Proceedings of the 2007 IEEE Workshop on
Evolvable and Adaptive Hardware (WEAH 2007)

Fault Recovery Using Evolvable Fuzzy Systems

Garrison W. Greenwood
Department of Electrical and
Computer Engineering
Portland State University
Portland, OR 97207–0751
Email: greenwd@ece.pdx.edu

***Abstract*— Autonomous systems must survive for long periods without relying on humans for fault recovery. Faults may be impossible to analyze remotely and redundant hardware is frequently not allowed. The only recourse may be to replace the existing control strategy. This paper proposes a fuzzy logic controller architecture that executes a replacement control strategy for autonomous systems. I will show this architecture is ideally suited for fault recovery in autonomous systems with imprecisely defined faults. The controller is evolved *in-situ* and evaluated intrinsically.**

I. Preface

Imagine a deep-space probe speeding towards the outer reaches of the solar system. The probe, launched by NASA several months ago, measures solar wind effects. It maneuvers with three rocket side thrusters under the direction of an on-board control strategy. The probe has already passed the planet Mars and has now just entered the asteroid belt.

Suddenly the probe accidently bumps into a large rock fragment and the collision partially disables one of the side thrusters. Now the existing control strategy, which was designed to work with three fully functional side thrusters, can no longer effectively maneuver the probe. A new control strategy is needed if the probe has any chance of safely traversing the asteroid belt.

II. Introduction

Autonomous systems must often operate for long periods without relying on humans for maintenance support. Nevertheless, they must survive and continue operating when faults do occur. Fault tolerant systems can detect, isolate, and at least to some level, recover from faults automatically. Hardware redundancy is a common fault recovery method but limited space and weight requirements, such as in the space probe, won't always allow room for spares. Evolvable hardware (EH) provides an alternative to hardware redundancy. Under control of an evolutionary algorithm (EA), the hardware is automatically reconfigured until an acceptable level of functionality is restored.

Although EH is powerful, it isn't necessarily the silver bullet in all situations. Realistically every circuit in the system won't be reconfigurable. What if the fault is in a portion of the circuit that can't be reconfigured? It's not at all clear if modifying some circuitry that can be reconfigured would help. Besides, in the space probe problem described in the preface, the problem wasn't a faulty circuit. The problem was physical damage to a rocket thruster. No circuitry was involved.

A real challenge in keeping autonomous systems operational is coming up with an effective way of dealing with the uncertainty. Autonomous systems often operate in remote locations, which makes it difficult to get precise information about the nature of the fault. Furthermore, it may be extremely difficult to find out what capabilities still remain in the faulty system. This uncertainty impedes the fault recovery efforts.

One solution—perhaps the only solution—is to reconfigure the system's control strategy. The control strategy determines how the system responds to inputs from the operational environment. Change the control strategy and you change the system's behavior. This is where EH provides a real benefit because it contains all the necessary infrastructure to evolve new control strategies.

In this paper I describe how an evolved fuzzy system makes an ideal replacement control strategy because it has an inherent ability to deal with the uncertainty found in faulty autonomous systems. FLCs don't require precise information about a system before they can control that system and they can accurately control a system without mathematics. Instead, the FLC uses a natural language set of control laws to mimic human thinking. FLCs can control linear or non-linear systems and systems too complex to be described quantitatively. I therefore believe a FLC has the greatest flexibility to deal with the uncertainty surrounding an autonomous system with imprecisely defined failures.

EAs can design the FLC *in-situ*, which makes the FLC self-organizing. This feature doesn't need any simulations since the control strategy evaluation is done intrinsically—i.e., by physically implementing it and then running a small set of tests to check it's performance. Of course simply trying any randomly created control strategy would be unwise because further damage to the already faulty system is always possible. Consequently, the evolution of a new FLC is performed in a particular manner that guarantees safety. Figure 1 shows a simplified diagram of my proposed method.

Section III describes my approach. Readers unfamiliar with FLC concepts should review the appendix first.

III. Technical Description

The reader should refer frequently to Figure 1 while reading this section.

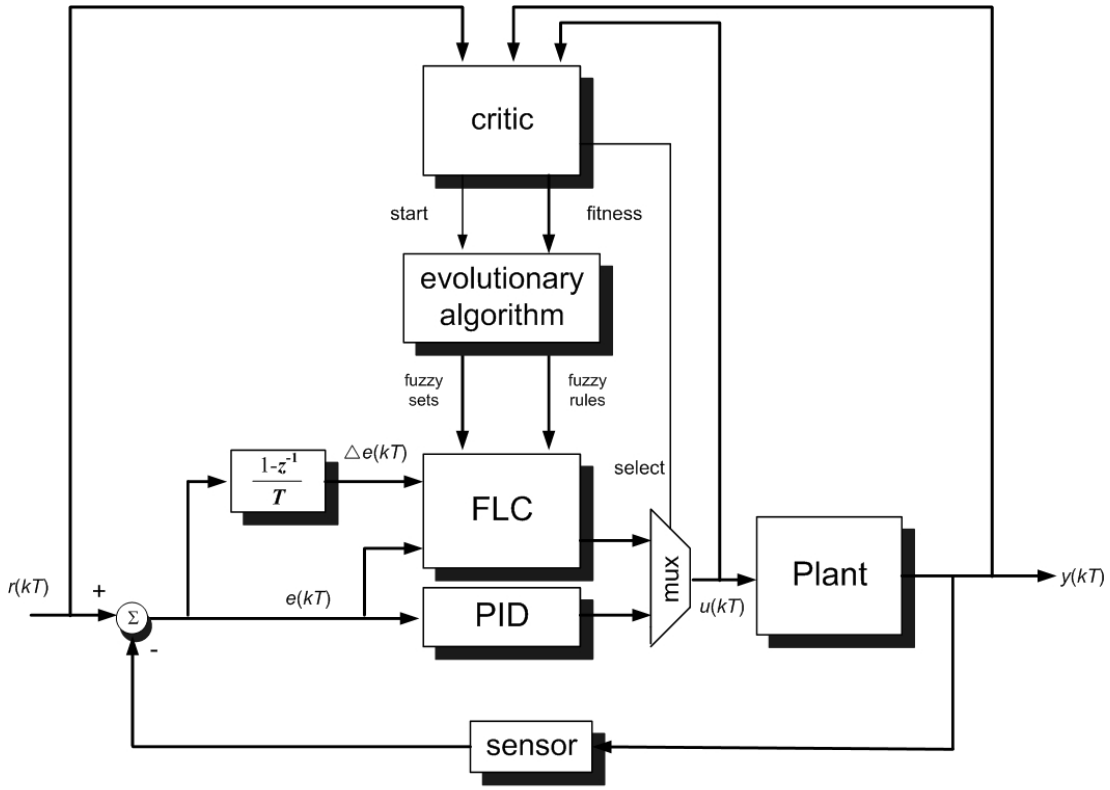

Fig. 1. The proposed system architecture. For convenience the existing control strategy is depicted as a PID controller although in practice any type of control strategy could be used. The evolutionary algorithm evolves the rule-base and database for a FLC, which becomes the replacement control strategy. Details are given in Section III.

A. The Evolutionary Algorithm

The EAs are responsible for tuning the FLC membership functions and evolving the FLC rule-base.

1) Tuning: Tuning involves setting the input signal gain values and choosing the membership function locations. Each of these tasks are described below.

Membership functions are defined over a normalized range. FLC inputs are scaled to map the crisp input value range onto this normalized range. Studies indicate FLC performance is strongly affected by this scaling [1]. Several types of scaling functions are in wide use. Many FLCs use a linear scaling function $x'_i = \beta_{0i}x_i + \beta_{1i}$ where β_{0i} and β_{1i} are constants designed to map the crisp input $x_i \rightarrow [-a, a]$. Hoffman [1] points out once this linear scaling is done a nonlinear mapping of the form

$$\tilde{x} \;=\; \text{sign}(x')|x'|^{\alpha}$$

can be done. This has the effect of making the control more sensitive to $x' \approx 0$ if $\alpha < 1$ and less sensitive for larger x' values. The opposite effect happens if $\alpha > 1$.

For this portion of the tuning a (μ, λ) ES can be used to evolve the three real numbers associated with each crisp input x_i. More specifically, for an FLC with crisp inputs x_1 and x_2, each individual in the ES population would have the form $(\beta_{01}\ \beta_{11}\ \alpha_1\ \beta_{02}\ \beta_{12}\ \alpha_2)$.

The second type of tuning manipulates the membership functions. The most straightforward approach is to pick symmetric triangular functions with fixed width and have an ES evolve their center locations. Typically the ZE (zero) membership function has its center fixed at 0 in the universe of discourse. The other membership functions can have their centers changed, but this must be done in a way that preserves the ordering. In other words, the NS membership function must never be to the left of a NB membership function. Fortunately, ordering is easy to preserve. Let c_i be the center of the triangular fuzzy set A_i. c_i should be to the left of c_j if $i < j$. One way to make sure this always is true is to move

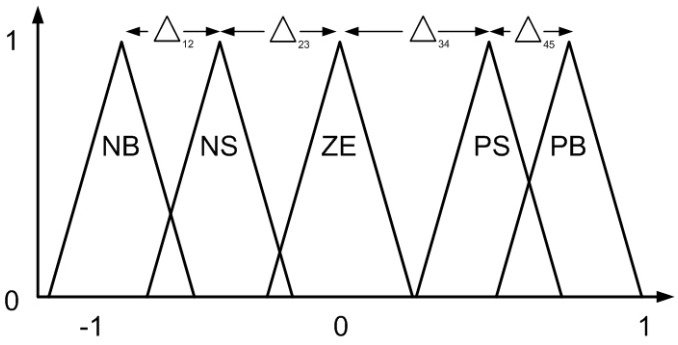

Fig. 2. Tuning of membership locations. Each $\triangle_{ij} > 0$ to preserve the order with respect to the fuzzy set ZE, which is fixed with it's center at 0.

the center locations relative to each other. The fuzzy set ZE is fixed with its center at 0 and the order is enforced by making each $\triangle_{ij} > 0$ as shown in Figure 2. In this case a (μ, λ) ES would be evolving a real-number vector $(\triangle_{12}\ \triangle_{23} \ldots)$. Another possibility, using the same real-number vector, is to fix the left and right bottom positions of the triangles and move the center locations relative to each other thereby producing asymmetric triangles like those shown in Figure 4 in Appendix A.

2) Self-Organization: An $(\mu + \lambda)$ ES evolves a new rule-base for the FLC. This process is self-organizing in the sense that input from an external expert—which is where a FLC would traditionally get it's rule-base from—isn't required here. Camazine, et.al [6] define self-organization as "... an emergent property of the system rather than a property imposed on the system by an external ordering influence". The objective is to create a rule-base that can restore as much functionality as possible. Since any information about the failures is vague, and there is no way to provide precise information about them to an external expert, the evolution must be done *in-situ* and without external assistance.

Each individual in the population is an integer array that encodes a complete set of rules. The encoding is positional, which means a particular value at a particular location in the array corresponds to a specific rule in the decision table. To see this, consider the 2-input, single output FLC decision table below.

	A_1	A_2	A_3
A_1	B_1	B_4	B_2
A_2	B_2	B_1	B_2
A_3	B_1	B_4	B_3

Each input has one of three membership functions $\{A_1\ A_2\ A_3\}$ and the output has one of four membership functions $\{B_1\ B_2\ B_3\ B_4\}$. The rows are scanned from left-to-right, top-to-bottom while recording the subscript of each B_i. For instance, the above decision table would be encoded as (1 4 2 2 1 2 1 4 3). New rules would be created by mutating the integers—i.e., replacing the current value of an integer in a randomly chosen position with a new integer from the set $\{1\ 2\ 3\ 4\}$. Another variation operator might randomly choose another individual from the population and applying a crossover operator.

3) The Fitness Function: The fitness of each FLC configuration is computed by the critic. Since the critic is defined in Section III-B, the fitness function is also described in Section III-B.

4) Creating Safe FLC Configurations: Every FLC configuration is evaluated intrinsically—i.e., physically implemented and tested in hardware. Whenever intrinsic evolution is used, it is absolutely essential the control strategy be safe to prevent harm to the system while it undergoes evaluation. This requirement is achieved by making sure all control strategies pass a safety check before they are downloaded for evaluation.

Safety is the one area where an expert has to get involved. Fortunately, that involvement can be occur before the autonomous system is deployed. A designer could formulate safety conditions and then convert them to a specific set of fuzzy rules. Indeed, the antecedents for these safety rules can use the same input fuzzy sets as used for the operational rule-base. However, the consequent in these safety rules specifies which output fuzzy sets are NOT acceptable—i.e., safety rule consequents are always complemented because they describe forbidden actions. For example, a safety rule might be

if θ is NB **AND** $\triangle\theta$ is NB
then the force **F** is $\neg NB$

where "$\neg$" represents the **NOT** operation. The semantics of the above rule is if θ and $\triangle\theta$ are both negative and large, then the output force should not be negative and large. All safety rules are collected into a safety rule-base.

Every rule evolved by the ES must be checked for safety. A safety check is performed by comparing the evolved rule against every rule in the safety rule-base. An evolved rule is discarded if there is a complemented safety rule—i.e., a safety rule that is identical except for a complemented consequent. Safety rules always take precedence over evolved rules.

It is important to note that the safety rules are invariant. It doesn't matter what the control strategy is—FLC, PID or something else—because the safety properties are with respect to the plant and not the control strategy. The EA doesn't modify anything in the safety rule-base. Safety rules remain fixed throughout the lifetime of the system.

Finally, it is worth mentioning some FLC actions may not necessarily damage the plant, but they may produce unstable plant behavior. The expert should be mindful of this possibility and add rules to the safety rule-base as needed to prevent undesired plant behavior.

B. The Critic

The critic block is essential to the self-organizing process and it is responsible for several tasks. First, it performs fault detection by monitoring the difference between the actual system step response and the step response of a fault-free system. (How this is done is described below.) If the error is too large, the critic sends a start signal to the EA, telling it to begin execution, and a select signal to the analog multiplexer to switch from the PID controller to the FLC. A series of step inputs to the system evaluate each FLC configuration produced by the EA. The critic uses the resultant $y(kT)$ response to compute the fitness value of that configuration for the EA.

The critic contains an internal lookup table to store the step response of a fault-free system at time kT with $k = 0, 1, \ldots, L$. Referring to Figure 1, notice the critic also receives the input step response $r(kT)$. When $r(kT)$ exceeds 1.0 volts the lookup table entries are extracted and compared against the actual system's step response. Let $\tilde{y}_k$ denote the k-th entry in the lookup table. The error at time $t = kT$ is $\eta_e(kT) = |y(kT) - \tilde{y}_k|$.

The critic uses an internal watchdog timer to autonomously perform fault detection. The timer is initialized to a value

N and decrements at regular intervals determined by a clock signal. However, the counter is enabled at time $t = kT$ if and only if $\eta_e(kT) > \epsilon$ where ϵ is a user-defined threshold. This means the timer won't necessarily decrement on every clock pulse. If the timer reaches zero before say K clocks, then the error has been too high for an excessive amount of time. In this case the current control strategy is assumed to be faulty and the fault recovery process described above begins. If the count is not zero after K clocks, the timer is re-initialized to N so another detection cycle can begin.

Once a candidate rule-base is downloaded to the FLC, the system receives a series of step inputs. Each step input has a different magnitude to ensure all input membership functions are exercised. (The lookup table has the step response for these step inputs as well.) The critic now uses $\eta_e(kT)$ to compute a fitness value for the configuration undergoing evaluation. The fitness function is similar to one often used in optimal control problems. Specifically, the fitness of an FLC configuration equals $1/J$ where

$$J = \sum_{k=0}^{L} [P\eta_e^2(kT) + Qu^2(kT)] \quad (1)$$

P and Q are both scalars. P is the penalty for too large a error between a fault-free system and the plant whereas Q is the penalty for too large an input to the plant. The objective is to find an FLC configuration with a minimum J value because that configuration has the highest fitness.

Depending on the nature of the fault it may or may not be possible to get $\eta_e(kT) < \epsilon$ even with an optimal FLC. In this situation the user should choose some lower bound η_L where any FLC that gives a $\eta_e(KT) < \eta_L$ for $k = 0, 1, \ldots, L$ is deemed acceptable. At this point the EA can terminate and the error threshold ϵ is set equal to η_L. The lookup table can remain unchanged.

C. FLC

Referring to Figure 1, the FLC has two inputs and one output. Seven fuzzy sets

$$\{NB, NM, NS, ZE, PS, PM, PB\}$$

are used for each input and five fuzzy sets

$$\{NB, NS, ZE, PS, PB\}$$

are used for the output. The membership functions for both inputs are fixed-width triangles whereas the output fuzzy sets are singletons. (As a side note, this gives 5^{49} possible rule-bases.)

An FLC configuration requires a database and a rule-base. A (μ, λ) ES tunes the membership functions in the database and a $(\mu+\lambda)$ ES evolves the rules for the rule-base. Each individual in the ES encodes either an entire database or a complete rule-base with 49 total rules. Each individual is assigned a fitness value, which is determined intrinsically. To do this intrinsic evaluation the individual (database or rule-base) is physically implemented in the FLC, and a series of steps are applied to the physical system as described in the previous section. This process is repeated for every individual in the population. The evolutionary algorithms terminate when a suitable FLC configuration is found.

Normally the rule-base is evolved first followed by database tuning. But a database is still needed while the rule-base is evolved. The simplest solution centers the ZE membership function at 0 and then equally distributes the rest of the triangular membership functions throughout the universe of discourse.

IV. Implementation Details

The FLC, the EAs and the critic can all be implemented in software on a microcontroller. The microcontroller platform must have the following features:

- a multi-channel analog-to-digital converter
- a digital-to-analog converter
- a watchdog timer (for the critic)
- multiply-accumulate circuitry (to do the fitness calculation)
- JTAG interface (to download code)

The above requirements are satisfied with a DSP microcontroller. The Texas Instrument's TMS320C24x family is an excellent choice [5].

V. Discussion

FLCs have been successfully used in control applications ranging from washing machines to transmission systems in automobiles, in aircraft sensor management systems and even in nuclear fusion experiments. Experience shows a FLC yields results superior to those obtained by conventional control algorithms [3].

The novelty of my approach comes from several fronts:

1) FLC have previously been used for fault recovery, but those FLCs were pre-designed and only served as a backup for the primary controller should that controller fail. In other words, the FLC was not designed to handle faults in the plant. My FLC provides fault recovery for faults in the primary controller or in the plant.
2) My FLC is self-organizing, which means no external expert input is required to create the database or rule-base needed for the fault recovery. Consequently, my FLC can handle faults never anticipated by the original designers.
3) The fault detection process is fully automatic requiring no user involvement.
4) The rule-base is evolved through a stochastic process. Nevertheless, the evolved rules are guaranteed safe.
5) All candidate FLC configurations are tested intrinsically—i.e., in hardware and in the actual operational environment. No simulation is required.

The FLC architecture I proposed here was described as a software implementation. Some readers may believe it therefore might not qualify as "EH". On the contrary. First, FLCs have been implemented in hardware for some time now

(e.g., see [4]). It therefore is entirely reasonable to expect a FLC could be put into a suitable reconfigurable device, such as an FPAA, and all of the concepts I presented would still apply. Second, and more importantly, evolvable hardware refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment [7]. Clearly my evolved fuzzy system controls hardware behavior. All architectures are autonomously evolved over time and verified intrinsically. Hence, my evolved fuzzy system satisfies all of the criteria needed to qualify as EH.

Finally, the literature often refers to a FLC designed by an evolutionary algorithm as a genetic fuzzy system. I purposely changed that name because the word "genetic" implies a GA did the evolution. My choice of "evolved" fuzzy system indicates other evolutionary algorithms such as an ES or EP could be used for the evolution.

VI. Final Remarks

EH work is in two problem domains: original design and adapting existing designs to compensate for failures or a changing environment. Hoffman [1] gave an overview of past evolvable fuzzy system research. All of this prior work involved the original design of FLCs for robot control. But Hoffman didn't seem very impressed with the results obtained from this prior research—and with good reason. None of the results were really impressive and could even be considered mediocre. Indeed, Hoffman's assessment of his own work on evolving a FLC to help a robot avoid obstacles summarizes the state of evolvable fuzzy system research (as of 2001):

> "*. . . the purely reactive obstacle-avoidance behavior did not pose a particularly challenging task, and it remains an open question whether evolutionary techniques are able to design more complex robot behaviors that are otherwise difficult to conceive manually.*"

I surveyed papers on evolved fuzzy system robot control published in IEEE journals after Hoffman's paper was published and concluded nothing much has changed since then. Wall following problems remain popular [8]–[10], but those problems are only mildly interesting because they don't demonstrate any real complex behavior, and frankly don't need any either.

The true potential of evolvable fuzzy systems hasn't yet been determined. Hoffman's open question will remain open until the type of investigations currently done begins to change. I am not convinced that contrived laboratory experiments are helping to get any answers. Commercial, military and space agencies currently have requirements for autonomous systems that can't be fulfilled because existing systems can't provide sufficiently complex behavior. That's what researchers should be working on now. A robot that can autonomously navigate over open terrain would generate far, far more interest than a wall following robot ever will.

Appendix: Introduction to Fuzzy Logic Controllers

This appendix gives a brief introduction to FLCs. More detailed information is available in a number of excellent textbooks (e.g., see [2]).

Fuzzy control is based on fuzzy logic, which uses natural language to express human thinking. A FLC implements a control strategy with a set of linguistic control rules. Put simply, a FLC takes an expert's knowledge about how to control a system, expressed in natural language as a set of control laws, and converts that knowledge into an actual control strategy.

Fuzzy logic sets differ from conventional Boolean sets. In Boolean or *crisp sets* a parameter is either in the set (TRUE) or not in the set (FALSE). In fuzzy sets a parameter can be partially in a set and the degree to which it is in the set is determined by a *membership function*. Let $M_A(\cdot)$ denote the membership function associated with the fuzzy set A. For a crisp input x, $v = M_A(x)$ is on the unit interval where $v = 0$ means $x \notin A$, $v = 1$ means $x \in A$ and $0 < v < 1$ means x is partially in A.

Figure 3 shows membership functions for a parameter x that might represent say a value from a pressure sensor. Note the range of x values is normalized between -1 and 1. This is a common practice. The range over which the membership functions are defined is called the *universe of discourse*.

The dashed line shows the degree of membership a crisp value of $x = -0.44$ has in two fuzzy sets. Notice x is partially in the negative big set (NB) and partially in the negative small set (NS) with membership values 0.24 and 0.52, respectively. This captures the vagueness about accurately classifying x values. The membership values says it is believed -0.44 belongs in both sets, but more so in the NS set than the NB set. Both triangular and a trapezoidal membership functions are depicted in the figure, but bell-shaped, gaussian and singleton membership functions are also commonly used. Indeed, many FLCs use singletons as the output fuzzy sets. The membership function for singletons is

$$M_{\text{singleton}}(x) = \begin{cases} 1 & \text{if } x = u \\ 0 & \text{otherwise} \end{cases}$$

Fuzzy sets have behaviors similar to their crisp counterparts. In fuzzy sets an AND operation of two inputs takes the smaller value, the OR operation takes the larger value and NOT takes one minus the input.

Figure 4 shows the basic components of a FLC. Essentially fuzzy control involves three phases:

1) fuzzification (crisp input mapping to fuzzy sets)
2) inference (rule evaluation)
3) defuzzification (fuzzy set conversion to a crisp output)

The database holds problem specific information. It specifies any scaling requirements for the inputs or outputs and it defines the input and output membership functions. The fuzzification process takes a crisp input x, scales it if necessary, and

then determines it's membership value in various fuzzy sets using the membership functions in the database.

In FLCs the control strategy is specified in a rule-base describing how an expert would do things. For example, in a typical FLC a rule might be

> **if** θ is large and negative **AND** the change in θ ($\triangle\theta$) is large and negative
> **then** the force **F** is positive and large

All rules are collected into a rule-base and all rules have an **if-then** format. The **if** side is called the condition or *antecedent* and the **then** side is called the conclusion, action or *consequent*. A condition such as "θ is PB" is called a fuzzy proposition and $M_{\mathrm{PB}}(\theta)$ represents the fuzzy equivalent of θ. The consequent of a rule is also a fuzzy set.

Many rules, such as the example shown above, have more than one condition in the antecedent. An *aggregation operation* determines the firing strength of the rule based on the individual condition degree of membership. For instance, in a rule with a fuzzy membership value for θ and one for $\triangle\theta$, their aggregation might be

$$M_{NB}(\theta) \textbf{ and } M_{NB}(\triangle\theta) \;=\; \min(M_{NB}(\theta), M_{NB}(\triangle\theta))$$

The firing strength for the rule reflects an uncertainty about how applicable that rule is. Remember there is only a vague notion about the accuracy of conditions, which is reflected in the membership in various fuzzy sets. If there is uncertainty about the set membership, there is likewise uncertainty about any rule that has those variables in its antecedent.

A *decision table* is frequently used to display a rule-base. The if-then rule above is in the first row and first column of the following decision table.

		$\triangle\theta$		
	F	NB	ZE	PB
	NB	PB	ZE	NB
θ	ZE	PB	ZE	NB
	PB	NB	ZE	ZE

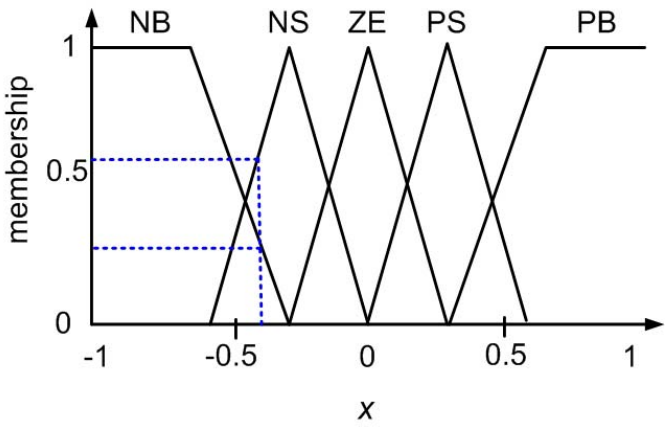

Fig. 3. Example membership functions for $-1 \leq x \leq 1$.

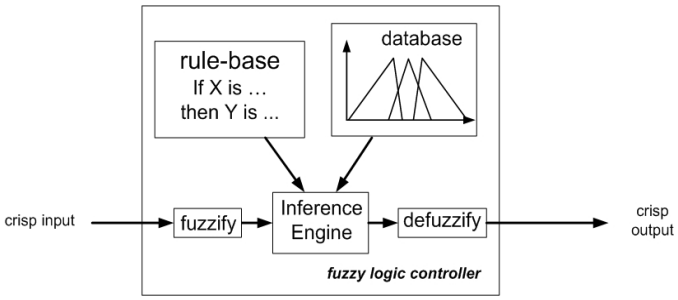

Fig. 4. A basic FLC

The inference engine determines which rules are active and combines their recommended action into a single fuzzy conclusion. Inference modifies the shape of the output fuzzy set, specified in the consequent, based on the degrees of membership in the antecedent. These modified output fuzzy sets are called *implied fuzzy sets* because they specify the certainty with which the crisp output should take the i-th rule into consideration.

A usable crisp output must be generated once the FLC has applied the inputs to the rule-base. Defuzzification does that. There are several defuzzification methods but *center of gravity* (COG) and *centroid* are the most widely used. With singletons the COG defuzzification is very straightforward and computationally efficient. The crisp output value u is

$$u = \frac{\sum_i \alpha_i s_i}{\sum_i \alpha_i}$$

where s_i is the i-th singleton value and $\alpha_i \in [0, 1]$ is the firing strength of the i-th rule.

REFERENCES

[1] F. Hoffman. Evolutionary algorithms for fuzzy control system design. *Proceedings of the IEEE*, 89(9):1318–1333, 2001.

[2] Z. Kovacic and S. Bogdan. *Fuzzy Controller Design: Theory and Applications*. CRC Press, 2006.

[3] C. Lee. Fuzzy logic in control systems: fuzzy logic controller—part I. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):404–418, 1990.

[4] M. Patyra (Guest Editor). Guest editorial: Fuzzy logic hardware implementations. *IEEE Transactions on Fuzzy Systems*, 4(4), 1996.

[5] TI TMS320LF24X family of DSP microcontrollers. http://www-s.ti.com/sc/ds/tms320lf2403a.pdf.

[6] S. Camazine, J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[7] G. Greenwood and A. Tyrrell. *Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*. Wiley, 2006.

[8] S. Lee and S. Cho. Emergent behaviors of a fuzzy sensory-motor controller evolved by genetic algorithm. *IEEE Transactions on Systems, Man and Cybernetics—Part B*, 31(6):919–929, 2001.

[9] K. Sim, K. Byun, and D. Lee. Design of fuzzy controller using schema coevolutionary algorithm. *IEEE Transactions on Fuzzy Systems*, 12(4):565–568, 2004.

[10] H. Hagras. A hierarchical type-2 fuzzy logic control architecture for autonomous mobile robots. *IEEE Transactions on Fuzzy Systems*, 12(4):524–539, 2004.