over all the rules in this example, the partial result will not approximate the final result. To solve this problem a fuzzy rules database should be altered so that an order of the resultant fuzzy rules would exist in any-time fuzzy controller.

In the example above rule R_3 should be split:

If C **then** P_1 is V_{13} and P_2 is V_{21}

into two sub-rules R_{3_1}, R_{3_2}

- R_{3_1}: **If** C **then** P_1 is V_{13}
- R_{3_2}: **If** C **then** P_2 is V_{21}

As a result rule R_{3_2} will be fired first, followed by R_1, R_2 and R_{3_1}. This offers a monotonic increasing order of intermediate results, and a fuzzy controller that handles rules in that order is an any-time fuzzy controller.

To determine a general order in which fuzzy sets will be handled, one of linearization methods described below should be used. If the rules are seen to be returned in the order that the linearization method returned them, an any-time behavior controller will be guaranteed.

6.2 Improvements to general defuzzification

Improvements described below may optionally be implemented in a fuzzy controller. They do not change a main property of a fuzzy controller described in this thesis — any-time. Further research is necessary to explore the exact influence of these improvements on the overall calculation time.

6.2.1 Reverse order

The following example demonstrates improvement:

Example 2

Fuzzy sets: S_1, S_2, S_3 and S_4.

Two fuzzy parameters P_1 and P_2. $\{P_1 : S_1, S_2\}$ such that $S_1 < S_2$. $\{P_2 : S_3, S_4\}$ such that $S_3 < S_4$.

Fuzzy rules R_1 and R_2 that modify fuzzy sets: $\{R_1 : S_1, S_4\}$. $\{R_3 : S_2, S_3\}$.

Here is a conflict situation of two parameters and two rules if an attempt is made to keep increasing the order of fuzzy sets. To solve the situation set S_4 can be delayed by splitting R_1 into two rules, and the final result then is handling fuzzy sets in the next order:
$(S_1), (S_2, S_3), (S_4)$.
Sometimes such a situation may be resolved without delaying sets. Sequence
$(S_1, S_3), (S_2, S_4)$
is a legal, non-conflictual sequence in which sets of parameter P_2 are performed in reverse order. It is still any-time because COG of P_2 is changed monotonically although it is in reverse order.

6.2.2 Unclear Order

In certain cases an order of self values of the defuzzification function of two sets depends on the height of these sets, for which reason the order needs to be determined at the runtime. This happens when fuzzy sets overlap in a certain way. Doing this may significantly slow down the controlling process. It also complicates planning (creating a sequence of fuzzy rules in the order in which they are supposed to be handled). Therefore, these sets should be handled as one unit in

order to promise monotony. The internal order of these sets is not important in most cases, particularly when COG is the issue.

6.2.3 Approximations

In basic any-time algorithms all fuzzy sets of each variable are supposed to be defuzzified in a certain order. As a result, it may take a (relatively) long time till the temporary result approximates the actual result. Several ways to reduce these times are introduced in section 7 below.

6.3 Linearization methods

This section will discuss linearization methods. Linearization is necessary to generate a sequence of fuzzy rules in order for these rules to be handled. For each output variable, a vector of fuzzy rules is created that modifies fuzzy sets of a certain output variable. First the vector in a certain (increasing/decreasing) order of self values of fuzzy sets' defuzzification function (MOM/COG) is ordered. These vectors are passed as input to linearization algorithm. In the previous section several items were introduced, that come up when there is more then one output variable, i.e., there are a few sequences of fuzzy rules (one for each output variable), each with its own result.

Often, some rules present in more then one sequence, i.e., the same rule modifies fuzzy sets of more then one output variable. This can cause conflict because of inconsistency in the order of certain rules (fuzzy sets) in various output variables. Two main approaches that can solve these conflicts are:

1. Using topological sorting and split fuzzy rules causing conflicts.

2. Split all rules to simple rules with one operand at the right hand side of the rule.

Each of the suggested approaches has its advantages and disadvantages, the advantages of one approach being the disadvantages of another. Topological sorting creates a database with relatively few rules, but the equal advance of intermediate

results among various variables cannot be guaranteed. On the other hand, splitting rules creates a big database, but the advance of the intermediate result may easily be controlled, e.g., priorities among variables.

Linearization methods are used to generate a sequence of rules, in order, that should be handled by a fuzzy controller.

6.3.1 Using topological sort

For a given database of fuzzy rules and output fuzzy sets, topological sort may be used to sort output fuzzy sets. Topological sort returns a vector of rules. In the case of this study, slight modification of topological sort is necessary. Modified by the same fuzzy rule, output fuzzy sets should then be handled sequentially (if possible). Thus there is no need to split a fuzzy rule. In contrast, output fuzzy sets that are not handled sequentially cause splitting of a fuzzy rule to few fuzzy rules. Each such split fuzzy rule modifies a subset (of output fuzzy sets) of the original fuzzy rule.

The result of such sort is a sequence having two main properties:

1. Fuzzy rules that modify fuzzy sets of certain parameters keep the internal order they had before topological sort.

2. Conflicts as in the example above are solved by splitting fuzzy rules to sub-rules.

This gives us an order of fuzzy rules processing.

Modification of topological sort mentioned above does two things: It causes the output of fuzzy sets that are modified by the same fuzzy rule to be sequential (when possible) and unites such sequences of fuzzy sets to one unit. Such output should be treated as a list of fuzzy rules in the order they should be handled by the controller.

The number of items in the list is greater (or equal) than the number of original rules and less (or equal) to the total number of output fuzzy sets at all output variables. The size of the list depends on the amount of split rules and granularity of the splits.

The main advantage of using such sort is the relatively small number of fuzzy rules that should be handled. There are environments for which more then one valid output of the sort exists. Further work is necessary to find a best output of the sort; for now, any valid output of the sort is used.

On the other hand, the disadvantage of this method is that rules that modify fuzzy sets of several variables simultaneously define dependencies between the variables. Such dependencies make it very hard, and sometimes impossible, to control the promotion of the intermediate result of one output variable relative to another.

6.3.2 Splitting to atomic rules

To avoid conflicts that arise when the same rule modifies fuzzy sets of more than one variable, such rules can be split into atomic rules - rules that modify only one output fuzzy set.

A database composed of atomic fuzzy rules gives maximum flexibility and control over the intermediate results of each output variable, because there are no dependencies between the output variables. Thus, it is easy to create a sequence of fuzzy rules that gives priority to some variables over others. In contrast, it is possible to generate a sequence that keeps an equal relative promotion of intermediate results for all variables.

The implementation of priorities is simple — rules that modify sets of variables with a higher priority must be handled before the others. Such rules will then precede other rules in the sequence of rules.

Implementation of equal relative promotion of intermediate results (fairness) is more complex hence a weighted round robin algorithm will be used. A weighted round robin handles in each turn a 'number' of rules for each variable, where

$$number = \frac{\textit{max number of output fuzzy sets}}{\textit{min number of output fuzzy sets}}$$

This number is actually a weight of the output variable. Weights may be modified in order to combine them with priorities.

The database of atomic fuzzy rules may have many more fuzzy rules than the

original database, depending on how complex were the fuzzy rules in the original DB. This is a value paid for flexibility in database of atomic fuzzy rules.

6.3.3 Hybrid algorithm and priority clusters

Topological sort and atomic rules have their advantages and disadvantages. Advantage can be taken in certain situations of both approaches, to which end they will be combined. Although further work on the hybrid of topological sort and atomic rules is necessary, some modifications are obvious.

Certain prioritization of output variables may be achieved by defining priority clusters of output variables. Clusters are subsets of output variables. All variables in the cluster have the same priority; any prioritization policy is possible between clusters.

According to defined clusters, rules that define dependencies between output variables in different clusters are split. New rules are not necessarily atomic rules. All output variables are modified by a certain rule contained in the same priority cluster. The fuzzy rules database obtained as a result of such modification consists of more rules than were in the original database. Despite this, only a limited number of necessary rules is split, thus the database size increases insignificantly.

Whenever priority clusters are defined, any prioritization policy is available. The simplest policy is a strict priority: Handle rules that modify output variables of certain priority clusters before others. It is possible to use a weighted round robin to guarantee equal relative promotion of intermediate results between priority clusters, but it should be noted that there is no guarantee of promotion of an intermediate result between output variables of the same priority cluster. A weighted round robin may also be used here to give relative priority between priority clusters.

The combination of topological sort and split rules as described above enables prioritizing algorithms without significantly increasing the fuzzy rules database. This method is useful for large databases with many dependencies and relatively few priority clusters. Otherwise, a database obtained as a result of clustering is similar to database of atomic rules.

6.3.4 Summary

Basic linearization methods described above are of the topological sort on the one hand with rule-splitting on the other. According to certain environmental constraints it is possible to choose a method that optimally appropriates. After using a chosen method on a certain fuzzy rules database, a sequence on fuzzy rules is obtained that should be used as input to the fuzzy controller. In order to attain an any-time effect a controller should have the following properties:

- Process fuzzy rules in a defined order;
- Return a defuzzification value of all output variables at any moment during processing (return an intermediate result);
- May be interrupted before all rules are processed.

The combination of linearization and fuzzy controller with these properties affords an any-time fuzzy controller. Such a controller can be interrupted at any moment and return a result of processing till interruption. It is guaranteed that increasing calculation time causes the returned result to be closer to the optimal (real) result.

6.3.5 Complexity

In case of atomic rules only sort of output sets of each rule is necessary. Its complexity is

$$O(|v||s| \log |s|)$$

Where:

- |v| – number of output variables
- |s| – max number of fuzzy sets in each output variable

In case of topological sort, it is necessary to sort output sets of each variable and then to perform a topological sort. The usual algorithms for topological sorting have running time linear in the number of nodes plus the number of edges $O(|V|+|E|)$. In our case number of nodes is a number of output sets i.e. $O(|v||s|)$. Number of edges is a number rules (edges between variables) $|r|$ and number of

sets on each variable (edges from set to set of the same variable) $|r||v||s|$. As a result complexity of linearization based on topological sort is

$$O(|v||s| \log |s| + |v||s| + |r| + |r||v||s|)$$

Where:

- |v| – number of output variables
- |s| – max number of fuzzy sets in each output variable
- |r| – max number of fuzzy sets in each rule

Chapter 7

Extensions

An any-time controller should be used to solve a problem of a long calculation time. An any-time controller facilitates obtaining a result that may not be perfect, but is attained much faster then the result of a full controlling sequence.

Claims mentioned above discuss an order between two fuzzy sets, in which these sets should be handled — increasing order of their own defuzzified value MOM/COG. However, this order will not always provide satisfying calculation times. Below, some methods will be presented that may significantly improve the quality of the intermediate result obtained after certain calculation time. Although improvements described below do ameliorate some aspects of an any-time controller, the exact influence should be further researched.

7.1 Optimal sequence

In this paper, the optimal sequence is a sequence of output sets that considers all output sets, but causes intermediate results to be as close as possible to the final result, while taking into account for defuzzification the least possible number of fuzzy sets.

The idea of an optimal sequence is next: Each step of defuzzification takes into account a set that has a maximal influence on the result, but still does not break a monotony.

MOM. Assume: Current intermediate result is MOM of set S whose height is $height(S)$ and MOM is $MOM(S)$. To bring the next intermediate result as close as possible to the final result, the next set to be taken into account should be a set S' that satisfies one of the following two constraints:

- If $height(S') = height(S)$ then for each set T, if $height(T) > height(S')$ then $MOM(S') \leq MOM(T)$
- $height(S') > height(S)$

When there are no sets that satisfy these constraints a final result is received. Sets previously skipped can now be handled, but it is not necessary since they do not influence the final result.

COG. Like in MOM, each iteration should handle a set that has a maximal influence on total COG, but still does not break a monotony.

Let $S = \{S_1, ..., S_n\}$ be sets for which COG is to be calculated. After i iterations the intermediate result is COG_i. $A \subseteq S$ — sets for which COG has been calculated already and B — sets in S that not in A. COG_i is COG of sets in A. W.l.o.g., assuming $A = \{S_1, ..., S_i\}$ and $COG(S_i) \leq COG(S_{i+1}) \leq ... \leq COG(S_n)$.

A set S_k is to be picked from B, moved to A and COG of $A \cup S_k$ should be calculated. Let COG_{AS_k} be $COG(A \cup S_k)$. We should now pick such S_k that suits next condition:

$$\forall COG_{AS_k}, COG_{AS_m} \leq COG(S_{i+1}) : COG_{AS_k} \geq COG_{AS_m}$$

7.2 Approximations

The optimal sequence presented above may indeed promise the fastest promotion of the intermediate result when all fuzzy sets are taken into account, but it still requires many calculations and possibly more calculations then the sub-optimal sequence. For this reason it should not be used, but other ways should rather be found to calculate good intermediate results. Possible ways to improve intermediate results are discussed below:

1. Using a previous result;
2. Using a subset of all fuzzy sets to calculate an intermediate result;
3. Use maximal possible change of a certain fuzzy set.

Note: COG is influenced by all fuzzy sets. MOM is determined by a few maybe even one fuzzy set. It can therefore be useful to find sets that do influence the final result.

7.3 COG

7.3.1 Use the previous result

When a COG of set of fuzzy sets is to be calculated, one may determine that use be made of a previous defuzzified value of these sets as an intermediate result as long as COG of fuzzy sets that are already taken into account is less then the previous defuzzified value (see algorithm below). `pop_first()` removes first element from a list and returns it:

```
1.  intr ← Previous defuzzified result
2.  tmp_cog ← −∞
3.  A ← all fuzzy sets
4.  Sort A in increasing order of COG
5.  B ← ∅
6.  Loop while A ≠ ∅
7.    B ← B ∪pop_first(A)
8.    tmp_cog ← COG(B)
9.    If tmp_cog > intr Then
10.       intr ← tmp_cog
11. End of loop
12. intr ← tmp_cog
```

In this algorithm, `intr` holds an any-time result in each stage. At the beginning it holds a previously defuzzified result. Later, when COG of new fuzzy sets becomes higher then `intr`, it holds new COG. Monotony of `intr` is not broken in any step, because it only grows up.

It is possible that $COG(A)$ is less then the previously defuzzified result. Line 12 of the algorithm covers this case: No matter what $COG(B)$ is at the end of the algorithm, it is used as an intermediate (final) result. Monotony is not broken here either: There is only one intermediate result and one final result.

The main advantage of using a previously defuzzified result is the fact that when changes in the fuzzy system between two following runs of the algorithm are not significant, there is sometimes not enough time to get close to real COG in certain iterations. An intermediate result does stays close to real COG.

On the other hand, the main disadvantage of using a previously defuzzified result is the fact that when real COG becomes less and less each time defuzzification is performed but there is not enough time to calculate it for the few times defuzzification is performed and the intermediate result returned by the algorithm becomes too unreal (retaining the same value, when real COG decreases each time).

This problem may be solved by not using the previously defuzzified result as is, but decreasing it each time the algorithm is started. In that case, after a few algorithm runs, the intermediate beginning result will be decreased to a value that can be reached while calculating COG of fuzzy sets with little self COG.

This solves a problem of a temporary result that totally differs from results calculated by the defuzzificator. On the other hand, the results returned here may be caused by fuzzy sets with little values of self COG, causing the defuzzification result to (a) have relatively little value; (b) be unaffected by fuzzy sets with high self COG.

7.3.2 Use subset of fuzzy sets

Another improvement suggested here for the first intermediate result is to calculate COG of two or more fuzzy sets with more or less equally dispersed values of self COG. Here fuzzy sets with both low and high self COG are used.

7.3.3 Use maximal possible change of a certain fuzzy set

Maximal possible change is not useful with COG because it only doubles or triples calculation time without any real benefit.

7.4 MOM

Using the previous result and a subset of fuzzy sets for MOM are similar to those of COG, but it should be noted that these methods are not useful with MOM. MOM, in contrast to COG, is a result of a small subset of sets, and usually only one set. For this reason the previous results should not be used as was the case with COG.

7.4.1 Use maximal possible change of a certain fuzzy set

Using the fact that only the small subset of all fuzzy sets participates in MOM, it does not have to be calculated for all sets, but only those that may influence MOM.

To explore which fuzzy sets influence MOM the maximal possible change of each fuzzy set height may be used (if possible), i.e., this information can be used if a maximal and minimal value for the height of the fuzzy set can be used at the next iteration. Notable in this regard is MOM of fuzzy sets whose maximal value is lower then the minimal value of fuzzy sets that determined previous MOM should not be calculated. Thus can the number of fuzzy sets for which MOM should be calculated be significantly reduced.

If sequential heights of a certain fuzzy set are close, one can assume there is a curve that determines these heights and try to predict next values in order to determine whether the fuzzy set should be handled.

Let $h_i, h_{i+1}, h_{i+2}, h_{i+3}$ be sequential heights of certain fuzzy set. Let h'_i be $h_{i+1} - h_i$, $h'_{i+1} = h_{i+2} - h_{i+1}$ and $h''_i = h'_{i+1} - h'_i$. If a constant second derivative is assumed then:

$$h'_{i+2} = h''_i + h'_{i+1}$$

$$h_{i+3} = h'_{i+2} + h_{i+2}$$

h_{i+3} (an approximation) and one may assume that a real value of h_{i+3} is in range $[h_{i+3} - \varepsilon, h_{i+3} + \varepsilon]$.

7.5 Summary

The approximation mentioned above may significantly reduce the time necessary for the intermediate result to approach the real result. The actual influence of approximations should be researched further. Additional approximations should be explored and examined. The necessary constraint for approximation is a monotonous intermediate result in any possible scenario.

Chapter 8

Experiments

Below results of experiments performed with any-time fuzzy controller are presented. Experiments were performed on simulation of robot moving in 2-dimensional environment with a single obstacle to avoid (see figure 8.1).

Environment details are as follows:

- Robot initial location is (0, 1)
- Destination is (10, 0)
- Obstacle — a line segment between (5, 5) and (5, -5)
- Gray line is an intended robot path.

Fuzzy rule base properties are as follows:

1. Two output variables: direction (-90, 90), acceleration (-0.5, 0.5)
2. Input variables:
    - Direction to destination
    - Distance to left edge of obstacle
    - Distance to right edge of obstacle
    - Straight distance to obstacle
    - Current speed

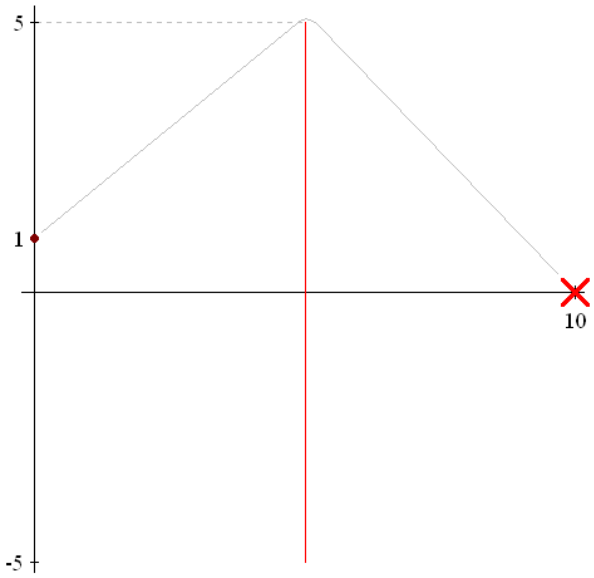


Figure 8.1: Experiments environment

3. Total number of 36 fuzzy rules. A limitation of fuzzy controller used for experiments in current thesis is the fact that all input variables have to be explicitly teated by each rule. There actually were used two pseudo controllers. Each pseudo controller for one output variable. Each input variable has 3 fuzzy sets. One output variable is calculated based on 3 input variables, and another one based on 2 input variables. As a result there are 27 rules for one output variable, and 9 for another one.

A path that a robot walked on using full calculation (no any-time property) is displayed in figure 8.2.

This fuzzy controller was used to perform two kinds of experiments:

- Proof of concept — show that sorting fuzzy rules in increasing (decreasing) order of self defuzzification values results in a monotonic intermediate results.
- Show that any-time fuzzy controller fulfill its purpose.

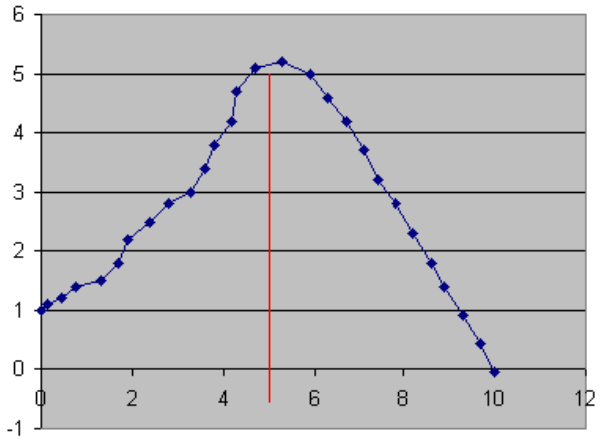


Figure 8.2: Full calculation path

8.1 Proof of concept

In order to prove that intermediate results of controller described in this thesis are monotonic, following measurements were performed at few arbitrary points on path mentioned above. At each one of chosen points of the path, $1, 2, ...$ rules were fired and intermediate results were recorded. Inputs and intermediate results are presented in tables in figure 8.3. Intermediate results were taken for all combinations of COG, MOM and incrementing, decrementing order.

There are two tables at figure 8.3. Top table describes acceleration variable and a bottom table describes a direction variable. Both tables have the same structure, so following description is valid for both tables. There are intermediate results for both defuzzification methods MOM and COG. Four rows for each. Each defuzzification method was checked in both increasing and decreasing orders of self defuzzification values: two rows for each. So, each combination of defuzzification method and firing order was recorded at two different points of the path. In case of direction variables, one of points is a starting point $(0, 1)$. This shows us that for given defuzzification method, full calculation of both increasing and decreasing orders result in the same value. Content of each row is as follows:

- First two columns are x and y coordinates of robot, where a measurement was performed.

		Location		Input		Acceleration Intermediate Results (# rules fired)				
		x	y	Straight distance	Current speed	1	2	3	4	5
COG	Incr	0.45	1.2	4.6	0.33	-0.15	-0.12	-0.04	0.071	0.071
		1.3	1.5	3.8	0.51	-0.16	-0.11	-0.076	-0.018	-0.018
	Decr	1.7	1.8	4.1	0.49	-∞	0.1	0.063	0.01	-0.0086
		2.4	2.5	2.9	0.55	∞	0.1	0.05	-0.024	-0.052
MOM	Incr	0.47	1.1	4.5	0.3	-∞	-∞	0	0.1	0.1
		3	3.5	∞	0.5	-∞	-∞	0	0	0.2
	Decr	0.17	1.1	5.6	0.2	0.2	0.1	0.1	0.1	0.1
		1.7	2.8	3.3	0.7	0	0	0	-0.1	-0.1

		Location		Input			Direction Intermediate Results (# rules fired)				
		x	y	Destination direction	Left edge distance	Right edge distance	1	2	3	4	5
COG	Incr	0	1	5.71349	6.40312	7.81025	-∞	-30.15	-30.15	22	40.87
		3.6	3.4	73	2.1	8.5	-∞	-30	-30	-5.2	27
	Decr	0	1	5.71349	6.40312	7.81025	64.9	46.65	46.65	40.87	40.87
		4.7	5.1	88	∞	∞	∞	∞	∞	-30	-30
MOM	Incr	0	1	5.71349	6.40312	7.81025	-∞	-29.7	-29.7	29.7	29.7
		6.5	4.2	23	∞	∞	-∞	-30	-30	-30	-30
	Decr	0	1	5.71349	6.40312	7.81025	90	29.7	29.7	29.7	29.7
		7	3.3	-9	∞	∞	∞	30	0	0	0

Figure 8.3: Intermediate results

- Next 2-3 columns depends on table are the input values at particular location.
- Following five columns are intermediate results, where first column is a result of one rule, and last column is a result of full calculation — 5 fired rules.

8.2 Any-time controller works

Let us call N a maximal number of fuzzy rules that fire for particular output variable. For both output variables direction and acceleration, N is 5.

Following experiments were performed to check influence of any-time property on controller's performance:

1. Run a controller when rules are fired in increasing order of self COG of fuzzy sets, and $N - m$ rules are fired in each iteration. Where $m < N$.

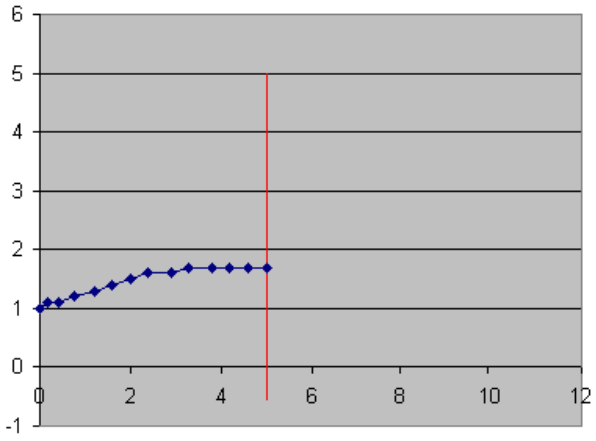


Figure 8.4: Experiment 1

2. Run a controller when rules are fired in increasing order of self COG of fuzzy sets. Let k be a number such that at iteration $k * i$ where $i \in 1, 2, 3, ...$ full calculation is performed. At rest iterations less then N rules are fired.

In addition to experiments 1, 2 described above, similar experiments 3, 4 with rules fired in decreasing order of self COG of fuzzy sets were performed.

Results

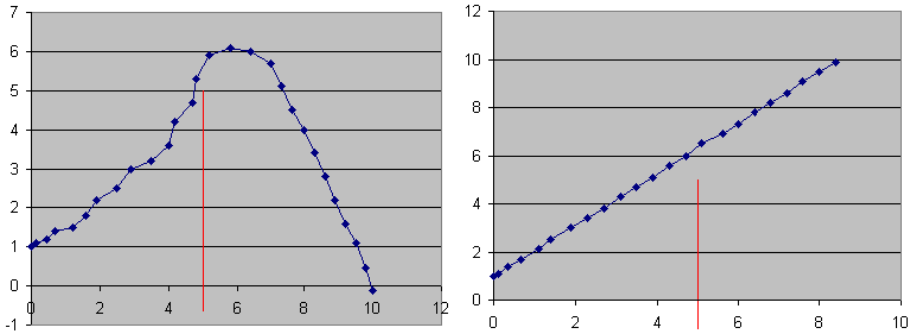
Experiment 1 — failed.

- Acceleration variable remains in unreasonable values.
- Direction variable does not reach high values. As a result it fails to avoid an obstacle.

See figure 8.4.

Experiment 3 — succeeded.

- Acceleration variable returns reasonable results with even 1 rule fires each time.
- Direction variable returns reasonable results for $N-1$ rules fired. For $N-2$ rules fired, rules that supposed to bring a robot to the goal do not fire, so



(a) $N-1$ rules fired (b) $N-2$ rules fired

Figure 8.5: Experiment 2

robot avoids an obstacle and goes on to nowhere. On the other hand, firing only 2 rules at every 7-th iteration, and performing a full calculation at rest iterations brings a robot successfully to destination.

See figures 8.5 (a) and (b). It is seen on figure that when $N-1$ rules were used, a path is less accurate, then in case of full calculation.

Experiment 2 — succeeded.

- Acceleration variable returns reasonable results in all checked configurations up to using only 1 rule, and once in 150 iterations calculate all rules (see figure 8.6). Entire walk length is ≈ 60 iterations.
- Direction variable. Robot succeeded to avoid an obstacle only with $N-1$ and $k=2$, i.e. each second iteration full calculation is performed, and when partial result is returned, it has to be based on 4 rules (see figures 8.7).

Experiment 4 — succeeded.

- Acceleration variable returns reasonable results in all checked configurations up to using only 1 rule, and once in 150 iterations calculate all rules (see figure 8.8). Entire walk length is ≈ 60 iterations.
- Direction variable. Robot succeeded to avoid an obstacle only with $N-4$ and $k=3$, i.e. at each third iteration full calculation is performed, and when partial result is returned, 1 fired rule is enough (see figures 8.9).

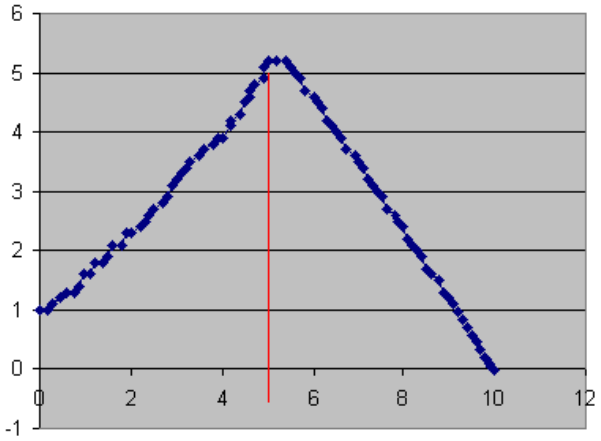
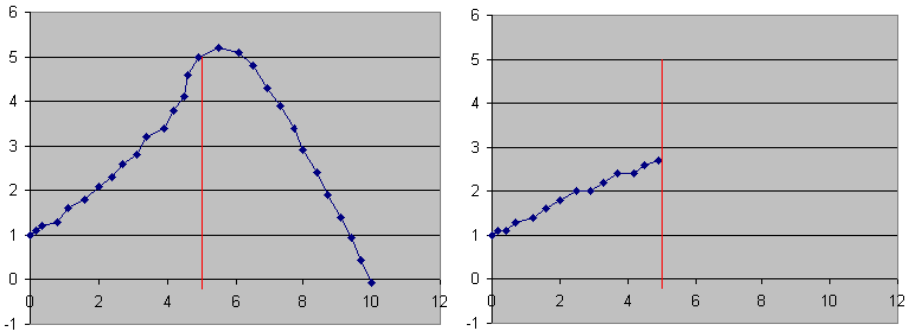
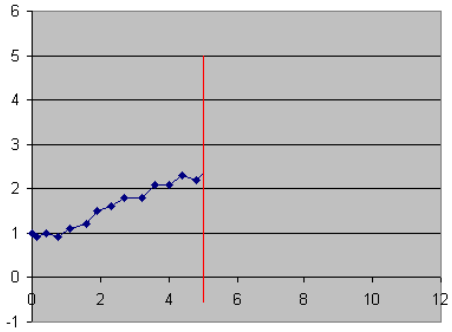


Figure 8.6: Experiment 2 acceleration variable. Once in 150 iterations use all rules, in rest cases check only one rule.



(a) $N-1, k=2$

(b) $N-1, k=3$



(c) $N-2, k=2$

Figure 8.7: Experiment 3

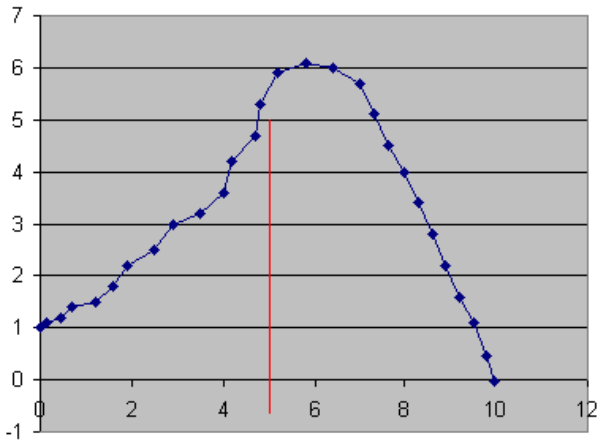
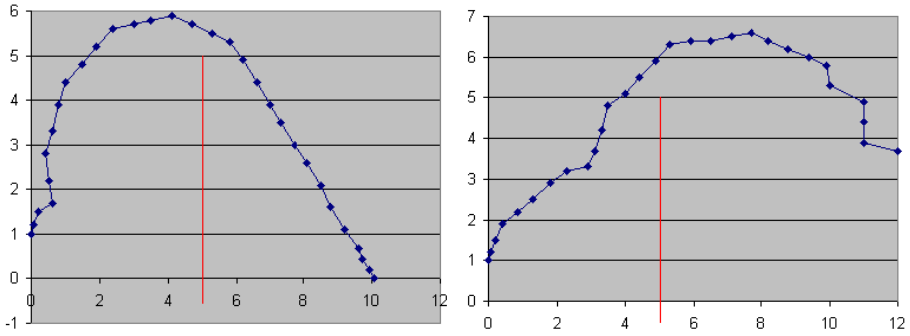


Figure 8.8: Experiment 4 acceleration variable. Once in 150 iterations use all rules, in rest cases check only one rule.



(a) $N-4, k=3$ (b) $N-4, k=4$

Figure 8.9: Experiment 4

8.3 Summary

In experiments performed on simulation of robot using any-time fuzzy controller, it was figured out that is most situations it is possible to use an intermediate result of any-time fuzzy controller as a final result. In most cases full calculation should be performed to compensate a partial result calculated earlier.

Chapter 9

Summary and Future Work

9.1 Summary

It has been shown and proven in this thesis that a general fuzzy controller may be converted to an any-time fuzzy controller by ordering fuzzy rules in a certain order of defuzzification values of output fuzzy sets. For this reason, defuzzification is an only step of fuzzy control that determines whether a controller has an any-time behavior. This makes it possible to add an any-time behavior to almost any fuzzy controller architecture. Although, aggregation and defuzzification methods checked and proven in this thesis are only a few of the well-known methods (max-min, sum-product, MOM and COG) it is shown that other combinations are possible whenever a monotony of intermediate results is kept and may be proven. Aggregation and defuzzification methods of certain cases analyzed in this thesis were:

- max-min with MOM
- sum-product with MOM
- max-min with COG
- sum-product with COG

In order to add an any-time behavior to a fuzzy controller preprocessing should be conducted. The preprocessing is a linearization. It may and should be done

once off-line, so no slow-down is caused at the run time. Three linearization techniques are introduced:

- Topological sort
- Atomic fuzzy rules
- Combination of previous two

Topological sort splits the minimal number of fuzzy rules — only rules that have to be split are changed. So minimal modifications to rule base are performed and the minimal possible size of rule base is maintained. Atomic fuzzy rules split all rules and give maximum flexibility in fuzzy rules reordering. This enables the use of any prioritizing algorithm to achieve various goals including priorities and fairness. The combination of both linearization techniques offers a trade-off between the advantages and disadvantages of both techniques.

Improvements described in this thesis give the best results in complex environments with a large database of fuzzy rules, output fuzzy variables with many fuzzy sets and complex dependencies between fuzzy rules. These properties are common in automatically learned fuzzy controllers that are widely used to handle complex environments.

9.2 Future Work

This is a first study on any-time fuzzy control that deals with major issues relating the topic. There are still some issues that should be researched in order to improve the quality of such controller's output. Issues that seem to have a significant influence are described below.

9.2.1 Unclear order

In certain cases an order of self values of the defuzzification function of two sets depends on the height of these sets. In this current study, these sets are handled as one unit. Actually, the internal order of these sets is not important in most cases

when dealing with COG. Nevertheless, the existence of a simple (in complexity terms) way to determine an order of such sets it should be investigated and checked.

9.2.2 Linearization

Linearization methods described here are:

1. Topological sort
2. Splitting fuzzy rules with a round robin
3. Combination of the previous two

Splitting rules with a round robin is quite a simple method. Fuzzy rules are split into atomic rules, and any variation of a round robin (or any other prioritizing algorithm) is used to schedule the exact order of fuzzy rules examination. An order and prioritizing algorithm depend on the desired result, e.g. priority, equal promotion of the intermediate result or any combination of them. Topological sort and its combination with a prioritizing algorithm need more research.

For certain given inputs topological sort has more then one output. Since topological sort is used to reduce fuzzy rules database size, the aim is for an output of topological sort that causes the least number of fuzzy rule splits. Further research is necessary to find out a way to obtain the optimal (or approximate) result of topological sort.

For a given output of topological sort there are fuzzy rules that may be reordered without any damage to any-time order. It is possible to use prioritizing algorithms for such fuzzy rules. In addition, the number of fuzzy rules that may be reordered depends on the output of the topological sort, thus in certain cases, output with more split rules might be preferred if more prioritizing is possible and necessary.

9.2.3 RETE networks

As mentioned in 3.3 it is hard, and perhaps even impossible, to influence rule's firing order in RETE network. Generally, this makes any-time solution incompatible

for the fuzzy controller base on RETE network. Nevertheless, a simple solution of temporal disregard of fuzzy rules that break a monotony is possible. In that case a desired order of fuzzy rules' firing is defined. When rules are fired, rules that all preceding rules have already fired according to predefined order, is taken into consideration. In that case not all the existing information is used, but the optimal is the aim. The number of rules that should fire before a first intermediate result is returned cannot be guaranteed. In the worst case scenario, the first intermediate result will be returned after all rules but one have been fired.

Further research is necessary to find out whether there is a better solution that can guarantee an intermediate result after a short period of time.

Bibliography

[1] An introduction to fuzzy control systems. http://www.faqs.org/docs/fuzzy. Last checked on 10 Sep 2006.

[2] S. Assilian. *Artificial Inteligence Techniques in the Control of Real Dynamic Systems*. PhD thesis, Queen Mary Colledge, Univercity of London, London, UK, 1974.

[3] Tristan Cazenave. Speedup mechanisms for large learning systems. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference i*, 1998.

[4] Hoffmann F. Navascues L.J. del Jesus, M.J. and L. Sánchez. Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. *Fuzzy Systems, IEEE Transactions*, 12(3):296–308, 2004.

[5] Shi Y. Eberhart R. and Chen Y. Implementation of evolutionary fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 7(2):109–119, April 1999.

[6] C. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *The Journal of Artificial Intelligence*, 19(1):17–37, 1982.

[7] Gandolfi E. Gabrielli, A. and M. Masetti. Design of a very high speed fuzzy processor by VHDL language. http://www.bo.infn.it/fuzzy/papers/96_03_parigi_EDTC.pdf. Last checked at 2004.

[8] Trautzsch T.A. Gao, Z. and J.G. Dawson. A stable self-tuning fuzzy logic control system for industrial temperature regulation. In *IEEE Industrial Ap-*

plication Society 2000 Annual Meeting and World Conference on Industrial Applications of Electrical Energy. October 2000.

[9] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1-2):139–157, 2001.

[10] R.C. Hicks. The impact of verification on rule ordering and inference. *PC AI*, 16.2 Paid version:56–60, March/April 2002.

[11] K.K. Hoe. Fuzzy expert system for navigation control, 1998. http://www.geocities.com/SiliconValley/Lakes/6165/khkoay_project.html. Last checked on 06 May 2006.

[12] Shih-Hsu Huang and Jian-Yuan Lai. High-speed VLSI fuzzy inference processor for trapezoid-shaped membership functions. *J. Inf. Sci. Eng.*, 21(3):607–626, 2005.

[13] Toru Ishida. Optimizing rules in production system programs. In *National Conference on Artificial Intelligence*, pages 699–704, 1988.

[14] Marcel Jacomet and Roger Walti. A VLSI fuzzy processor with parallel rule execution. In *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, volume 1, pages 554–558, September 1996.

[15] R. Jager. *Fuzzy Logic in Control*. PhD thesis, Delft University, 1995.

[16] R.E. King. *Computational Intelligence in Control Engineering*. Marcel Decker NY, 2005.

[17] Sun Y. Kohane I. and Stark A. Fuzzy logic control of inspired oxygen concentration in ventilated newborn infants. In *Proc Annu Symp Comput Appl Med Care*, pages 756–761, 1994.

[18] E.H. Mamdani. Applications of fuzzy algorithm for simple dynamic plant. In *Proceedings IEEE International Conference on Fuzzy Systems 121(12)*, pages 1585–1588, 1974.

[19] DeSouza G.N. Pan, J. and A.C. Kak. Fuzzyshell: A large-scale expert system shell using fuzzy logic for uncertainty reasoning. *IEEE Transactions on Fuzzy Systems*, 6(4):563–581, 1998.

[20] J. Pan and A.C. Kak. Design of a large-scale expert system using fuzzy logic for uncertainty-reasoning. In *Proceedings of the World Congress on Neural Networks*, volume 2, pages 703–708, Washington D.C., 1995.

[21] Thomas A. Runkler and Manfred Glesner. DECADE - fast centroid approximation defuzzification for real time fuzzy control applications. In *SAC*, pages 161–165, 1994.

[22] A. El Hajjaji S. Bentalba and A. Rachid. Fuzzy path tracking control of a vehicle. In *IEEE International Conference on Intelligent Vehicles*. 1998.

[23] A. Saffiotti. The uses of fuzzy logic for autonomous robot navigation: a catalogue raisonn'e. *Soft Computing Research journal*, 1(4):180–197, 1997.

[24] A. Saffiotti, E. H. Ruspini, and K. Konolige. Blending reactivity and goal-directedness in a fuzzy controller. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 134–139, San Francisco, California, 1993. IEEE Press.

[25] A. Saffiotti, E. H. Ruspini, and K. Konolige. Using fuzzy logic for mobile robot control. In H. Prade D. Dubois and H. J. Zimmermann, editors, *International Handbook of Fuzzy Sets and Possibility Theory*, volume 5. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands, 1997.

[26] H. Surmann and M. Maniadakis. Learning feedforward and recurrent fuzzy systems: a genetic approach. *Journal of Systems Architecture, Special issue on evolutionary computing*, 47(7):535–556, 2001.

[27] Hartmut Surmann. Learning a fuzzy rule based knowledge representation. In *Proceedings of 2 ICSC symposium on neural computation NC'2000*, pages 349–355, May 2000.

[28] E. Tunstel. Mobile robot autonomy via hierarchical fuzzy behavior control. In *Proc. of the World Automation Congress (WAC), ISRAM Track*, pages 837–842, Montpellier, FR, 1996. ASME Press.

[29] Wikipedia. Fuzzy logic. http://en.wikipedia.org/wiki/Fuzzy_logic. Last checked on 10 Sep 2006.

[30] Wikipedia. Fuzzy set. http://en.wikipedia.org/wiki/Fuzzy_set. Last checked on 10 Sep 2006.

[31] L.A Zadeh. Fuzzy sets. *Journal of Information and Control*, 8:338–353, 1965.

[32] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.