

Fuzzy Logic Controllers: from Development to Deployment*

Piero P. Bonissone and Kenneth H. Chiang
Artificial Intelligence Laboratory
General Electric Corporate R&D
Schenectady, New York 12301
Bonissone@crd.ge.com ChiangK@crd.ge.com

Abstract— We view fuzzy logic control technology as a high level language in which we can efficiently define and synthesize nonlinear controllers for a given process. We contrast fuzzy Proportional Integral (PI) controllers with conventional PI and two dimensional sliding mode controllers. Then we compare the development of Fuzzy Logic Controllers (FLC) with that of Knowledge Based System (KBS) applications. We decompose the comparison into reasoning tasks (representation, inference, and control) and application tasks (acquisition, development, validation, compilation, and deployment). After reviewing the reasoning tasks, we focus on the compilation of fuzzy rule bases into fast access lookup tables. These tables can be used by a simplified run-time engine to determine the FLC's crisp output for a given input.

I. INTRODUCTION

The number of applications of Fuzzy Logic Controllers (FLC) [1] has increased considerably over the last few years. These applications range from the development of autofocus cameras[2], to the control of subway trains [3], cranes [4], domestic appliances, automobile sub-systems [5], and consumer electronic products.

Fuzzy Logic Controllers are knowledge based systems in which the knowledge of process operators or product engineers has been used to synthesize closed loop controllers for given processes.

Classical controllers are derived from control theory techniques based on mathematical models of the open-loop process to be controlled. Typically, these controllers use rather accurate (and therefore expensive) sensors to monitor the output of the process.

*A modified version of this paper will appear in *Fuzzy Sets, Neural Networks and Soft Computing*, edited by R. R. Yager and L. A. Zadeh, and published by Van Nostrand Reinhold.

The precision is required by the control algorithm that has been synthesized from the process model. An actuator takes the output signal generated by the controller, transforms that signal into an action, and executes it.

On the other hand, FLCs are typically derived from a knowledge acquisition process or are automatically synthesized from a self-organizing control architecture [6]. In either case, the result of the synthesis is a Knowledge Base (KB), rather than an algorithm. The KB consists of a set of fuzzy rules and termsets, which are evaluated by an interpreter. The interpreter is composed of a quantification (or fuzzification) stage, an inference engine (or fuzzy matcher), and a defuzzification stage. The quantification makes the sensor-generated input to the controller dimensionally compatible with the Left-Hand Side (LHS) of the rules. The inference engine matches the LHS of all the rules with the input, determines the partial degree of matching of each rule, and aggregates the weighted output of the rules, generating a possibility distribution of values on the output space. The defuzzification summarizes this distribution into a point that is used by the actuator as the resulting control action. Figure 1 (a, b) shows the typical configuration of classical and fuzzy controllers. The actuators (A) and sensors (S) are common to both configurations.

The use of an interpreter requires the evaluation of *all* the rules in the KB at every iteration. There is, however, another alternative. Under certain conditions, we can compile the KB, like a programming language or a knowledge base application, and use a simpler run-time engine. The result of this compilation process is a lookup table that allows for a faster, more efficient execution that can be performed by simpler processors. Not only is the response time reduced, but the memory requirements are so drastically decreased that it is possible to implement the FLC using a very small amount of memory. This

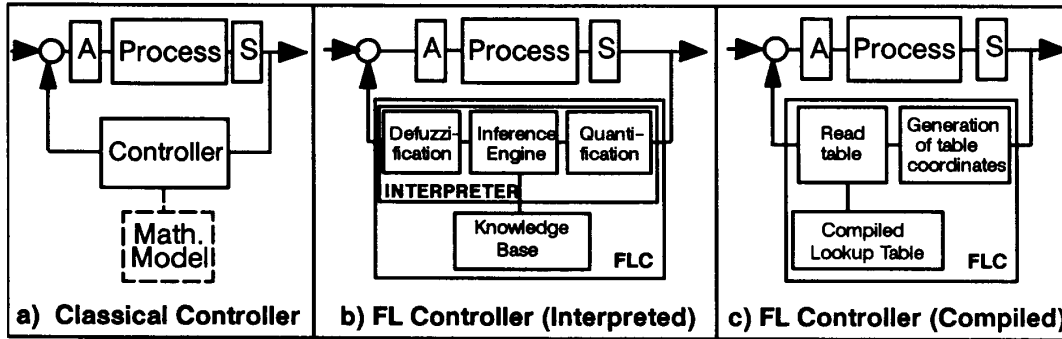


Figure 1: Comparison between traditional and fuzzy logic controllers.

feature enables us to implement inexpensive FLCs for cost-sensitive applications. Figure 1 (b, c) provides a comparison between the interpreted and the compiled versions of the fuzzy controller.

A. Structure of the paper

First we will compare FLCs with conventional controllers: in particular we will contrast conventional PI and two dimensional sliding mode controllers with fuzzy PI controllers. Then we will analyze the Fuzzy Logic Controllers from a Knowledge Based System perspective. We will briefly describe the main issues in the reasoning tasks (knowledge representation, inference, and control). Then we will describe the application tasks (knowledge acquisition, development, validation, compilation, and execution). Finally, we will focus on the compilation process, which is described in more detail in [7].

II. CONVENTIONAL PI, FUZZY PI, AND SLIDING MODE CONTROL

The fuzzy Proportional Integral (PI) controller is one of the most common fuzzy logic controllers. This controller is defined by a customized nonlinear control surface in the (e, \dot{e}, du) space. In this section we will compare the fuzzy PI with some of its conventional counterparts, namely the conventional PI and the two dimensional sliding mode controller.

A. Conventional PI controllers

A conventional proportional-integral controller can be described by the function

$$\begin{aligned} u &= K_p e + K_i \int e dt \\ &= \int (K_p \dot{e} + K_i e) dt \end{aligned}$$

or by its differential form

$$du = (K_p \dot{e} + K_i e) dt$$

The proportional term provides control action equal to some multiple of the error, while the integral term forces the steady state error to zero. Otherwise, the controller will always force a change in the manipulated variable.

Let e be defined as the set point subtracted from the actual value of a given signal, and let positive \dot{e} denote an increasing rate of change of e . Assume a control law that requires a high positive du to counteract a high negative e with a high negative \dot{e} and a high negative du to counteract a high positive e with a high positive \dot{e} . Also assume that the goal of the control law is to bring the system to the equilibrium point of zero e and zero \dot{e} . In a three dimensional space with axes e , \dot{e} , and du , the control surface du of a conventional PI would be a plane passing through the origin and oriented at some angle with respect to the e - \dot{e} plane, the angle determined by the particular values of K_p and K_i , as shown in Figure 2 (a).

Once initial values of K_p and K_i have been determined by the Zeigler-Nichols method, a number of heuristics are used to fine tune those values. Increasing K_p causes the rise time to decrease, because the error will be amplified and the controller will output a greater controller action. However, a large increase in K_p will also cause the controlled variable to overshoot its steady state value, and the oscillations about that value to markedly increase. Decreasing K_i will reduce the overshoot of the controlled variable at the expense of the rise time, because the integral of the error will be attenuated.

B. Fuzzy Logic PI Controllers

A fuzzy logic PI controller is described by a Knowledge Base (KB). The KB is composed of a *rule set*, *termsets*, and *scaling factors*. The rule set maps linguistic descriptions of state vectors $[e, \dot{e}]$ into incremental control actions du ; the termsets define the

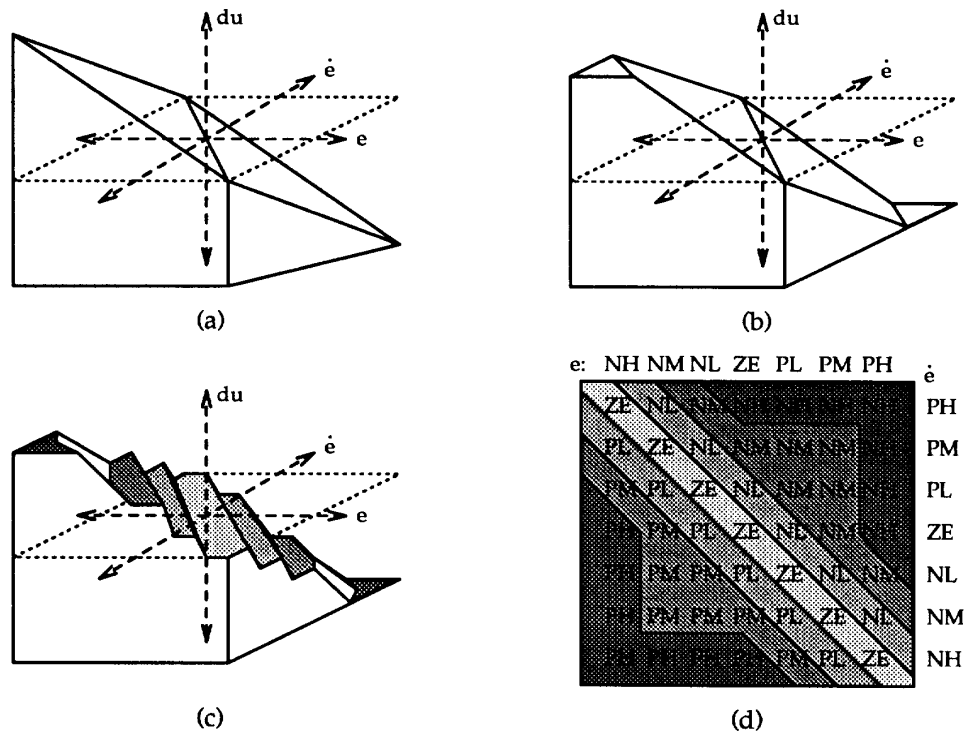


Figure 2: The control surface of: (a) a conventional PI controller; (b) a two dimensional sliding mode controller; (c) a fuzzy logic PI controller; and (d) the rule set defining the fuzzy PI's surface.

semantics of the linguistic values used in the rules; and the scaling factors determine the extremes of the numerical range of values for both the input and output variables. Using fuzzy logic, a step-like control surface with gradations between the steps can be synthesized to generalize the control surface of the conventional PI, as can be seen in Figure 2 (c).

Reference fuzzy sets are defined for e , \dot{e} , and du in their corresponding termsets. Similarly, scaling factors N_e , $N_{\dot{e}}$, and N_{du} are also defined to determine the range of values for e , \dot{e} , and du , e. g. $-N_e \leq e \leq N_e$. In the rule set, a distribution for the controller output du is defined for each combination of the linguistic sets for e and \dot{e} , as illustrated in Figure 2 (d). In the figure, e has been divided into seven fuzzy sets; PH is positive high, PM positive medium, PL positive low, ZE zero, NL negative low, NM negative medium, and NH negative high. \dot{e} has also been divided into the reference fuzzy sets with the same linguistic labels. It is important to note that e and \dot{e} are not defined over the same universe of discourse, so their membership functions need not be identical. The rules have an intuitive interpretation. For example, if e has a negative medium value and

\dot{e} has a negative low value, then the error is slowly increasing. Thus, the appropriate control action is a positive medium increase in u . If the membership functions for e and \dot{e} are properly defined (typically overlapping by twenty-five percent) and if either or both e and \dot{e} happen to fall into the overlapping area, two or more rules will fire. The controller output du will be an interpolation of the du values for each firing rule. This results in the gradations in the control surface.

From the comparison of Figure 2 (a) and Figure 2 (c), it is obvious that a larger number of parameters must be defined to specify the nonlinear surface. As summarized in Figure 3, the linear controller requires only a gain vector, whereas the fuzzy controller needs a rule base, termsets, and the equivalent of the gain vector, represented by the input and output scaling factors.

This rule set and the associated termsets define the contents of the knowledge base for the fuzzy logic PI. The fuzzy logic analogue of K_p and K_i are reflected in the normalizing factors of the termsets for e and \dot{e} . In particular, K_p is approximately equal to N_{du}/N_e , while K_i corresponds to $(N_{du}/N_{\dot{e}})df$, where

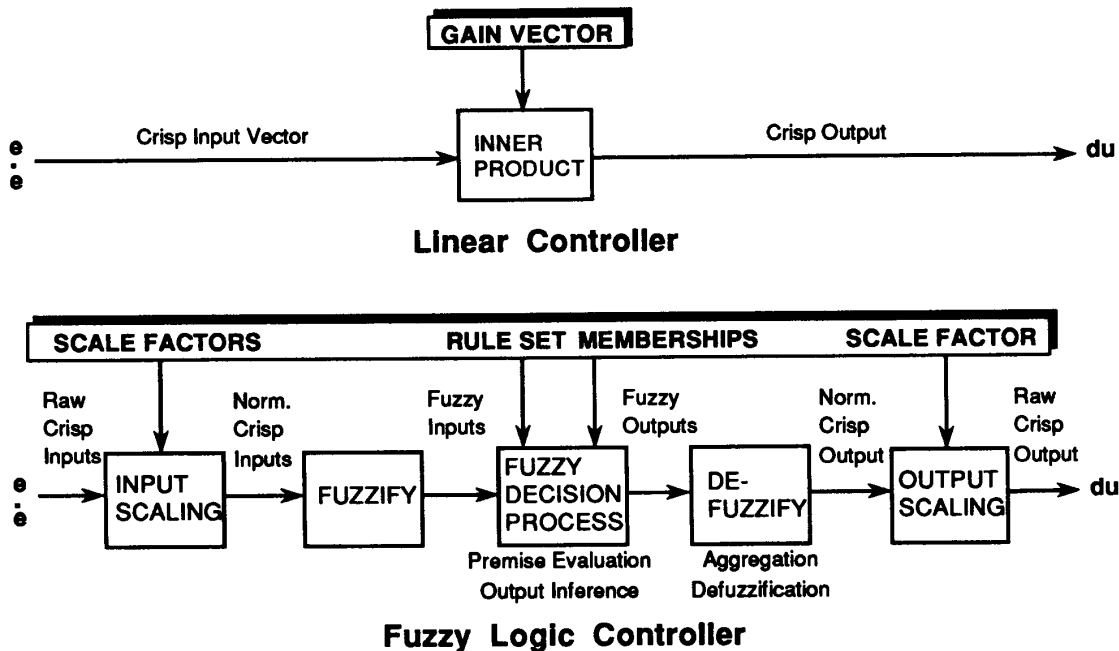


Figure 3: Design parameters for linear and fuzzy PI controllers.

df is $1/dt$ [8]. By increasing N_d , K_p is decreased. Likewise, by increasing N_e , K_i is decreased [9]. Similarly, the normalizing factor of the termset for the incremental control action du is directly proportional to both K_p and K_i .

Small variations in the error or in the error derivative have less effect on the fuzzy logic PI. In the conventional PI, perturbations in e and \dot{e} in the direction of or against the gradient of the control surface would cause du to change greatly. Examining the control surface of the fuzzy logic PI reveals that it is mostly parallel to the e - \dot{e} plane. Only if the perturbation forced e and \dot{e} into a transitional region where two or more rules would fire would the perturbations have any effect on the controller output.

Since the fuzzy PI is a nonlinear controller, we will extend its comparison to a conventional nonlinear controller, implemented using sliding mode control. In the remainder of this section we will provide an interpretation of the fuzzy logic PI in terms of a two dimensional sliding mode control. A detailed explanation of the sliding mode control and of its generalization to the FLC can be found in references [10] and [11], respectively.

C. Two Dimensional Sliding Mode Controllers

For second order systems, the problem of forcing state vector $\vec{x} = [x \ \dot{x}]$ to track a desired vector $\vec{x}_d = [x_d \ \dot{x}_d]$ can be reduced to the problem of keeping the function

$$s = \dot{e} + \lambda e$$

as close to zero as possible, where e is the tracking error $x - x_d$, \dot{e} is the tracking error derivative $\dot{x} - \dot{x}_d$, and λ is some problem-specific constant.

In two dimensions, the line defined by the equation $s = 0$ is termed the *switching line*. A sliding mode controller attempts to drive the error vector onto the switching line as rapidly as possible, and then force it to the equilibrium point $[e \ \dot{e}] = [0 \ 0]$. This is accomplished by defining the control law u as follows:

$$u(s) = \begin{cases} +K & \text{if } s > 0 \\ 0 & \text{if } s = 0 \\ -K & \text{if } s < 0 \end{cases}$$

However, the discontinuity at the switching line $s = 0$ causes extremely high control action if the system does not settle onto the switching line. To remedy this problem, the discontinuity can be smoothed out by a gradation in the region $|s| \leq \Phi$, so the con-

trol law becomes

$$u(s) = \begin{cases} +K & \text{if } s > \Phi \\ +K \frac{s}{\Phi} & \text{if } |s| \leq \Phi \\ -K & \text{if } s < -\Phi \end{cases}$$

The sliding mode controller is linear in the region close to the switching line. For the fuzzy logic PI, the controller output du can be given exponential gains in the region $|s| \leq \Phi$. For instance, if the membership functions for low magnitude du had their centers of mass a distance d away from the origin, the membership functions for medium magnitude could have centers of mass $2d$ away from the origin, and those for high magnitude $4d$ away. This would cause the state vector to approach the switching line faster, reducing rise time.

Settling time can also be reduced by placing a deadband around the switching line close to the equilibrium point. This is done by defining the membership functions for positive low and negative low error so that they stop some small distance away from the point at which the error is zero. The same is done for the error derivative. Thus, when the equilibrium point is approached, there is no change in controller output.

Finally we note that by tuning a fuzzy PI, we can smooth the step-like control surface and we can modify the switching line and generalize it to be a smoother higher-order curve, as illustrated in [12].

III. THE REASONING TASKS

Three main reasoning tasks are common to Fuzzy Logic Controllers and Knowledge Based Systems: the knowledge representation, the inference mechanism applicable to the chosen representation and the control of the inference. Due to space constraints, we will limit our description to the reasoning tasks for the FLC. For a description of KBS reasoning tasks, organized within the same framework, the reader is referred to references [13].

A. The Knowledge Representation

The main representational issues for the FLC are: the quantification of the input; the termset definition for each state variable and for each control action; the definition of the type of fuzzy production rule to be used in the KB.

The input quantization consists of describing the input as a fuzzy subset of the input space. This process is necessary to make each input element dimensionally compatible with each state variable. When the universe of discourse of the state is discretized, the FLC designer needs to determine a mapping from

intervals of the universe of discourse to its point representation. When the universe of discourse is continuous, this mapping is not needed. For a formal description of this process the reader is referred to [7].

The definition of the termset is perhaps the most important of the representational issues. For each state variable and control action, we need to define the granularity of their values [14]. Thus, we must determine the cardinality of termsets used to represent each state variable and action, the semantics of the above termsets, and the scaling factors for each state variable and action. These design choices are crucial to issues such as steady-state performance and stability. In particular we have observed that the FLC steady-state behavior improves considerably when the termset provides finer granularity around the equilibrium point [15].

The most common definition of a fuzzy rule set R is the disjunctive interpretation found in the majority of FLC applications [16], i.e.,

$$R = \bigcup_{i=1}^m r_i = \bigcup_{i=1}^m (\overline{X_i} \rightarrow Y_i) \quad (1)$$

R is composed of m rules. Each rule r_i defines a mapping between a fuzzy state vector $\overline{X_i}$ and a corresponding fuzzy action Y_i . r_i is typically represented by the Cartesian product operator. There are, however, many other representations for a fuzzy rule, which are based on the material implication operator and the conjunctive interpretation of the rule base. For a definition of different rule types, the reader is referred to references [17] and [18].

B. Inference Engine

The inference engine of a FLC can be defined as a parallel forward chainer operating on fuzzy production rules. An input vector \overline{I} is matched with each n -dimensional state vector $\overline{X_i}$, i.e., the Left Hand Side (LHS) of rule $r_i = \overline{X_i} \rightarrow Y_i$. The degree of matching indicates the degree to which the rule output can be applied to the overall FLC output. The main inference issues for the FLC are: the definition of the fuzzy predicate evaluation, which is usually a possibility measure [19]; the LHS evaluation, which is typically a triangular norm [20, 21]; the conclusion detachment, which is normally a triangular norm or a material implication operator; and the rule output aggregation, which is usually a triangular conorm for the disjunctive interpretation of the rule base, or a triangular norm for the conjunctive case.

Under the most commonly used assumptions [7]

we can describe the output of the FLC as

$$\mu_Y(y) = \bigvee_i^m (\min[\lambda_i, \mu_{Y_i}(y)]) \quad (2)$$

where λ_i is the the degree of applicability of rule r_i

$$\lambda_i = \bigwedge_j^n \Pi(X_{i,j} | I_j) \quad (3)$$

and $\Pi(X_{i,j} | I_j)$ is the possibility measure representing the matching between the reference state variable and the input element¹

$$\Pi(X_{i,j} | I_j) = \bigvee_{x_j} (\min[\mu_{X_{i,j}}(x_j), \mu_{I_j}(x_j)]) \quad (5)$$

Equations (2), (3), and (5) describe the generalized modus ponens [22], which is the basis for the interpreter of a fuzzy-rule set.

C. Inference Control

The most basic design choice is the selection of the defuzzification mode. The output of the rule aggregation stage is a composite membership distribution defined on the space of control actions. This distribution must be summarized into a scalar value before it is passed to an actuator for execution. This summarization can be performed by a variety of defuzzifiers: the Mean of Maxima (MOM), the Center of Gravity (COG), the Height Method (HM). The selection of the defuzzifier is a tradeoff between storage requirements (MOM lends itself to easy compilation), performance (COG typically exhibits the smoothest performance), and computational time (HM is faster to compute than COG) [23].

C.1. Mean Of Maxima (MOM)

The MOM method defines the crisp output y^* as the value of y in which the membership distribution $\mu_Y(y)$ achieves its maximum. If the maximum is obtained in multiple points, y^* is the average of such set of points. Therefore we can define the interval of points \bar{y}^* where such maximum is achieved as

$$\bar{y}^* = \{\hat{y} \in Y \mid \mu_Y(\hat{y}) = \bigvee_y \mu_Y(y)\} \quad (6)$$

and then we define y^* as the average of \bar{y}^* .

¹ When the input is crisp, the degree of matching is the evaluation of the reference membership distribution at the point representing the value of the input:

$$\Pi(X_{i,j} | I_j) = \min[\mu_{X_{i,j}}(x_0), \mu_{I_j}(x_0)] = \mu_{X_{i,j}}(x_0) \quad (4)$$

C.2. Center Of Gravity (COG)

The COG method derives the crisp output y^* as

$$y^* = \frac{\int y \mu_Y(y) dy}{\int \mu_Y(y) dy} \quad (7)$$

C.3. Height Method (HM)

The HM method derives the crisp output y^* as

$$y^* = \frac{\sum_{i=1}^m \lambda_i \times c_i}{\sum_{i=1}^m \lambda_i} \quad (8)$$

where c_i is the center of gravity of $\mu_{Y_i}(y)$.

IV. THE APPLICATION TASKS

Following the rapid prototyping paradigm, the authors have identified five application tasks for a KBS (see reference [24]): (1) requirements and specifications: the knowledge acquisition; (2) design choices: the KB development stage; (3) testing and modification: the KB functional validation stage; (4) optimizing storage and response time requirements: the KB compilation; (5) running the application: the deployment stage.

The same task decomposition applies to the development of a FLC application. The first three stages correspond to the development of the FLC application (performance characteristics, order estimation, state variable identification, rule base generation, validation and robustness analysis); the fourth stage corresponds to the transition from development to deployment (fuzzy rule set compilation); the fifth stage corresponds to the application deployment (porting and embedding the FLC on the host computer). Because of space constraints we will focus the rest of this paper on the compilation stage, which is the necessary link to provide for the transition from development to deployment.

A. The Fuzzy KB Compilation Process

The compilation of fuzzy rule bases into fast access lookup tables is analogous to the compilation process used in programming languages and KBS.

Traditionally, the compilation process in programming languages refers to the translation of statements from a high-level source language into a low-level target language, such as assembly language or machine language, to allow for an efficient execution.

The compilation process in Knowledge Based Systems usually refers to the process of translating variables, predicates, and rules into a dependency graph. Such a graph is used to keep a pointer for all rules

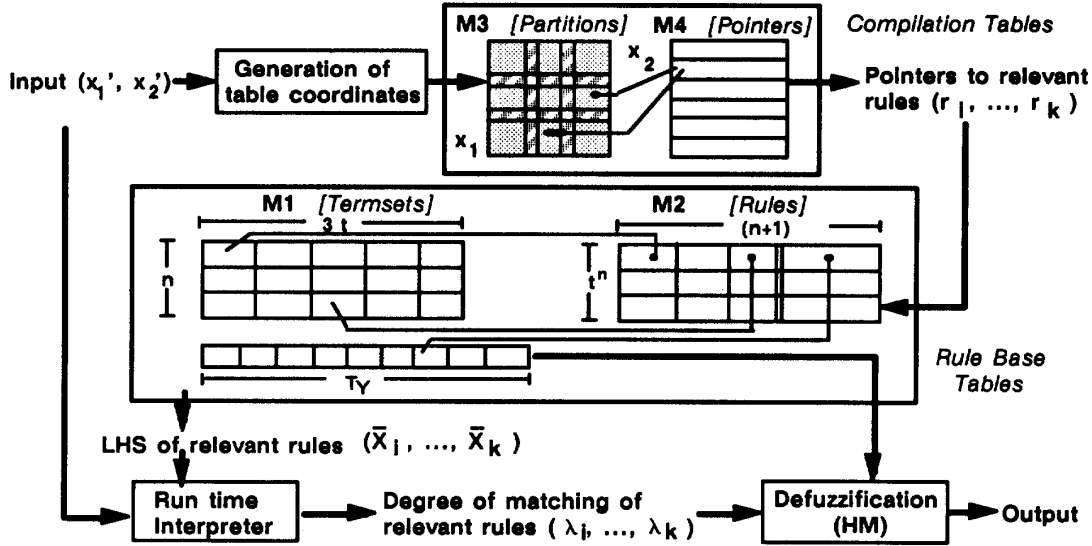


Figure 4: Architecture of a compiled FLC, functionally equivalent to that of an interpreted FLC.

Counter i ($i = 0 \dots n$)	Number of non-zero rule outputs $r = 2^i$	Number of cells in M3 pointing to slots containing r rules $\binom{n}{i} (t-1)^i t^{(n-i)}$	Number of cells in slots of M4 pointed to by corresponding cells in M3 $2^i \binom{n}{i} (t-1)^i t^{(n-i)}$
$i=0$	1	$t^n = 25$	$t^n = 25$
$i=1$	2	$n(t-1)t^{(n-1)} = 40$	$2n(t-1)t^{(n-1)} = 80$
$i=2$	4	$\binom{n}{2} (t-1)^2 t^{(n-2)} = 16$	$4 \binom{n}{2} (t-1)^2 t^{(n-2)} = 64$
TOTALS		$ M3 = (2t-1)^n = 81$	$ M4 = \sum_{i=0}^{n-1} 2^i \binom{n}{i} (t-1)^i t^{(n-i)} = 169$

Figure 5: Number of elements in M3 for $t = 5$ and $n = 2$.

containing the same variable, and for all variables affected by the same rule, thus eliminating the need of run-time search. The graph typically maintains the current evaluation of each rule, allowing for incremental rule evaluation when new information is entered [25, 26]. Therefore, an interpreter can be used during the KBS development phase, where changing requirements and functionalities may require changes in the source code or in the knowledge base. After a successful validation stage, the KB is considered stable and can then be compiled to minimize storage requirements, avoid exhaustive evaluation, and improve run-time performance.

In a similar fashion, a FLC application can be compiled after the validation stage. During the development phase we use the FLC interpreter to generate, fine-tune, and validate the fuzzy rule set (KB). Once

the validation is complete, we employ a FLC compiler to generate lookup tables from the fuzzy rule sets.

Figure 4 shows the architecture of a compiled FLC. Table M1 contains the termsets of the input and output variables. Table M2 lists the rules, with pointers to the termsets that make up the LHS of those rules, as well as pointers to the conclusions of those rules.

The main effort of the compiler is expended in the construction of tables M3 and M4. The state space is partitioned into a number of cells whose boundaries are defined by the termsets of the input variables. If the termsets are overlapping only with adjacent terms, and the number of terms for each of the n state variables is t , then the state space is composed of $(2t-1)^n$ distinct cells, as seen in the last row of Figure 5. In general, the expression for the number

PARAMETERS		INTERPRETED	COMPILED			
Terms	State	Rule Evaluation	Rule Evaluation		Storage	
t	n	Total number of Rules: t^n	Worst Case: (2^n)	Average Case: $ M4 / M3 $	Tot. number of cells (M3): $(2t-1)^n$	Tot. number of rule pointers (M4): $\sum_{i=0}^{i=n} 2^i \binom{n}{i} (t-1)^i t^{(n-i)}$
3	2	9	4	1.96	25	49
3	3	27	8	2.74	125	343
3	4	81	16	3.84	625	2,401
4	2	16	4	2.04	49	100
4	3	64	8	2.92	343	1,000
4	4	256	16	4.16	2,401	10,000
5	2	25	4	2.09	81	169
5	3	125	8	3.01	729	2,197
5	4	625	16	4.35	6,561	28,561

Figure 6: Rule evaluations and storage requirements using compilation.

of pointers to $r = 2^i$ rules is

$$\binom{n}{i} (t-1)^i t^{(n-i)}$$

where $i = 0 \dots n$. Each of these cells contains a pointer to a corresponding slot in table M4, which in turn contains a list of pointers to the rules that are applicable in that cell. The maximum length of this list of pointers is 2^n .

In the case of the fuzzy PI, where $n = 2$, M1 contains the termsets for e , \dot{e} , and du , while the entire rule set is listed in M2, with the appropriate pointers to the relevant termsets in M1. M3 details a partitioning of the $e-\dot{e}$ plane into a two-dimensional array of size $(2t-1)^2$, with each cell in the array pointing to a slot in M4. Of the $(2t-1)^2$ slots in M4, t^2 contain a pointer to a single rule, $2(t-1)t$ contain pointers to two rules, and $(t-1)^2$ contain pointers to four rules.

At run-time, the values of the input variables and the termsets associated with those variables in M1 are used to determine into which region of state space the input falls. Once that region is determined, M3 is consulted, and a list of pointers to the relevant rules in M2 is obtained from M4. The rest of the inferencing process is similar to that of an interpreted FLC. However, the run-time process has been made more efficient by avoiding the evaluation of those rules whose contribution is zero. Thus, expression (8) becomes

$$y^* = \frac{\sum_{i=1}^m (\lambda_i \times c_i)}{\sum_{i=1}^m \lambda_i} = \frac{\sum_{i|\lambda_i \neq 0} (\lambda_i \times c_i)}{\sum_{i|\lambda_i \neq 0} \lambda_i} \quad (9)$$

where m , the maximum number of rules evaluated in interpreted mode, is a function of the number of state variables n , and the cardinality t of each state

variable's termset, namely $m \leq t^n$. After compilation, the *worst-case* maximum number of rule evaluations is $|\lambda_i \neq 0| \leq 2^n$, but the compiled FLC is still functionally equivalent to the interpreted FLC. However, the run-time interpreter must still calculate the degrees of matching for the firing rules, perform conclusion detachment, and defuzzify the resulting output.

Figure 6 shows the decrease of run-time rule evaluations obtained by the compilation process. In the worst case, rule execution will be reduced from t^n to 2^n . In the average case, the reduction will be from t^n to about 2-4 rules. The average case was computed as the ratio $\frac{|M4|}{|M3|}$, assuming uniform distributions for the input values and equal partitions size in the state space.² The last two columns of the same table illustrate the memory requirements for M3 and M4, i.e., the storage price to be paid for the run-time gains.

Alternatively, to increase the throughput of the controller at the expense of accuracy, run-time rule evaluation can be completely avoided by performing that evaluation during compilation. For each region in the state space partition, a representative point is chosen, and each of the firing rules is evaluated at that point. The resulting output value is stored in M3, instead of storing pointers to lists of rules. At run-time, as shown in Figure 7, the input values are used to determine the region of state space in which the input falls, and the output value stored there is returned.

²The cells into which the state space has been subdivided have equal size (area, volume, or hypervolume) if the support of any two adjacent inner terms in each termset has a 33% overlapping and the support of any adjacent inner term with any extreme term of every termset has a 50% overlapping. Any smaller overlapping will reduce the size of partitions containing multiple rules and further decrease the average number of run-time rule evaluations.

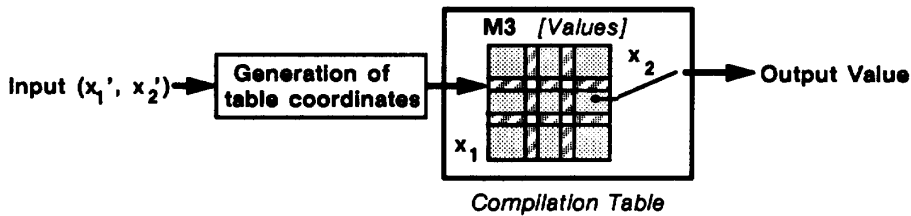


Figure 7: Architecture of an approximated FLC.

This approximation results in a non-uniform sampling of the state space. In the t^n regions where only one rule can fire, the output value is equivalent to the interpreted FLC. The control surface of the approximated FLC differs the greatest in the $(2t-1)^n - t^n$ regions of overlap, where two or more rules are applicable. In the case of the fuzzy PI, the horizontal areas of the control surface remain unchanged, while the sloping areas are replaced with horizontal ones, whose heights are intermediate to those of the boundaries of the sloping areas.

Further details on the compilation process are available in [7], where we show that compilation can reduce run-time rule evaluations by two orders of magnitude in the *average* case. In the same reference we analyze the memory requirements to show that the compilation can be implemented using the small microprocessors (e.g. 4-8 bit) and limited memory (e.g. 4K-8K bytes) typical of cost-sensitive applications such as appliances and consumer electronics.

V. CONCLUSIONS

We have described fuzzy logic controllers as nonlinear control surfaces that generalize the concept of sliding mode controllers. The control surface of a FLC is defined in a *high level language* by a Knowledge Base composed of a rule set and termsets. The KB can be *interpreted* during development for validation and refinement. Prior to deployment the KB can be *compiled* into lookup tables and used directly (as an approximation of the interpreted version) or with an efficient run-time engine (as a functionally equivalent version of the interpreted version).

REFERENCES

- [1] M. Sugeno, editor. *Industrial Applications of Fuzzy Control*. North Holland, Amsterdam, 1985.
- [2] T. Shingu and E. Nishimori. Fuzzy Based Automatic Focusing System for Compact Camera. In *Proceedings of the Third International Fuzzy Systems Association*, pages 436-439. IFSA, August 1989.
- [3] S. Yasunobu and S. Miyamoto. Automatic train operation by predictive fuzzy control. In M. Sugeno, editor, *Industrial Applications of Fuzzy Control*, pages 1-8. North Holland, Amsterdam, 1985.
- [4] S. Yasunobu and G. Hasegawa. Evaluation of an automatic crane operation system based on predictive fuzzy control. *Control Theory and Advanced Technology*, 2:419-432, 1986.
- [5] H. Takahashi, K. Ikeura, and T. Yamamori. 5-Speed Automatic Transmission Installed Fuzzy Reasoning. In *Proceedings of the International Fuzzy Engineering Symposium '91 (IFES'91)*, pages 1136-1137, November 1991.
- [6] T.J. Procyk and E.H. Mamdani. A Linguistic Self-Organizing Process Controller. *Automatica*, 15(1):15-30, 1979.
- [7] P. P. Bonissone. A Compiler for Fuzzy Logic Controllers. In *Proceedings of the International Fuzzy Engineering Symposium '91 (IFES'91)*, pages 706-717, November 1991.
- [8] Li Zheng. A Practical Guide to Tune Proportional and Integral (PI) Like Fuzzy Controllers. In *Proceedings of the IEEE International Conference on Fuzzy Systems 1992*, pages 633-640. IEEE, March 1991.
- [9] K. L. Tang and R. J. Mulholland. Comparing Fuzzy Logic with Classical Controller Design. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(6):1085-1087, 1987.
- [10] J. E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, Englewood Cliffs, 1991.
- [11] R. Palmer. Sliding Mode Fuzzy Control. In *Proceedings of the IEEE International Conference on Fuzzy Systems 1992*, pages 519-526. IEEE, March 1991.
- [12] S. M. Smith and D. J. Comer. An Algorithm for Automated Fuzzy Logic Controller Tuning.

- In *Proceedings of the IEEE International Conference on Fuzzy Systems 1992*, pages 615–621. IEEE, March 1991.
- [13] P. P. Bonissone, S. Gans, and K. S. Decker. RUM: A layered architecture for reasoning with uncertainty. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 891–898. AAAI, August 1987.
- [14] P. P. Bonissone and K. S. Decker. Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity. In L. Kanal and J. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 217–247. North-Holland, Amsterdam, 1986.
- [15] D. Burkhardt and P. Bonissone. Automated Fuzzy Knowledge Base Generation and Tuning. In *Proceedings of the IEEE International Conference on Fuzzy Systems 1992*, pages 179–188. IEEE, March 1991.
- [16] E.H. Mamdani and S. Assilian. An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *Int. J. Man Machine Studies*, 7(1):1–13, 1975.
- [17] M. Mizumoto and H. Zimmerman. Comparison of Various Fuzzy Reasoning Methods. *Fuzzy Sets and Systems*, 8:253–283, 1982.
- [18] C.C. Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I, II. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):404–435, 1990.
- [19] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.
- [20] B. Schweizer and A. Sklar. *Probabilistic Metric Spaces*. North Holland, New York, 1983.
- [21] P. P. Bonissone. Summarizing and propagating uncertain information with triangular norms. *International Journal of Approximate Reasoning*, 1(1):71–101, January 1987.
- [22] L. A. Zadeh. A theory of approximate reasoning. In P. Hayes, D. Michie, and L. I. Mikulich, editors, *Machine Intelligence*, pages 149–194. Halstead Press, New York, 1979.
- [23] M. Mizumoto. Improvements Methods of Fuzzy Controls. In *Proceedings of the Third International Fuzzy Systems Association*, pages 60–62. IFSA, August 1989.
- [24] P. P. Bonissone. Now that i have a good theory of uncertainty, what else do i need? In M. Henrion, R. D. Shachter, L. Kanal, and J. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 237–253. North-Holland, Amsterdam, 1990.
- [25] Charles L. Forgy. Rete: A fast Algorithm for the many pattern/many object pattern match problem. *Journal of Artificial Intelligence*, 19(1):17–37, September 1982.
- [26] Piero P. Bonissone and Pete C. Halverson. Time-Constrained Reasoning Under Uncertainty. *Journal of Real Time Systems*, 2:22–45, May 1990.