Implementation of Fuzzy Logic

Selected Applications

SPRA028
January 1993

PRINTED WITH SOY INK™

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments Incorporated (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

These three reports describe selected applications that the Texas Instruments Digital Signal Processing (DSP) Group, in conjunction with TI research groups, has developed in the emerging field of fuzzy logic.

The first paper gives an overview of the fuzzy logic theory and suggests methods of implementation that are illustrated in the next two papers.

The second paper describes the implementation of fuzzy logic as a serial algorithm on a TMS320 DSP system. The algorithm is based on parts of the theory described in the third paper.

The third paper presents a theory for a dedicated silicon-level fuzzy logic implementation.

Contents

What is Fuzzy Logic? An Overview of the Latest Control Methodology **Page**

Timothy A. Adcock

Implementation of Fuzzy Logic Servo Motor Control on a Programmable TI TMS320C14 DSP **Page**

Mathew George, Jr.

The Programmable Fuzzy Logic Array **Page**

Philip Thrift

What is Fuzzy Logic?
An Overview of the
Latest Control Methodology

Timothy A. Adcock
Digital Signal Processing — Semiconductor Group
Texas Instruments Incorporated

Introduction

The name “Fuzzy Logic” seems to imply an imprecise methodology that is useful only when accuracy is not necessary or important. That is what many people assume when they first hear about fuzzy logic—and understandably so. In a world increasingly manipulated by computers with their absolute “1” or “0” and “on” or “off” concepts, a term like fuzzy logic suggests inaccuracy or imprecision. Even Webster’s dictionary defines “fuzzy” as:

fuzz·y (–ē) adj. 2. not clear, distinct, or precise; blurred

This is not true of fuzzy logic. Fuzzy logic can address complex control problems, such as robotic arm movement, chemical or manufacturing process control, antiskid braking systems, or automobile transmission control with more precision and accuracy, in many cases, than traditional control techniques have.

Fuzzy logic was invented and named by Lotfi Zadeh, a professor at the University of California at Berkeley. Fuzzy logic is a methodology for expressing operational laws of a system in linguistic terms instead of mathematical equations. Many systems are too complex to model accurately, even with complex mathematical equations, but fuzzy logic’s linguistic terms provide a useful method for defining the operational characteristics of such a system. These linguistic terms are most often expressed in the form of logical implications, such as If – Then rules:

If air_temp is WARM, then set fan_speed to MEDIUM.

The terms WARM and MEDIUM are actually sets that define ranges of values known as membership functions. By choosing a range of values instead of a single discrete value to define the input variable “air_temp”, you can control the output variable “fan_speed” more precisely. Fuzzy logic controllers can often improve the performance of a control system by reducing the chance of wild functions in the output that may be caused by variations in the measured input variables.

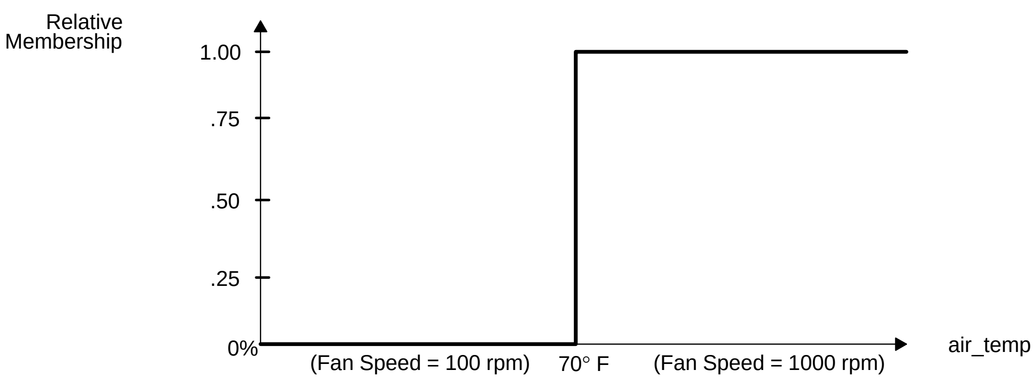
The Traditional Approach

To illustrate the difference between fuzzy logic and the traditional approach, here is a control problem. First, consider how the traditional—often called “crisp”—controller would handle it:

If air_temp is ≥ 70° Fahrenheit, then set fan_speed to “1000 rpm”.
If air_temp is < 70° Fahrenheit, then set fan_speed to “100 rpm”.

A nonfuzzy, or “crisp,” controller relies on a discrete valued decision point. For this type of system, the input must reach an exact value before the control system reacts in a certain way. Even small variances in this input value may cause the output to react drastically differently. For instance, if the temperature is 70° or above, the first rule will set the fan_speed to 1000 rpm. If the temperature is below 70°, the second rule will set the fan_speed much lower, to 100 rpm. Figure 1 shows a diagram of this crisp valued controller.

Figure 1. Crisp Controller



What would happen if the temperature were 69.5°? Or, more importantly, what would happen to the control system if the temperature were transitioning from below 70° to above 70°? The temperature might even fluctuate back and forth slightly above or below 70° (e.g., 69.0° to 71.0°). This would cause the control system to alter the fan speed wildly for changes in the input variable air_temp, although the temperature change may not be significant.

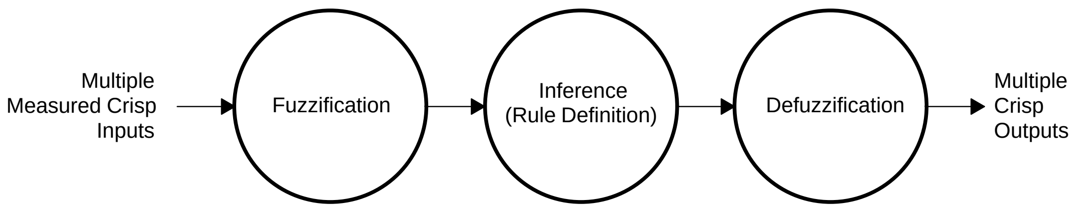
These transition points are difficult for “crisp” control systems to handle, but they are exactly where “fuzzy logic” excels.

Fuzzy Control

Fuzzy logic is implemented in three phases (see Figure 2):

1. Fuzzification (crisp input to fuzzy set mapping).
2. Inference (fuzzy rule generation).
3. Defuzzification (fuzzy to crisp output transformation).

Figure 2. Fuzzy Logic Phases

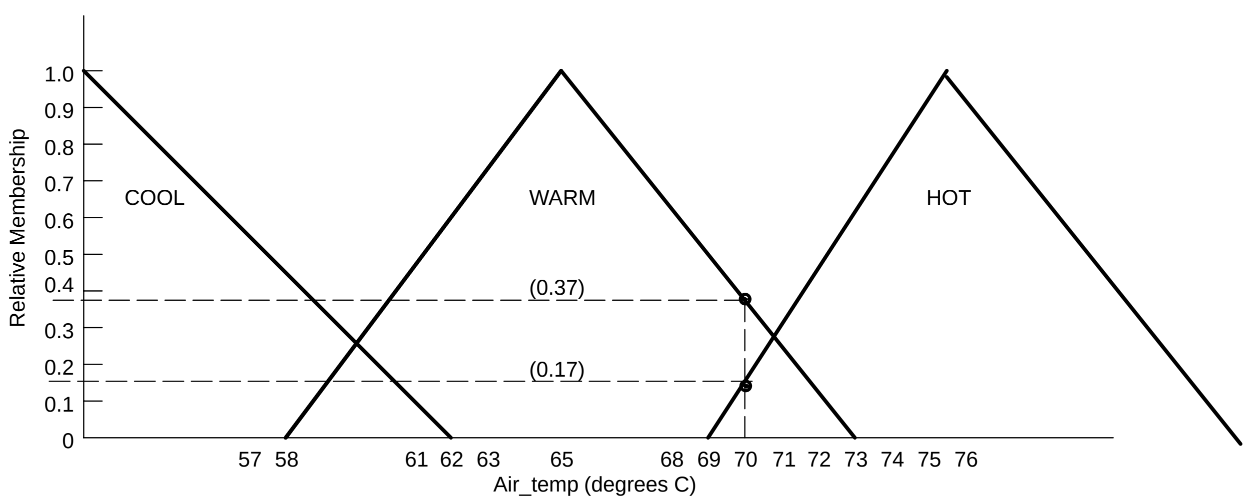


Fuzzification

In the first fuzzy logic phase—fuzzification—actual measured input values are mapped into fuzzy membership functions. As an example, a climate-control system has been developed with fuzzy logic.

To create a climate control system, we first developed membership functions for the input variable “air_temp”. These membership functions are defined by both a range of values and a degree of membership. In fuzzy logic, it is important to distinguish not only which membership functions a variable belongs to, but also the relative degree to which it is a member. This gives the variable a “weighted” membership in a membership function. A variable can have a weighted membership in several membership functions at the same time. The membership functions for “air_temp” are shown in Figure 3.

Figure 3. Air_Temp Input Variable Membership Functions



3

As shown in Figure 3, fuzzy membership functions span a range of values and can actually overlap. Three sets of membership values are defined above for the variable "air_temp". They are COOL, WARM, and HOT. The degree of membership is found by finding the intersection point of a distinct input value on the horizontal axis with the line defining one or more fuzzy membership functions. This intersection point is assigned a corresponding value on the vertical axis to define the relative membership in a set for an actual measured input value. Notice that when "air_temp" is at a particular value, it may be contained in one or more fuzzy sets. For instance, at 70°, "air_temp" is a member of the function HOT with a relative membership of 0.17. It is also a member of the function WARM with a relative membership of 0.37. Unlike a crisp system in which a value either is or is not a member of a function, a fuzzy logic system can take action based not only on membership in a fuzzy set, but also on the degree to which a variable is included in a membership function. In this case, because "air_temp" at 70° is more WARM (0.37) than it is HOT (0.17), the controller will take that into account when defining what output action to take.

Inference Rule Definition

Once membership functions have been defined for input and output variables, a control rule base can be developed to relate the output actions of the controller to the observed inputs. This phase is known as the inference, or rule definition portion, of fuzzy logic. Any number of rules can be created to define the actions of the fuzzy controller. Some examples are shown below.

Fuzzy Logic Rule Definition

If air_temp is COOL, then set fan_speed to SLOW.
If air_temp is HOT, then set fan_speed to FAST.
If air_temp is WARM, then set fan_speed to MEDIUM.

These If-Then rules can relate multiple input and output variables. Because the rules are based on word descriptions instead of mathematical definitions, any relationship that can be described with linguistic terms can typically be defined by a fuzzy logic controller. This means that even nonlinear systems can be described and easily controlled with a fuzzy logic controller. In addition, since variables have weighted memberships—in particular membership functions—the rules that are composed of these variables are weighted as well. This means that different rules have different impacts on the controller, according to the measured input variable. For a multiple-input/multiple-output system with many defining rules, a wild fluctuation in any single input will be tempered by these rule weightings. Because of this, fuzzy logic systems are very robust and often allow many rules to be removed or altered without significantly impacting the controller.

Defuzzification

After the fuzzy logic controller evaluates inputs and applies them to the rule base, it must generate a usable output to the system it is controlling. This may mean setting a voltage or current to a particular value to control the speed of a fan in the example above, or it may mean defining the optimal speed of a robotic arm as it nears its target. The fuzzy logic controller must convert its internal fuzzy output variables into crisp values that can actually be used by the controlled system. You can perform this portion of the fuzzy control algorithm, known as defuzzification, in several ways. Two of the most common methods are:

- maximum defuzzification method (page 5).
- centroid calculation defuzzification method (page 6).

Remember from fuzzification that in mapping input variables to membership functions, a particular measured value of the input variable determined the relative membership of that input variable in an input membership function. To determine the mapping of output variables to their corresponding output membership functions, the weighted input membership function and corresponding rule base determine the relative membership in the output function. Whatever relative membership was given to the input variable will also be given to the output variable, as assigned by its corresponding rule. For air_temp = 70°, the output variables are assigned a value that corresponds to the input value shown in Table 1.

Table 1. Fan_Speed (Membership Function Relative Membership)

Input Variable	Defining Rules	Output Variable
air_temp (WARM) = 0.37	If air_temp = WARM, then set fan_speed to MEDIUM	fan_speed (MED) = 0.37
air_temp (HOT) = 0.17	If air_temp = HOT, then set fan_speed to FAST	fan_speed (FAST) = 0.17

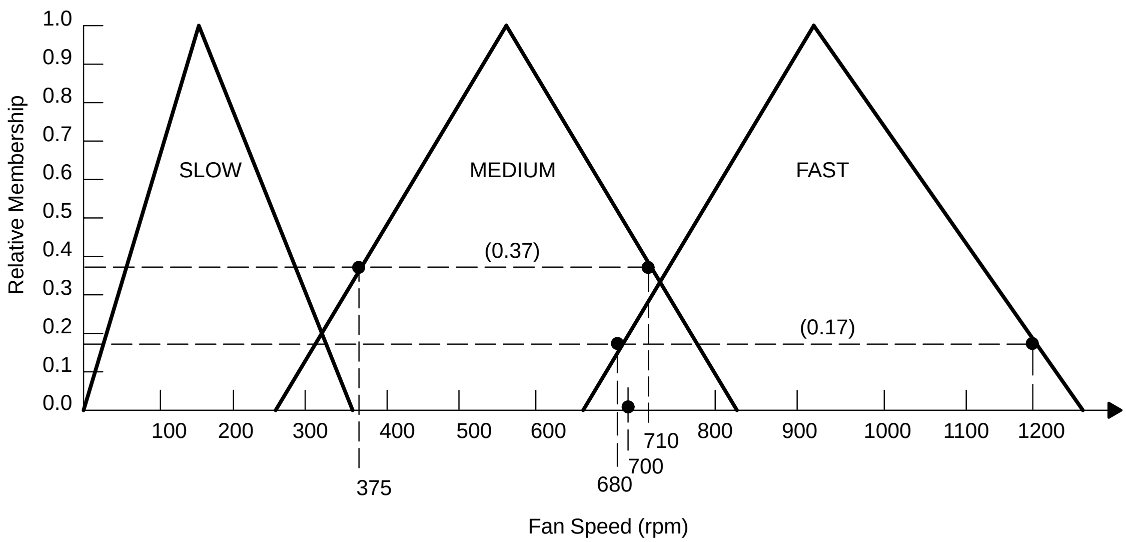
The output variable fan_speed is given the same relative mapping as the input variable air_temp that is defined by a particular rule.

Figure 4 illustrates the output variable membership functions. It this case, a distinct value on the horizontal axis is defined by the relative membership on the vertical axis. To create the actual crisp output value for the controller system output, membership functions are used with the output variable. In Table 1, the input value air_temp = 70 resulted in two weightings for fan_speed:

- Fan_speed = 0.37 was assigned to the output membership function MEDIUM.
- Fan_speed = 0.17 was assigned to the output membership function FAST.

As shown in Figure 4, the actual output value is determined by beginning at the weighting factor on the vertical axis and moving horizontally until an intersection point is reached on the lines defining its associated membership function. This intersection point is then transposed to the horizontal axis to determine the crisp output value.

Figure 4. Fan_Speed Output Variable Membership Functions



Maximum Defuzzification Method

One method of defuzzification is known as the maximum method. In this method, if more than one rule is active, the maximum relative membership is used to determine the output value. In the above example, making air_temp = 70° created two possible values for fan_speed:

fan_speed = 0.37
fan_speed = 0.17

The maximum defuzzification method provides a single output by choosing the active rule with the greatest relative membership value in the output membership function. In the preceding example, the following rule is chosen because it has the highest membership value for fan_speed.

If air_temp = WARM, then set fan_speed to MEDIUM. Fan_speed (MED) = 0.37

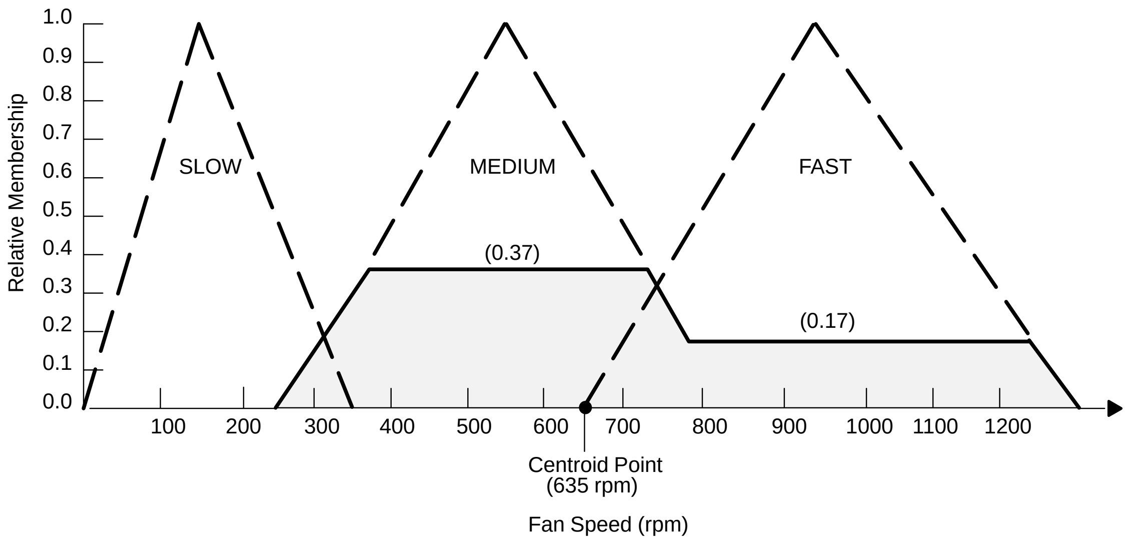
The value 0.37 on the vertical axis intersects the membership function MEDIUM at two points—one on the positive slope (at 375 rpm) and one on the negative slope (at 710 rpm) of the function. The two points represent two possible solutions that must be resolved.

5

Centroid Calculation Defuzzification Method

Another method for calculating the output value is the centroid method. In this method, a weighted average of all the active rules determines an output by summing all of the applicable output variables over their relative membership values. Although this method is more computationally intensive, it creates a distinct output value based on the relative memberships of all of the active rules that apply (see Figure 5). This method eliminates the problem of multiple solutions observed with the maximum method. A processor architecture with a hardware multiply-accumulate feature like that of the TMS320 DSP family excels at this method.

Figure 5. Fan_Speed Output Centroid Calculation



Summary

By using fuzzy logic, you can simplify complex control problems that once required a high-powered microprocessor to execute in real time; you can now execute them on a low-cost Texas Instruments TMS320 DSP or TMS370 microprocessor. The following application note shows the benefits of controlling a simple DC motor with fuzzy logic using a TMS320C14 digital signal processor.

Implementation of Fuzzy Logic Servo Motor Control on a Programmable Texas Instruments TMS320C14 DSP

Mathew George, Jr.
Digital Signal Processing — Semiconductor Group
Texas Instruments Incorporated

Abstract

This paper describes the implementation of a fuzzy logic compensator on a Texas Instruments TMS320C14 DSP-based servo motor control development system. The system contains a real motor that is controlled by the programmable DSP. An on-chip debugger and servo motor program allowed both simple code modification and interactive control of the motor. A fuzzy logic algorithm was directly substituted for the original PID algorithm; this resulted in comparable motor response and algorithm performance. This implementation proves the feasibility of real-time fuzzy logic-based servo motor control on a real system.

Introduction

Fuzzy logic is relatively new theory. Most of the readily available hands-on fuzzy logic system examples have been software simulations or bulky real systems. However, a TI commercial microprocessor (the TMS320C14) can serve as a simple, real-time, real-system platform for applying and investigating fuzzy logic. This facilitates both the understanding and implementation of fuzzy logic as a real-time programmable solution for the general engineering public.

Servo motor control is a viable and useful implementation. A programmable PID motor control board uses a Texas Instruments TMS320C14 chip and has an actual motor whose performance can be observed. Faster and newer parts are available, but the ’C14 is optimized for motor control with such features as on-board pulse-width modulation (PWM) generation capabilities. The board also has on-chip debugger code and interactive PID control code. The PID compensator code (written in TMS320 assembly language, which is upward-compatible with code executed by such newer fixed-point TI DSPs as the TMS320C25 or TMS320C50) uses position and velocity of the motor for inputs and motor input current as the output. The control code will allow the operator to interactively change such values as servo position and velocity and to monitor position error. Modifying the code for fuzzy logic required replacing the PID compensator section of the code with a fuzzy logic compensator.

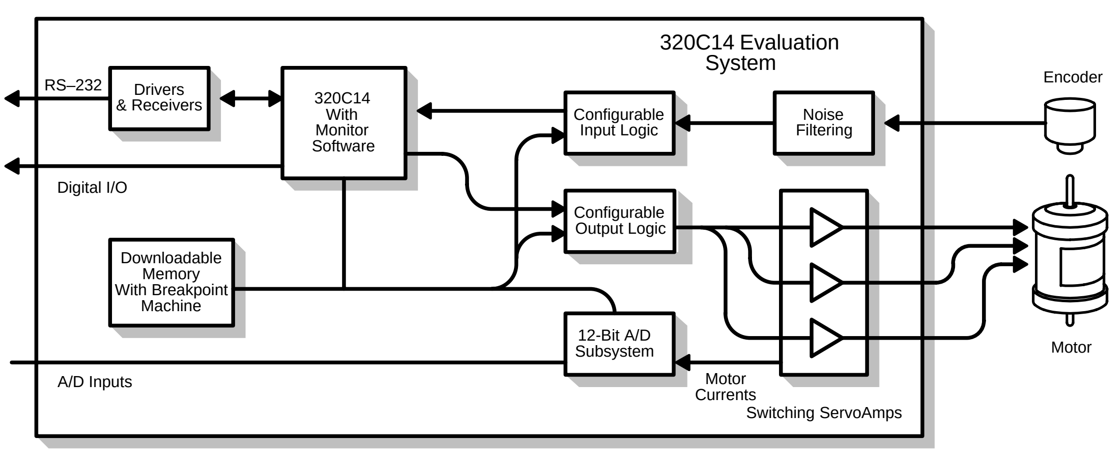
The membership function for the compensator defines the error between present motor position and the desired (command) position of the controller. Five linguistics variables characterize the function: negative medium, negative small, zero, positive small, and positive medium. The function is represented as overlapping isosceles right triangles for ease of fuzzification. Various rules for an inverted pendulum control system (ball and stick) are explained in [1]. These same eleven rules were used for servo motor control.

The algorithm’s three sections are implemented as a series of software loops: fuzzification (i.e., input evaluation), fuzzy inference (rule contribution that uses a table look-up), and defuzzification (using center-of-gravity method). It is based on an algorithm developed for [4]. Each section uses various arrays that are modified in that section. The sizes of these loops are directly proportional to the number of inputs, outputs, and rules used in the system.

Servo Motor System (Power-14)

The heart of the Power-14 board is a Texas Instruments TMS320P14 chip (one-time programmable TMS320C14). (See Figure 1).

Figure 1. Power 14 System



You communicate to the board through an RS-232 serial port connection by using standard terminal or terminal emulation software (such as Procomm). The 'P14 peripherals are optimized for control applications. An event manager can be operated in a PWM (pulse-width modulation) mode that is ideal for motor control. Monitor code is loaded into 'P14 external memory and run, which provides a command line debugger. The debugger has all standard debugger functions, such as memory dumps and modification, stepping through code, breakpoints, etc. The monitor can be used to load and run the servo motor program with PID compensator. The program is interactive and lets you control the motor from the keyboard. Position, velocity, and the PID values can be set. Data acquisition functions allow an ASCII text input stimulus table to be loaded onto the Power-14, an acquisition run to be executed, and the resulting ASCII output table to be written to a PC file for graphing. The rest of the board contains support circuitry: amplifiers for the motor and a serial port interface. An encoder on the motor is used as a position sensor for a compensator input. The velocity is found by executing a back-difference. Note that there is no separate velocity sensor.

PID Implementation

The source code for this system implements the PID compensator in one file (See Appendix A). The algorithm is a direct implementation of the PID equation. The Proportional, Integral, and Differential variables are derived from the motor encoder sensor detecting position. On each cycle, the present position is taken from the encoder and stored in Position. The error is found by subtracting Position from DesiredPosition and storing it in ErrNow. Thus, ErrNow is the position input for the proportional section of the PID. A back-difference is then taken with ErrNow and ErrLast (the error in the previous cycle) to give ErrDiff. ErrDiff is an approximation of the velocity and is therefore the second input (Differential) of the PID. The Integral is found by adding the ErrNow value and storing it in Kintegrator. The following equation then holds the compensator output:

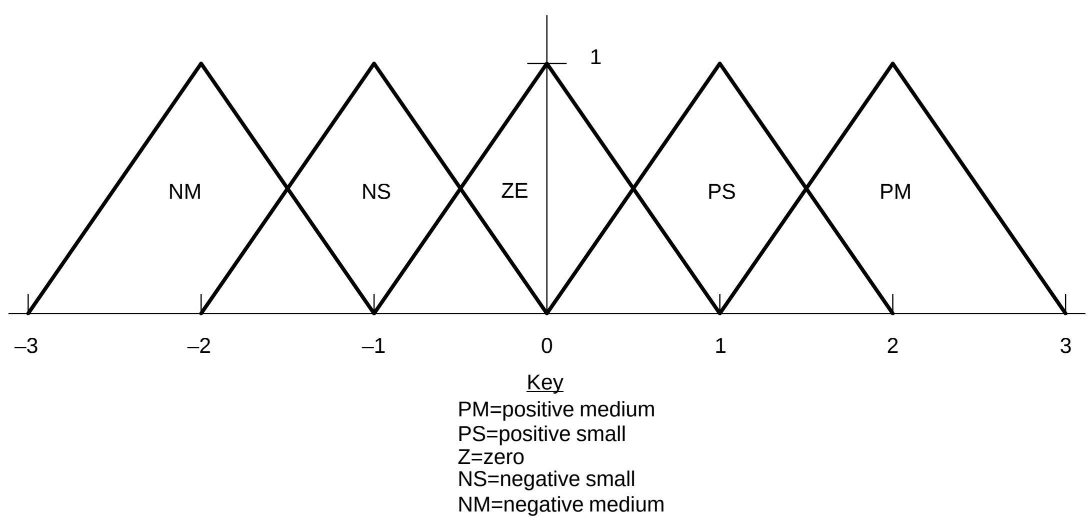
$$PWM \;=\; 9 * P * ErrNow \;+\; 9 * I * Kintegrator \;+\; 9 * D * ErrDiff \qquad \textbf{(1)}$$

The value is scaled for the PWM mode and stored in NewServo. Thus, for the compensator output, the actual motor current input value is converted to a PWM value for implementation. The PWM output is then sent to power amplifiers that finally drive the motor. The PWM frequency can be controlled from the PID program.

Fuzzy Logic Theory for Servo Motor Control

The membership function for this system is simple. Five fuzzy logic ranges (linguistic variables) were chosen to remain consistent with [1] and used for both inputs. Sets of values are represented as overlapping isosceles right triangles (Figure 2).

Figure 2. Membership Function



Two input variables, position (Theta) and velocity (dTheta) of the motor, are used in this fuzzy logic system. One output variable, motor current, is used, which will be proportional to the PWM output that is actually written. The rules for the compensator as mentioned were taken from [1]. Thus Theta, dTheta, and motor current operate according to the following "if a and b, then c" rules, as shown in Table 1.

Table 1. List of Rules

If Theta =	And dTheta =	Then Motor Current =
Z	Z	Z
PS	Z	NS
PM	2	NM
NS	Z	PS
NM	Z	PM
Z	NS	PS
Z	NM	PM
Z	PS	NS
2	PM	NM
PS	NS	Z
NS	PS	Z

These rules can then be indexed according to the scale shown in Figure 2. The mapping seen in Table 2 will be used in TMS320 programming.

Table 2. Indexed List of Rules

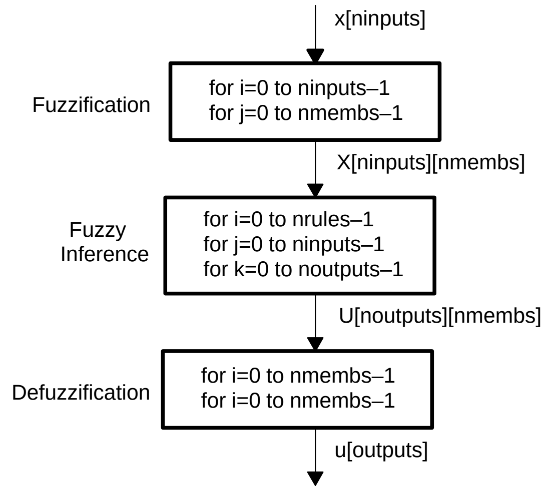
If Theta =	And dTheta =	Then Motor Current =
0	0	0
1	0	–1
2	0	–2
–1	0	1
–2	0	2
0	–1	1
0	–2	2
0	1	–1
0	1	–2
1	–1	0
–1	1	0

The defuzzification is done by the center-of-gravity method.

Fuzzy Logic Implementation

Arrays are used in the fuzzy logic calculations and modified in the various loops that implement the compensator. Appendix B lists the TMS320C14 code. Figure 3 shows the three sections of the compensator: fuzzification, fuzzy inference, and defuzzification. Notation for the arrays follows C language standard, with subscripts from 0 to n–1. The ’C14 algorithm is based on an algorithm developed for [4]. The figure key summarizes the values of the system that will be used in the examples in this section.

Figure 3. List of Arrays



Fuzzy Logic Implementation as a Series of Loops

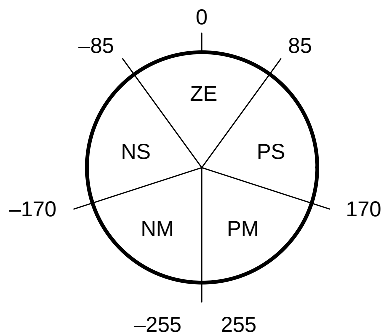
Key
nmembs=5
ninputs=2
nrules=11
noutputs=1
i, j, k = loops within loops

Note which arrays are modified in each section and their size boundaries. They are discussed later in more detail. The two inputs seen in x[ninputs] are position (found from the encoder) and velocity (found from an approximation of the derivative by taking the back-difference of the position). These are the Theta and dTheta variables described previously. As the compensator code begins, the position and velocity inputs (ErrNow and ErrDiff, respectively) are copied to the array X[ninputs]. The position is mapped so that one rotation of the motor ranges from –255 to +255 (See Figure 4 a). This relationship is mapped onto the x-axis of the membership function and thus fuzzifies the position of the motor (Figure 4 b). Note that this figure is not drawn to scale.

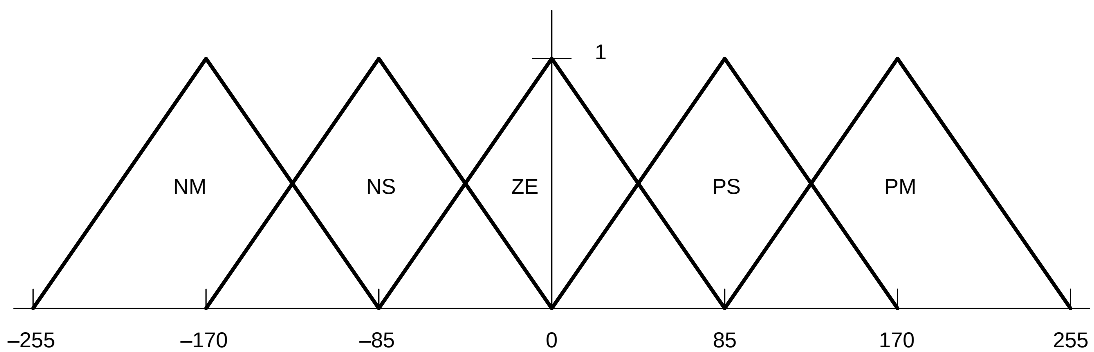
13

Figure 4. Motor Shaft/Membership Function Mapping

a. Motor Range

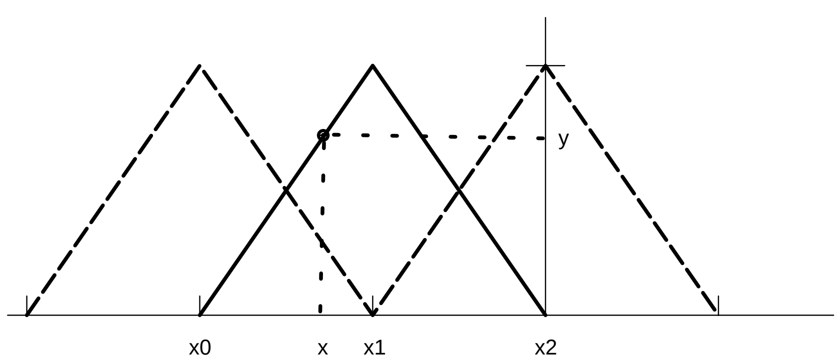


b. X-Axis Map



In the fuzzification loop, the degree of membership of each input relative to the input membership function is evaluated and written in the array X[ninputs][nmembs]. The value for a particular linguistic variable is the *y* value of the triangle for a particular value of *x* (See Figure 5).

Figure 5. Analysis for One Linguistic Variable



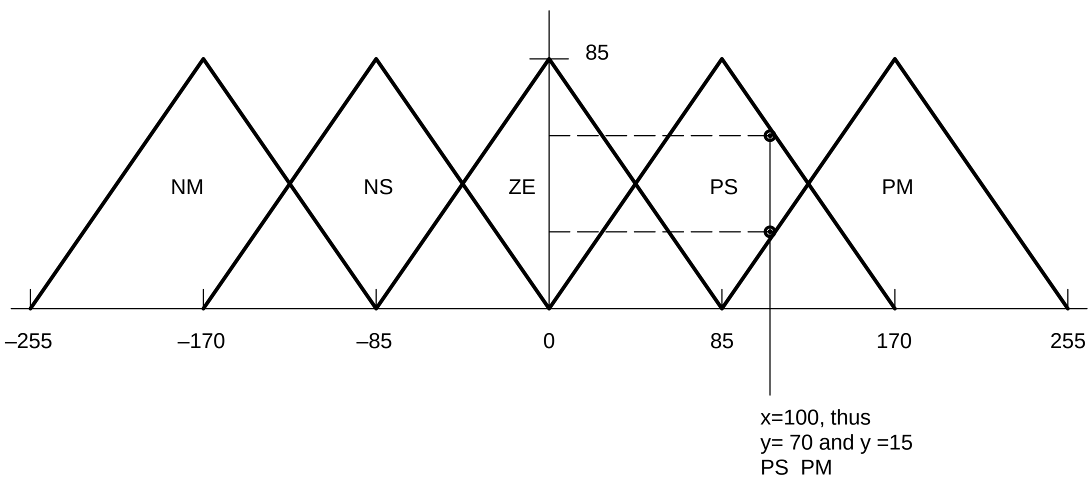
The *y* value is found by using the simple algebraic equation for a line ($y=mx+b$). This equation may be geometrically reduced to one of the two following equations, depending on which side of the triangle the value of *x* lies:

$$\begin{array}{l} if\ \{x : x0 < x < x1\}\ then\ y = (x - x0) / (x1 - x0)\ else \\ if\ \{x : x1 < x < x2\}\ then\ y = (x2 - x) / (x2 - x1) \end{array} \quad \textbf{(2)}$$

The division needed is costly on most microprocessors, usually requiring at least a number of cycles equivalent to the number of bits of the number being divided. But if the slope *m* is made equal to 1 by causing the elements of the membership functions to be isosceles right triangles, the equation can be reduced to $y=x-x0$. This translates into a simple one-cycle subtraction. Note that forcing the elements of the membership functions to be isosceles right triangles also forces the peak of the membership function to no longer be 1. Rather the peak value=$x1-x0=x2-x1$=85 (as seen in Example 1). This action also eliminates the need for using a *Q* format [6] to represent the fractional values from Equation 2 if the triangle were not isosceles.

Example 1 demonstrates this fuzzification calculation. If the position input value were 100 (i.e., x[0]=100), it would have nonzero degrees of membership in the PS and PM linguistic variables. You can also see this in Figure 5. To calculate the actual degree of membership value, the value for *x* is plugged into Equation 2 for both PS and PM boundaries. Thus, in PS the contribution is 70, while in PM it is 15. The rest of the linguistic variables are 0 because there is no contribution. (Of course, in software, all linguistic variables must be evaluated.) For this example, X[0][nmembs]=[0, 0, 0, 70, 15].

Example 1. Fuzzification Example



The next step involves fuzzy inference. In this loop, the maximum and minimum functions, as explained in [2], are implemented. In the actual code, the array indexing of the membership values is made nonnegative by adding a bias of three. Therefore, instead of NM to PM being indexed from –2 to +2, as seen in Figure 4, they are indexed from 0 to 4. Table 3 shows how the 11 rules are indexed and reindexed in the array RULE_TABLE[rule][input+output].

Table 3. Original and Reindexed Table of Rules

Original Index			Reindexed		
x(0)	*x(1)*	*u(0)*	*x(0)*	*x(1)*	*u(0)*
0	0	0	2	2	2
1	0	–1	3	2	1
2	0	–2	4	2	0
–1	0	1	1	2	3
–2	0	2	0	2	4
0	–1	1	2	1	3
0	–2	2	2	0	4
0	1	–1	2	3	1
0	1	–2	2	4	0
1	–1	0	3	1	2
–1	1	0	1	3	2

Some explanation is required for this decoding. The indexed rules match the explicit rules as described in Table 1. Also, the values given by accessing the RULE_TABLE are limited to the values indexed by nmembs. This characteristic is heavily used in the index manipulation and allows nmemb to be interchanged with RULE_TABLE[rule][input+output] in the appropriate parts of the algorithm.

To find the minimum value of X[ninputs][nmembs] decoded from the rule table inputs and stored in minZ (which is initialized to the maximum y value—in this case, 85), the equation is:

$$minZ = \min\ (X[input]\ [RULE\_TABLE\ [rule]\ [input]]) \quad \textbf{(3)}$$

(Since noutputs=1, the loop is simplified, and minZ does not need to be the general case array minZ[nrules]). Note that only the first two columns (the input columns) of RULE_TABLE are used in this part of the fuzzy inference section.

Then, for each rule, the max is taken of the output value U[nmembs], which is initialized to 0. The general case U[output][nmembs] is simplified because only one output is decoded from the rule table outputs and minZ. This equation can be summarized as:

$$\begin{aligned} U[nmembs] &= U(RULE\_TABLE[rule]\ [ninput + output]) \\ &= max\ (U[RULE\_TABLE[rule]\ [ninput + output]],\ minZ) \end{aligned} \quad \textbf{(4)}$$

Note that in this section only the last column (the output column) is used. The following equation summarizes the minimum and maximum functions that are executed for each rule to result in the array U[nmembs] by plugging Equation 3 into Equation 4:

$$\begin{aligned} U[nmembs] = max\ \{&U[RULE\_TABLE[rule]\ [ninput + output]]\ , \\ &min\ \ (X[input]\ [RULE\_TABLE[rule]\ [input]])\} \end{aligned} \quad \textbf{(5)}$$

The following example illustrates the fuzzy inference section. One cycle of the loop for rule=0 is shown. For the input array:

$$\begin{aligned} X[ninputs]\ [nmembs] &= |\ \ 0\ \ 0\ \ 70\ \ 15\ \ 0\ \ | \\ &= |\ \ 0\ \ 60\ \ 25\ \ 0\ \ 0\ \ | \end{aligned} \quad \textbf{(6)}$$

and for rule=0, input=0, and input=1, plug into Equation 3:

$$minZ = min \begin{vmatrix} (X[0]\ [2]) \\ (X[1]\ [2]) \end{vmatrix} = min \begin{vmatrix} 70 \\ 25 \end{vmatrix} = 25 \quad \textbf{(7)}$$

then for the max rule=0, input=0, and input=1, plug into Equation 4:

$$U[nmembs] \;=\; U[2] \;=\; max \begin{vmatrix} U[2] \\ minZ \end{vmatrix} \;=\; max \begin{vmatrix} 0 \\ 25 \end{vmatrix} \;=\; 25 \qquad \textbf{(8)}$$

This process continues for the list of 11 rules so that the array U[nmembs] contains the maximum values of the min—i.e., the contribution of each rule to the inference.

The final step involves defuzzifying the U[noutputs][nmembs] array. Since noutputs=1, U simplifies to U[nmembs]. The U[nmembs] array now has five values in it for its corresponding five positions. The center-of-gravity calculation is done with two loops. The first finds the numerator by using the multiplier to weight U[nmembs] by its position. The second loop finds the denominator by summing the position. The inevitable 16-bit divide loop then finds the output value u[output]. This u[output] represents the motor current mentioned in *Fuzzy Logic Theory for Servo Motor Control* (page 11). The divide operation is done on the ’C14 by a 16-cycle loop. This value is then scaled for the output and written to the NewServo memory location that sends it to the PWM generator.

As an example of defuzzification,

if U[nmemb] = [0 15 70 35 0],

then the output is:

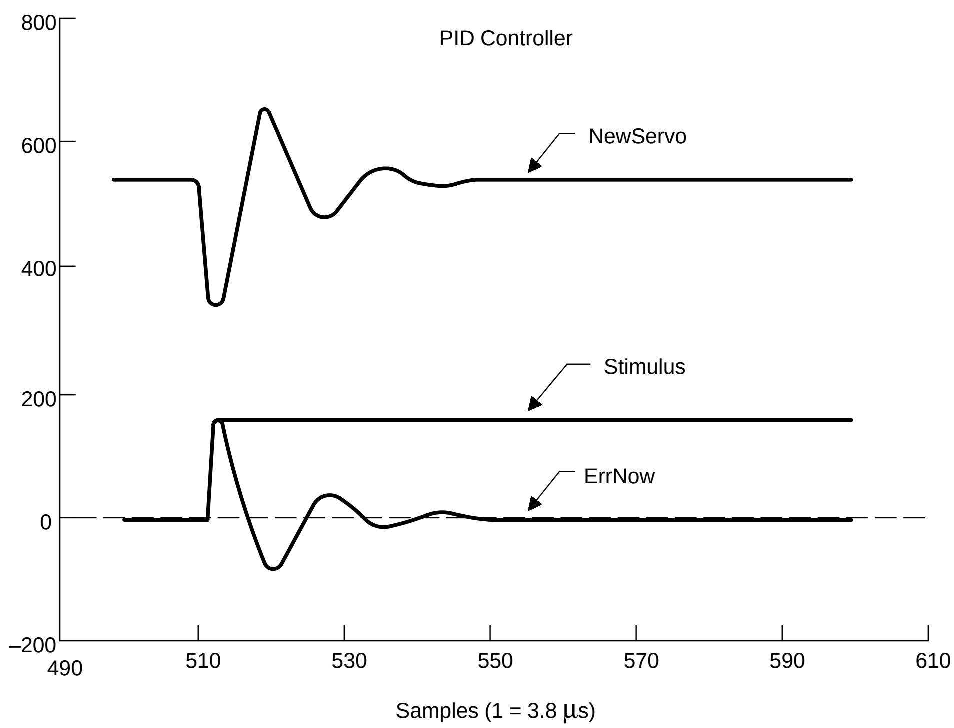
U[output] = 0*(–170) + 15*(–85) + 70*(0) + 35*(85) + 0*(170)
= 14.17

These loops thus evaluate the control input necessary for the servo motor.

Results

The fuzzy logic code with two inputs, eleven rules, one output, and five linguistic variables requires about 2000 instruction cycles to execute on the TMS320C14. This means a 400-µs period (2.5-kHz frequency) because one instruction cycle is 200 nanoseconds on the ’C14. The update period of the motor is 3.8 ms (263-Hz frequency), so the fuzzy code is obviously adequate (for comparison, the PID code execution required nine µs (111-kHz frequency), for the update period). Figure 6 shows the PID performance for a step with the classical control overshoot and settling times.

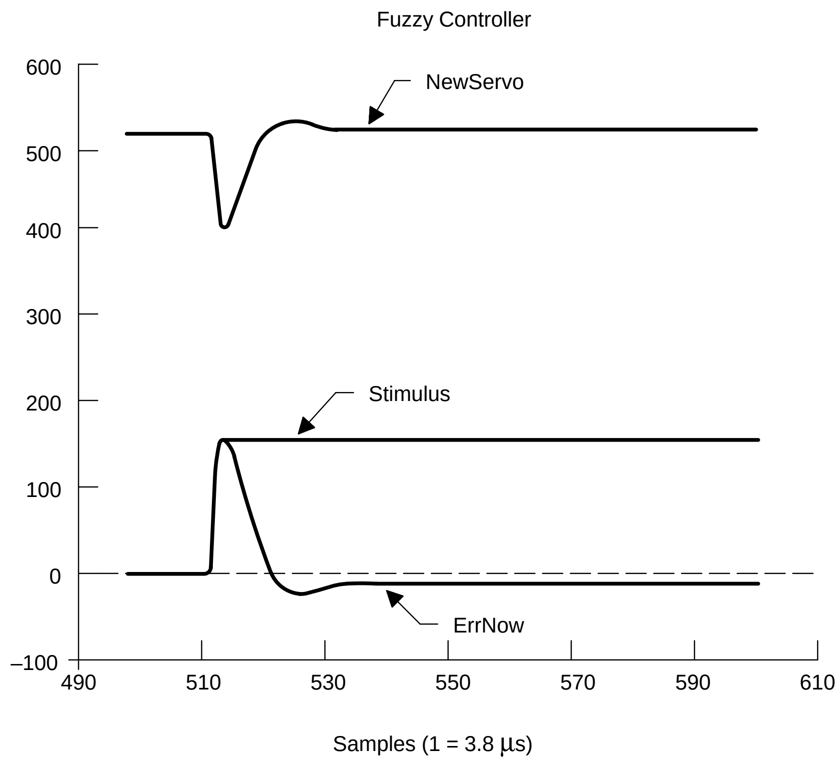
Figure 6. PID Controller



17

The fuzzy logic curve (Figure 7) is smoother than the PID controller curve but doesn’t go to zero.

Figure 7. Fuzzy Controller



The update period is adequate for applications such as servo motor, robotics, motion control, and automotive control. Better performance may be needed for such applications as hard-disk drives. Later-generation digital signal processors, such as the TMS320C5x, operate at up to 25 nanoseconds with much more efficient instruction code.

Conclusion

The final fuzzy logic system behaved favorably when compared to the conventional PID system. The system proved the feasibility of implementing a real-time fuzzy logic servo motor control. Proof of the fuzzy control was shown by positioning the motor spindle with an error outside the membership function, thus causing the control to desist. Further development could include a graphics display and the ability to vary the rules. The TMS320C14 board fits in a 12 x 8 x 6-inch suitcase conveniently and requires only an AC power supply and an RS-232 keyboard connection. This product easily demonstrates real-system fuzzy logic control on a microprocessor.

Fuzzy logic has great potential as a programmable solution for general engineering. For applications where performance is a priority, a hard-wired silicon solution (which may even configure as a microprocessor peripheral) based on [4] is being developed. The programmable solution may assist in the transition to this hard-wired option, depending on the software/hardware tradeoffs.

APPENDIX A
PID Code

```
.title "Servo compensator                    $Revision:   3.4  $"
; ****************************************************************************
; $Header::   C:/src/c14/ps/vcs/comp.s_v   3.4   01 Oct 1991 17:24:18   "$
;
; $config$="/T8 /K! /L;*.ref.def /R:*– /B80"
; !config!="/Mcomp.s"
;
; !NAME!
;               comp.s
;
; !PATHS!
;               modules
;                                 !0!
; DESCRIPTION
;               Servo compensator – uses a PID algorithm
;
; PRINCIPLE AUTHORS:
;               Dave Sewhuk
;
; CREATION DATE:
;               December 23, 1990  22:54:08
;
; COPYRIGHT NOTICE:
;               (C)Copyright 1990 Teknic Inc.  All rights reserved.
;
; !end!
; ****************************************************************************
; ****************************************************************************
; HEADERS UTILIZED
                .include "macrodef.inc"
                .include "c14io.inc"
;
; ****************************************************************************
; ****************************************************************************
; !NAME!
;               comp.s
;
; !PATHS!
;               Exported Variables
; !0!
                .def   PwmChannel,PwmPeriod
                .def   NewServo,ErrDiff,Kintegrator
                .def   ErrNow,ErrLast,DesiredPosition
                .bss   PwmChannel,1            ; Current PWM channel
                .bss   PwmPeriod,1             ; Current period of PWM channels
                .bss   NewServo,1              ; New servo value
                .bss   ErrDiff,1               ; Error difference
                .bss   Kintegrator,1           ; Error integrator
                .bss   ErrNow,1                ; Current error function
                .bss   ErrLast,1               ; Last error function result
                .bss   DesiredPosition,1       ; Servo to this position
```


19

```
; !end!
; ***************************************************************************

; ***************************************************************************
; !NAME!
;           comp.s
;
; !PATHS!
;           Local Variables
; !0!

; !end!
; ***************************************************************************

;****************************************************************************
            .text
            .def   servo_ISR
; !NAME!
servo_ISR:
;
; !PATHS!
;           functions\all
;           functions\interrupt
;           modules\utils.s
; !0!
; DESCRIPTION:
;
;           This is the servo compensator. The distribution algorithm is
;           the PID format.  The compensator performs the following function
;           PWM = 9*P*error + SUM(9*ek)*I + 9*(err diff)*D
;           with the PWM output clipped to timer hardware limits.
;
; RETURNS:
;           No changes to foreground environment.
;
; STACK LEVELS:
;           0
;
; EXTERNAL DATA REFERENCES
            .ref   Position            ; Position from feedback ISR
            .ref   ONE,CONSTFFFF       ; Useful constants
            .ref   Pgain,Igain,Dgain   ; Servo constants from user interface
            .ref   ISR_TMP             ; Scratch
;
;****************************************************************************
;!skip start!

            SOVM
; Saturate math
            LAC  ErrNow                          ; Save old error
            SACL          ErrLast
```

```
        ZALH    DesiredPosition
; Calc new error
        SUBH    Position
        SACH    ErrNow
;
; Run PID algorithm on new sample
;
        LAC  ErrNow                 ; Update error difference
        SUB  ErrLast
        SACL    ErrDiff

        ZALH    ErrDiff             ; Scale up value a bit/saturation
        ADDH    ErrDiff             ; *2
        ADDH    ErrDiff             ; *3
        ADDH    ErrDiff             ; *4
        ADDH    ErrDiff             ; *5
        ADDH    ErrDiff             ; *6
        ADDH    ErrDiff             ; *7
        ADDH    ErrDiff             ; *8
        ADDH    ErrDiff             ; *9
        SACH    ErrDiff

        ZALH    ErrNow              ; Scale up/saturation
        ADDH    ErrNow              ; *2
        ADDH    ErrNow              ; *3
        ADDH    ErrNow              ; *4
        ADDH    ErrNow              ; *5
        ADDH    ErrNow              ; *6
        ADDH    ErrNow              ; *7
        ADDH    ErrNow              ; *8
        ADDH    ErrNow              ; *9
        SACH    ISR_TMP
; Saved scaled up error

        ZALH    Kintegrator
; Update integrator
        ADDH    ErrNow
        SACH    Kintegrator

        LT   Pgain                  ; Do proportional part
        MPY ISR_TMP
        PAC

        LT   Igain                  ; Do integral part
        MPY Kintegrator

        LTA  Dgain                  ; Do differential part
        MPY ErrDiff
        APAC
        SACH    NewServo            ; Save result
;
; Check the answer so that it fits within the limits of the timer's
; period: [0..PwmPeriod*4)
;
        LAC  PwmPeriod,1            ; Check positive: 2*Period–4
        SUB  ONE,3
        SUB  NewServo               ;   – value
```

```
        BGZ  pwmisr20
        LAC  PwmPeriod,1            ; Oops, pin at biggest
        SUB  ONE,3
        SACL      NewServo
        B    pwmisrSet
pwmisr20:                           ; Negative OK?
        LAC  NewServo
        ADD  PwmPeriod,1            ; Add largest offset
        BGEZ      pwmisrSet
        ZAC
        SUB  PwmPeriod,1            ; Make largest negative
        SACL      NewServo
;
; Set Output PWM to new value
;
pwmisrSet:
        LAC  PwmPeriod,1            ; Get center period value
        ADD  NewServo               ; Add newly computed value
        SACL      NewServo
        .if  ChipV1R1               ; PWM bug workaround
        .ref CONST3
        SUB  CONST3
        BLEZ      pwmisr10
; No fixes needed
        LAC  NewServo               ; Lower 2 bits set?
        AND  CONST3
        SUB  CONST3
        BNZ  pwmisr10               ; OK, No fixes needed
        LACK      4
        ADD  NewServo
        SACL      NewServo
pwmisr10:
        .endif
        LACK      ActionBank        ; Set bank for timers
        SACL      ISR_TMP
        OUT  ISR_TMP,BSR
        OUT  NewServo,ACT0          ; Output channel 1 timer
        LAC  PwmPeriod,2            ; Calculate complimentary output
        SUB  ONE,2
        SUB  NewServo
        SACL      NewServo
        OUT  NewServo,ACT1          ; Set complimentary channel 2 timer
        RET                         ; All done.. back to the ISR already
                                    ; in progress.
; !skip end!
; !END!
; ==========================================================================
; END OF FILE
; ==========================================================================
        .end
```

APPENDIX B
Fuzzy Logic Code

```
        .title ”Fuzzy Servo compensator      Revision:   3.4  $”
; ***************************************************************************
; $Header::   C:/src/c14/ps/vcs/comp.s_v   3.4   01 Oct 1991 17:24:18   ”$
;
; $config$=”/T8 /K! /L;*.ref.def /R:*– /B80”
; !config!=”/Mcomp.s”
;
; !NAME!
;       compfuz.s
;
; !PATHS!
;                   modules
;                                                          !0!
; DESCRIPTION
;       Servo compensator – uses a fuzzy algorithm  (modifying PID code)
;
; PRINCIPAL AUTHORS:
;                   Dave Sewhuk
;                   Joe George w/ Fuzzy based on C++ code by Phillip Thrift
; CREATION DATE:
;                   December 23, 1990  22:54:08
;                   Jan. 1992
; COPYRIGHT NOTICE:
;                   (C) Copyright 1990 Teknic Inc.  All rights reserved.
;                   (C) Copyright 1992 Texas Instruments Incorporated.  All rights reserved.
; !end!
; ***************************************************************************

; ***************************************************************************
; HEADERS UTILIZED

                    .include ”macrodef.inc”
                    .include ”c14io.inc”
;
; ***************************************************************************

; ***************************************************************************
; !NAME!
;                   comp.s
;
; !PATHS!
;                   Exported Variables
; !0!
                    .def    PwmChannel,PwmPeriod
                    .def    NewServo,ErrDiff,Kintegrator
                    .def    ErrNow,ErrLast,DesiredPosition
                    .def    ninput, noutput, triangle, nmemb
                    .def    NM, NS, ZE, PS, PM, nloc, NMw, NSw, Zw, PSw, PMw
                    .def    nrule, rule_table, input, output, rule, memb
                    .def    loc, num, den, TEMP, TEMPH, x, X, U
```

```
            .bss   PwmChannel,1          ; Current PWM channel
            .bss   PwmPeriod,1           ; Current period of PWM channels
            .bss   NewServo,1            ; New servo value
            .bss   ErrDiff,1             ; Error difference
            .bss   Kintegrator,1         ; Error integrator
            .bss   ErrNow,1              ; Current error function
            .bss   ErrLast,1             ; Last error function result
            .bss   DesiredPosition,1     ; Servo to this position

; The following are fuzzy section, initialized and uninitialized

scalein     .set   256/3
scaleout    .set   2*100/3

;           .sect  ”pfuzzin”
            .asect ”pfuzzin”, 10h
            .label prule_table

rule_table:
            .word  2,2,2
            .word  3,2,1
            .word  4,2,0
            .word  1,2,3
            .word  0,2,4
            .word  2,1,3
            .word  2,0,4
            .word  2,3,1
            .word          2,4,0
            .word  3,1,2
            .word  1,3,2

ninput      .word  2
noutput     .word  1

triangle    .word  3
nmemb       .word  5
NMaddr      .word  NM
NM          .word  –3*scalein,  –2*scalein,  –1*scalein
NS          .word  –2*scalein,  –1*scalein,   0*scalein
ZE          .word  –1*scalein,   0*scalein,   1*scalein
PS          .word   0*scalein,   1*scalein,   2*scalein
PM          .word   1*scalein,   2*scalein,   3*scalein

nloc        .word  5
NMwaddr     .word  NMw
NMw         .word  –2*scaleout
NSw         .word  –1*scaleout
Zw          .word   0*scaleout
PSw         .word   1*scaleout
PMw         .word   2*scaleout
```

```
nrule           .word  11
rule_tableaddr .word   rule_table
input           .word  0
output          .word  0
rule            .word  0
memb            .word  0
loc             .word  0
num             .word  0
den             .word  0
minZ            .word  0

TEMP            .word  0
TEMPH           .word  0
ONE2            .word  1

xaddr           .word  x
Xaddr           .word  X
Uaddr           .word  U

rule_end:

rambeg          .usect ”fuzzin”, rule_end–rule_table

x               .usect ”fuzzunin”, 2
X               .usect ”fuzzunin”, 2*11
U               .usect ”fuzzunin”, 5

; !end!
; ****************************************************************************

; ****************************************************************************
; !NAME!
;               comp.s
;
; !PATHS!
;               Local Variables
; !0!

; !end!
; ****************************************************************************
```

```
;***************************************************************************
            .text
            .def    servo_ISR
; !NAME!
servo_ISR:
;
; !PATHS!
;           functions\all
;           functions\interrupt
;           modules\utils.s
; !0!
; DESCRIPTION:
;
;           This is the servo compensator. The distribution algorithm is
;           modified from PID to fuzzy format.
;
; RETURNS:
;           No changes to foreground environment.
;
; STACK LEVELS:
;           0
;
; EXTERNAL DATA REFERENCES
            .ref    Position                    ; Position from feedback ISR
            .ref    ONE,CONSTFFFF               ; Useful constants
            .ref    ISR_TMP                     ; Scratch
;
;***************************************************************************
;!skip start!

            SOVM                                ; Saturate math

; First need to copy data over from program

            LACK        prule_table
            ADD ONE, 11
            ADD ONE, 8

            LDPK        0
            LARK        AR0, 6Fh
            LARK        AR1, rambeg             ; beginning of RAM block

RAMLOOP: MAR    *, 1
            TBLR        *+
            LDPK        1
            ADD ONE
            LDPK        0
            MAR *,0
            BANZ        RAMLOOP

                                                ;  Get Theta and dTheta values

            LDPK        1
            LAC  ErrNow                         ; Save old error
            SACL        ErrLast

            ZALH        DesiredPosition         ; Calc new error
            SUBH        Position
            SACH        ErrNow
```

```
                LDPK        0
                MAR         *,0                     ; store x input value (Theta)
        LAR     AR0,xaddr
        SACH    *+

                LDPK        1
        LAC     ErrNow
        SUB     ErrLast
        SACL    ErrDiff

                LDPK        0
                SACL        *                       ; dTheta

 ; Calculate X
 ; indices for X[input][memberfn]
xfn:
        ZAC
        SACL    input                               ; for (i=0;
xfnil:          ZAC
        SACL    memb            ; for (j=0;
        LAC     ninput                              ;  i<ninput
        SUB     input
        BLEZ    ilend
xfnml:          LAC  nmemb                          ;  j<nmemb
        SUB     memb
        BLEZ    mlend
        LAC     xaddr                               ; load &x[input] AR0
        ADD     input
        SACL    TEMP
        LAR     AR0, TEMP
        LAC     NMaddr                              ; load &memb into AR1
        LT      triangle
        MPY     memb
         APAC
        SACL    TEMP
        LAR     AR1, TEMP
        CALL    membership                          ; Get membership value with AR's loaded
                                                    ; and leave answer in ACC
        SACL    TEMPH                               ; Save value
        LT      nmemb                               ; Calculate index for X[input][memb]
        MPY     input
        PAC
        ADD     memb
        ADD     Xaddr
        SACL    TEMP
        MAR     *,1
        LAR     AR1, TEMP
        LAC     TEMPH                               ; Store value in X[input][memb]
        SACL    *
        LAC     memb            ; memb++
        ADD     ONE2
        SACL    memb
        B       xfnml
mlend:          LAC  input                          ; input++
        ADD     ONE2
        SACL    input
```

```
      B           xfnil                            ; exit memb loop

ilend:            NOP
; exit input loop

; Zero out U values
      LAR         AR0, nmemb
      LAR         AR1, Uaddr
      ZAC
Uzl:  MAR         *,1
      SACL        *+
      MAR         *,0
      BANZ        Uzl

; calculate value of Z
zfn:
      ZAC
      SACL        rule                             ; for (i=0;
zfnrl:            ZAC
      SACL        input                            ; for (j=0;
      LAC         nrule                            ; i<nrule
      SUB         rule
      BLEZ        zrlend
;     LAC         ONE2                             ; initialize Z[rule]
      LAC         PM                               ; kluge since PM = 1*scalein
      SACL        minZ
zfnil:            LAC  ninput                      ; i<ninput
      SUB         input
      BLEZ        zilend
      LAC         ninput                           ; find &RULE_TABLE[rule][input]
      ADD         noutput
      SACL        TEMP
      LT          TEMP
      MPY         rule
      PAC
      ADD         input
      ADD         rule_tableaddr
      SACL        TEMP
      MAR         *,0                              ; find RULE_TABLE[rule][input]
      LAR         AR0, TEMP
      LAC         *
      SACL        TEMP                             ; TEMP = RULE_TABLE[rule][input]
      LT          nmemb                            ; find &X
      MPY         input
      PAC
      ADD         TEMP
       ADD        Xaddr
      SACL        TEMP                             ; &(X[input][RULE_TABLE[rule][input])
      LAR         AR0, TEMP                        ; check for min
      MAR         *,0
      LAC         minZ
      SUB         *
      BLZ         notmin
      LAC         *
      SACL        minZ
```

```
notmin:     LAC input                 ; input++
    ADD     ONE2
    SACL    input
    B       zfnil

; check max for U
zilend: LAC ninput                    ; get &U[RULE_TABLE[rule][input+output]
    ADD     noutput
    SACL    TEMP
    LT      TEMP
    MPY     rule
    PAC
    ADD     ninput                    ; total kluge for input+output
    ADD     rule_tableaddr
    SACL    TEMP
    MAR     *, 0
    LAR     AR0, TEMP
    LAC     *                         ; got RULE_TABLE[rule][input+output]
    ADD   Uaddr
    SACL    TEMP
    MAR     *,0
    LAR     AR0, TEMP
    LAC     minZ                      ;U[RULE_TABLE[rule][input+output]]
    SUB     *                         ;if U<minZ (looking for max)
    BLZ     notmax
    LAC     minZ          ;           store new U
    SACL    *
notmax:     LAC rule                  ; rule++
    ADD     ONE2
    SACL    rule
    B       zfnrl
zrlend: NOP

; Need to defuzzify output. u = num/den = sigma U*Loc/ sigma U
; (ignoring wt.)
defuzz:
    ZAC
    SACL    loc                       ; for (i=0;
    SACL    TEMP
    SACL    TEMPH
    LAR     AR0, NMwaddr
    LAR     AR1, Uaddr
```

```
; Sigma loc*U
dfl1:             LAC  nmemb                              ; i < nloc;
      SUB         loc
      BLEZ        dlend1
      MAR         *,0
      LT           *+
      MAR         *,1
      MPY         *+
      ZALH        TEMPH
      ADDS        TEMP
      APAC
      SACH        TEMPH
      SACL        TEMP
      LAC         loc                                     ; i++
      ADD         ONE2
      SACL        loc
      B           dfl1
dlend1: LAC       TEMP
      SACL        num    ; Store
      NOP

; Sigma U
      ZAC                                                 ; for (i=0;
      SACL        loc
      SACL        TEMP
      LAR         AR0, Uaddr
      MAR         *,0
dfl2:             LAC  nloc                               ; i<nloc;
      SUB         loc
      BLZ         dlend2
      LAC         TEMP
      MAR         *,0
      ADD         *+
      SACL        TEMP
      LAC         loc      ; ++i
      ADD         ONE2
      SACL        loc
      B           dfl2
dlend2: LAC       TEMP
      BZ          dzero
      SACL        den
      NOP
      B           cdiv
dzero:            ZAC
      B           endfuz

; divide num/den and scale for new servo
```

```
cdiv:       MAR *,0
      LT    TEMPH
      MPY   den
      PAC
      SACH  TEMP                          ; Sign
      LAC   den
      ABS
      SACL  den
      ZALH  TEMPH
      ADDS  num
      ABS
      LARK  AR0,15
DIV:        SUBC        den
      BANZ  DIV

      SACL  TEMPH                         ; Quotient
      LAC   TEMP                          ; Sign
      BGEZ  done
      ZAC
      SUB   TEMPH
      SACL  TEMPH                         ;  New Servo

done:       ZAC
      LAC   TEMPH
      B     endfuz

membership:
      MAR   *,0                           ; &x[input]
      LAC   *
      MAR   *,1                           ; &memb
      SUB   *+                            ; if x>x0
      BLEZ  zvalue
      MAR   *,0
      LAC   *
      MAR   *,1
      SUB   *+                            ; if x>x1
      BLZ   div1
      MAR   *,0
      LAC   *
      MAR   *,1
      SUB   *+                            ; if x>x2
      BLZ   div2
      B     zvalue
div1:   MAR *, AR0                        ; x–x0
      LAC   *
      MAR   *, AR1
      MAR   *–
      MAR   *–
      SUB   *
      RET
div2:   MAR *, AR1                        ; x2–x
      MAR   *–
      LAC   *
      MAR   *, AR0
      SUB   *
      RET
```

```
zvalue: LACK   0                                  ; value = 0
        RET

endfuz: LDPK   1
        SACL   NewServo                           ; Save result
        ZAC                                       ; negation kluge
        SUB    NewServo
        SACL   NewServo

; yoda:   B      servo_ISR

;
; Check the answer so that it fits within the limits of the timer's
; period: [0..PwmPeriod*4)
;
```

```
;	LACK	0FFh
;	SACL	PwmPeriod
	LAC	PwmPeriod,1		; Check positive: 2*Period–4
	SUB	ONE,3
	SUB	NewServo		;   – value
	BGZ	pwmisr20
	LAC	PwmPeriod,1		; Oops, pin at biggest
	SUB	ONE,3
	SACL	NewServo
	B	pwmisrSet
pwmisr20:				; Negative OK?
	LAC	NewServo
	ADD	PwmPeriod,1		; Add largest offset
	BGEZ	pwmisrSet
	ZAC
	SUB	PwmPeriod,1		; Make largest negative
	SACL	NewServo
;
; Set Output PWM to new value
;
pwmisrSet:
	LAC	PwmPeriod,1		; Get center period value
	ADD	NewServo		; Add newly computed value
	SACL	NewServo
	.if	ChipV1R1		; PWM bug workaround
	.ref	CONST3
	SUB	CONST3
	BLEZ	pwmisr10		; No fixes needed
	LAC	NewServo		; Lower 2 bits set?
	AND	CONST3
	SUB	CONST3
	BNZ	pwmisr10		; OK, No fixes needed
	LACK	4
	ADD	NewServo
	SACL	NewServo
pwmisr10:
	.endif
	LACK	ActionBank		; Set bank for timers
	SACL	ISR_TMP
	OUT	ISR_TMP,BSR
	OUT	NewServo,ACT0		; Output channel 1 timer
	LAC	PwmPeriod,2		; Calculate complimentary output
	SUB	ONE,2
	SUB	NewServo
	SACL	NewServo
	OUT	NewServo,ACT1		; Set complimentary channel 2 timer
	RET				; All done.. back to the ISR already
					; in progress.
; !skip end!
; !END!

; ==========================================================================
; END OF FILE
; ==========================================================================
	.end
```


33

References

[1] B. Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1992.

[2] Y.F. Li and C.C. Lau. "Development of Fuzzy Algorithms for Servo Systems," *IEEE Control Systems Magazine.* April, 1989, pp. 65-71.

[3] K. Self. "Designing with Fuzzy Logic," *IEEE Spectrum.* November, 1990, pp. 42-44, 105.

[4] P. Thrift. *The Programmable Fuzzy Logic Array.* Dallas, Texas: Texas Instruments Central Research Laboratories, 1992.

[5] *Power-14/Power Source User's Manual.* Rochester, New York: Teknic, Inc., 1989.

[6] *TMS320C1x User's Guide.* Dallas, Texas: Texas Instruments, 1991.

The Programmable Fuzzy Logic Array

Philip Thrift
Central Research Laboratories
Texas Instruments Incorporated

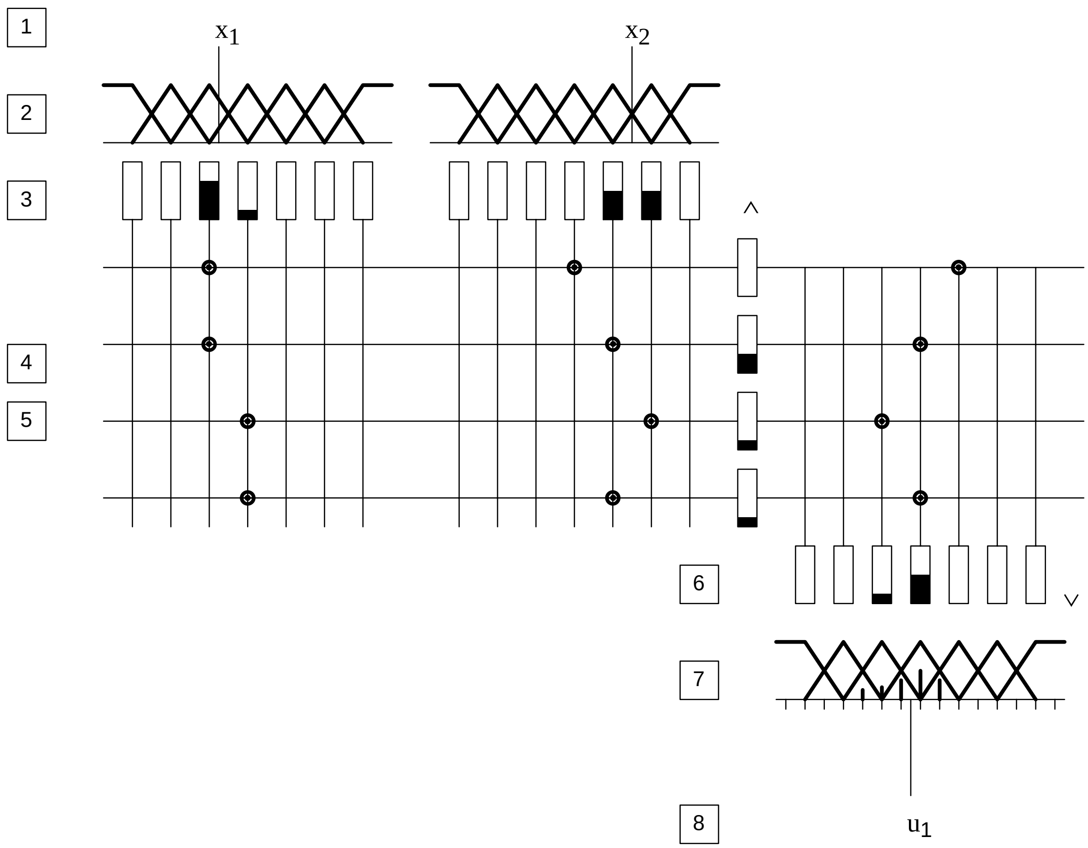
Introduction

The Programmable Fuzzy Logic Array (PFLA) produces nonlinear multidimensional mappings by encoding fuzzy rule systems in a programmable array architecture. It can be used as a component of a graphical-user interface (GUI) for designing fuzzy controllers, as well as a software blueprint for mapping onto VLSI hardware. An advantage of the PFLA over other fuzzy representations, such as the FAM (Fuzzy Associative Memory) [2], is the PFLA's ability to easily visualize several inputs and outputs simultaneously.

Data Flow

Figure 1 shows the PFLA data flow. Succeeding text describes PFLA in general mapping terms from *P* inputs to *Q* outputs, but only *P* = 2 and *Q* = 1 are shown in Figure 1. Each item number in the text corresponds to a step of Figure 1.

Figure 1. PFLA Data Flow



1. Each input x_i : i = 1,..., *P* to the PFLA is a numerical value in a range $[a_i,b_i]$.
2. Defined over each input range $[a_i, b_i]$: $i = 1,.., P$ are fuzzy membership functions $F_i^1, \ldots, F_i^{n_i}$, where n_i is the number of membership functions defined for input *i*. Each membership function varies between 0 and 1. The cases shown in Figure 1 are trapezoidal membership functions, which are defined in Appendix A. Other parametric families of membership functions can be substituted. Also, for each fuzzy membership function F_i^j, there is a corresponding label. A typical labeling scheme (labels are not shown in Figure 1) for $n_i = 7$ is: negative large (NL), negative medium (NM), negative small (NS), zero (ZE), positive small (PS), positive medium (PM), positive large (PL). Although the fuzzy sets for each input in Figure 1 appear symmetrically spaced, these are not necessarily the optimal settings. If trapezoidal memberships functions are used, four numbers $t_i^j = [t_{i,1}^j, t_{i,2}^j, t_{i,3}^j, t_{i,4}^j]$ must be specified for each membership function F_i^j.
3. Each input x_i is evaluated by each of its fuzzy membership functions to produce a value f_i^j :

$$f_i^j = F_i^j(x_i) \; : \; i = 1, \ldots, P \;\; j = 1, \ldots, n_i \tag{9}$$

These values appear in the boxes as shown as a thermometer level. f_i^j also refers to the box that contains its value. In the example shown, only two boxes for each input have positive evaluation.

4. Fuzzy rules are encoded in a crossbar pattern. Corresponding to each of the input boxes f_i^j in 3, above, is a vertical wire dropping down. A horizontal wire crosses those vertical wires and also the vertical wires corresponding to output boxes in 6, below. A connection is indicated by a •. For each horizontal wire, there is, at most, one connection per input and output variable. Each horizontal connection pattern encodes a rule. For example, the connection pattern on the first horizontal line encodes the rule:

$$\textit{If } x_1 \textit{ is } NS \text{ and } x_2 \textit{ is } ZE, \textit{ then } u_1 \textit{ is } PS. \quad (10)$$

Four rules are shown in Figure 1. *R* rules can be specified by a table of numbers:

$$r_1^1, \ldots, r_1^P, s_1^1, \ldots s_1^Q \quad (11)$$

$$r_R^1, \ldots, r_R^P, s_R^1, \ldots s_R^Q \quad (12)$$

where r_k^i is in {1,...,n_i,NULL}, s_k^j is in {1,...,m_j,NULL}. Here, m_j is the number of fuzzy sets for output j. The kth horizontal wire specifies the rule

$$F_1^{r_k^1}, \ldots, F_P^{r_k^P} \rightarrow G_1^{s_k^1}, \ldots, G_Q^{s_k^Q} \quad (13)$$

The NULL indicates that there is no connection for this input/output variable. In Figure 1, the connections would be

3,4,5
3,5,4
4,6,3
4,5,4

5. Intercepting the horizontal wires between the inputs and outputs are the $\wedge$ (“wedge”) boxes. A conventional $\wedge$ operator is the numerical minimum of the values (this is used in Figure 1), but other operators are possible (for example, product, or any of a set of so-called t-norms [1]).

At the kth $\bigwedge$ box, this is produced:

$$h_k = \bigwedge (f_1^{s_k^1}, f_2^{r_k^2}, \ldots, f_P^{r_k^P}) \quad \textbf{(14)}$$

If r_k^j is NULL, this argument is omitted.

6. The $\bigvee$ (“vee”) boxes are computed according to the values of the connections above them. A conventional $\bigvee$ operator is the numerical maximum of the values (see Figure 1). The values of the $\bigvee$ boxes are

$$g_j^i = \bigvee [h_k : s_k^i = j] \; i = 1, \ldots, Q \quad j = 1, \ldots, m_1 \quad \textbf{(15)}$$

   Other operators [1] can be substituted for this operator (for example, probabilistic sum: $x \oplus y = x + y - xy$, etc.).

7. Defined over each output range $[c_i, d_i]$: $i = 1,...,Q$ are fuzzy membership functions $G_1^1, \ldots, G_i^{m_i}$, where m_i is the number of membership functions defined for output i. Also defined for each output i is a vector l_i of locations quantizing the range: $1_i = [l_i^1, \ldots l_i^{q_i}]$, where q_i is the number of locations specified for output i. In Figure 1, 17 locations are shown for output 1.

$$w_i^j = G_i^j (1_i), \; i = 1, \ldots, Q \quad j = 1, \ldots, m_i \quad \textbf{(16)}$$

   Here, G_i^j is applied by component to get the resulting vector. The w_i^j are precomputed and stored for each $\bigvee$ box. Then, this is computed for output i:

$$v_i = \bigvee * (w_i^1 \bigwedge * g_i^1, \ldots, w_i^{m_i} \bigwedge * g_i^1) \quad \textbf{(17)}$$

   Here, the “wedge” and “vee” operators are not necessarily the ones used in the boxes above. In Figure 1, v_1 is shown as a sequence of vertical bars; $\bigvee$ * is the maximum operation, and $\bigwedge$ * is the product.

8. The final stage is to compute u_i from v_i for each output:

$$u_i = \frac{v_i \, . \, 1_i}{v_i \cdot 1} \quad \textbf{(18)}$$

   where 1 = [1,...,1] is a vector of 1s.

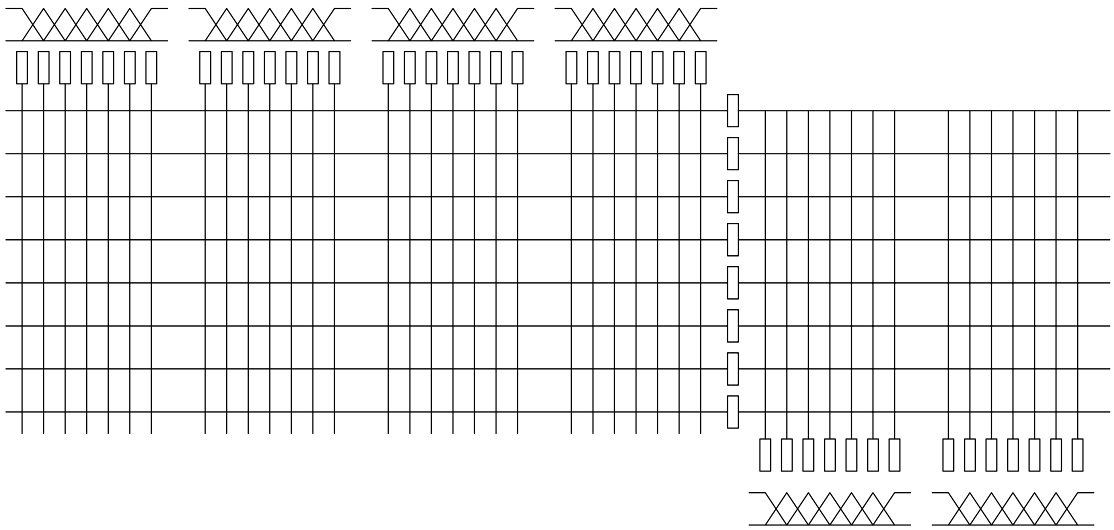
If, corresponding to each output fuzzy set G_i^j, there is a single distinct location l_i^j, $w_i^j = G_i^j(l_i^j)$, and $\bigwedge$ * is the product operation, then the above defuzzification procedure reduces to

$$u_i = \frac{w_i^1 \cdot g_i^1 \cdot l_i^1 + \ldots + w_i^{m_i} \cdot g_i^{m_i} \cdot l_i^{m_i}}{w_i^1 \cdot g_i^1 + \ldots + w_i^{m_i} \cdot g_i^{m_i}} \quad \textbf{(19)}$$

This provides “weighted point mass defuzzification”. In this case, only a “weight” w_i^j and a location l_i^j must be stored for each $\bigvee$ box.

Figure 2 shows the state of the system for particular inputs x_1, x_2. Thermometer levels indicate the values in both the fuzzification and the $\bigwedge$ and $\bigvee$ boxes (here, *min* and *max* are the operations performed). Darkened rule connections indicate values that propagated through the array (since *min* and *max* are used, on each output line there is, in general, one *max* winner and one *min* winner). Figure 2 also shows the layout for a four-input, two-output (4-2) system. In a GUI, you can use point-and-clicks to set the rule connections and membership function positioning.

Figure 2. PFLA 4–2 Layout



Summary

This is the information that provides the setting for the PFLA:

#Inputs: P	#Input fuzzy sets: $n_1,...,n_P$
#Outputs: Q	#Output fuzzy sets: $m_1,...,m_Q$
Input fuzzy sets	$[a_i,b_i]$, $t_i^1,..., t_i^{n_i}$, i = 1,..., P
#Rules: R	Operators: $\wedge \; \vee$
#Rule table:	$r_k^1, \ldots, r_k^P, s_k^1, ...s_k^Q, \quad k = 1, \ldots, R$
Output defuzzifiers	$[c_j, d_j], l_j, w_j^1, \ldots, w_j^{m_j}, \quad j = 1, \ldots, Q \quad \wedge * \; \vee *$

Appendix A

A trapezoidal membership function on an interval [a,b] is specified by four numbers t_1, t_2, t_3, t_4, satisfying

$$a \le t_1 \le t_2 \le t_3 \le t_4 \le b$$

The trapezoidal function is defined by

$$TZ(a,t_1,t_2,t_3,t_4,b)(x) \begin{cases} 0 & \text{for x in } [a,t_1] \\ (x - t_1)/(t_2 - t_1) & \text{for x in } (t_1,t_2) \\ 1 & \text{for x in } [t_2,t_3] \\ (t_4 - x)/(t_4 - t_3) & \text{for x in } (t_3,t_4) \\ 0 & \text{for x in } [t_4,b \end{cases}$$

Note that if $t_1 = t_2$ *or* $t_3 = t_4$, this part of the definition is void.

41

References

[1] Pedrycz, W. *Fuzzy Control and Fuzzy Systems.* John Wiley & Sons Inc., 1989.

[2] Kosko, B. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, Inc., 1992.