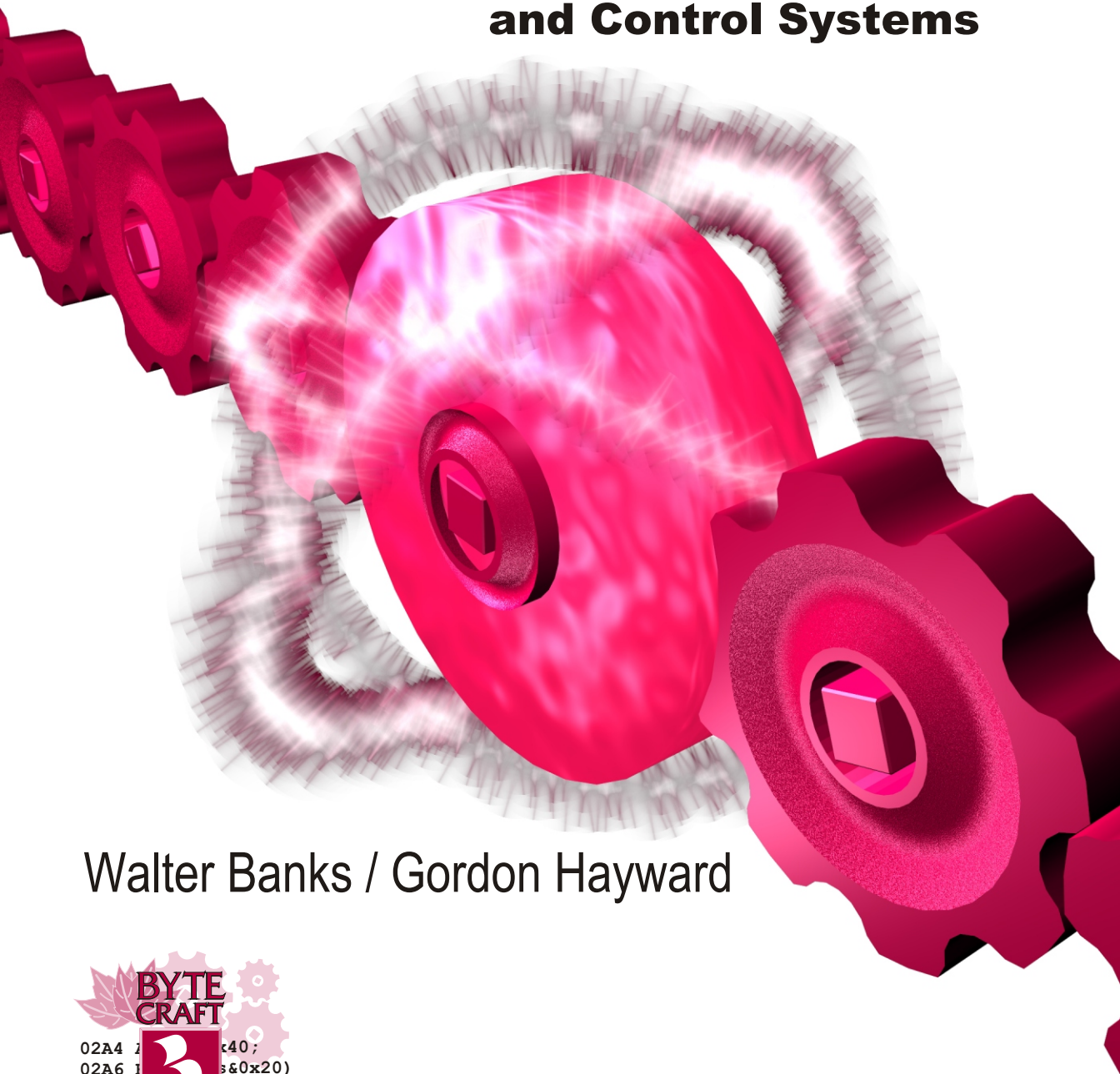
Fuzzy Logic

in Embedded Microcomputers and Control Systems

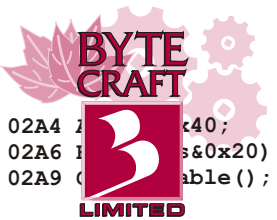


Walter Banks / Gordon Hayward

BYTE CRAFT LIMITED

Fuzz-C™

Fuzzy Logic Preprocessor for C



Fuzz-C™ is a stand-alone preprocessor that seamlessly integrates fuzzy logic into the C language. Now you can add fuzzy logic to your applications without expensive, specialized hardware or software. Fuzz-C accepts fuzzy logic rules, membership functions and consequence functions, and produces C source code that can be compiled by most C compilers, including the Byte Craft Limited Code Development System.

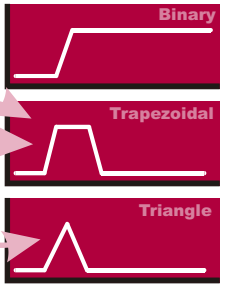
The preprocessor generates C code that is both compact and significantly faster than most current fuzzy logic commercial implementations—all with your favorite C compiler.

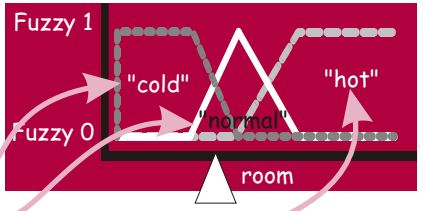
Fuzz-C provides a practical, unified solution for applications that require fuzzy logic control systems. Use your existing C libraries for program management, keyboard handlers and display functions without change; you can implement system control functions using fuzzy rules.

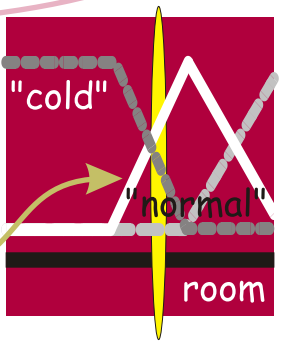
Fuzz-C is a flexible system that allows all data types supported by your C compiler. Standard defuzzification methods, such as *center of gravity*, *max left*, *max right*, and *max average*, are provided in source form. Fuzz-C lets you easily add new defuzzification methods.

```
/* Fuzzy Logic Climate Controller

This single page of code creates a fully
Functional controller for a simple air
conditioning system */

#define thermostat PORTA
#define airCon PORTB.7

/* degrees celsius */
LINGUISTIC room TYPE int MIN 0 MAX 50
{
  MEMBER cold   { 0, 0, 15, 20 }
  MEMBER normal { 20, 23, 25 }
  MEMBER hot    { 25, 30, 50, 50 }
}

/* A.C on or off */
CONSEQUENCE ac TYPE int DEFUZZ CG
{
  MEMBER ON  { 1 }
  MEMBER OFF { 0 }
}

/* Rules to follow */
FUZZY climateControl
{
  IF room IS cold THEN
    ac IS OFF
  IF room IS normal THEN
    ac IS OFF
  IF room IS hot THEN
    ac IS ON
}

int main(void)
{
  while(1)
    {
      /* find the temperature */
      room = thermostat;
      /* apply the rules */
      climateControl();
      /* switch the A.C. */
      airCon = ac;
      wait(10);
    }
}
```

Membership Functions



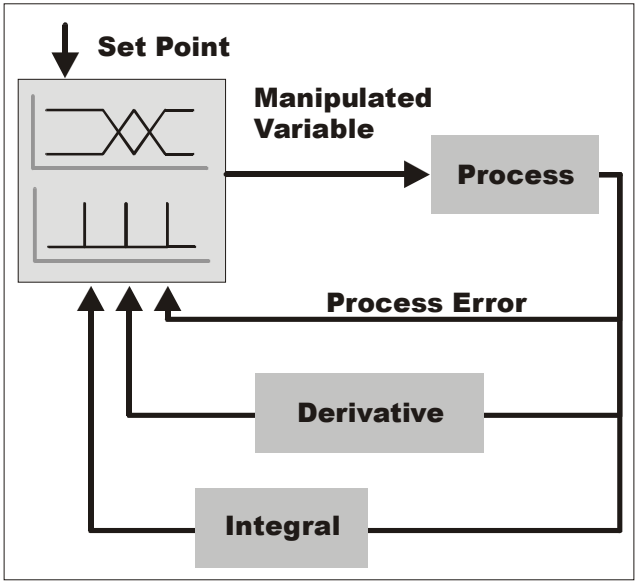




Center of Gravity Calculation

Terms: prepaid American Express, VISA or cheque. Overseas orders prepaid in U.S. funds drawn on a Canadian or U.S. bank only. Please obtain appropriate import documentation. Canadian customers are subject to applicable taxes. Specifications and price information subject to change without notice. Fuzz-C is a registered trademark of Byte Craft Limited. Other marks are trademarks or registered trademarks of their respective holders.

Fuzz-C™ includes one year technical support via phone or email. Fuzz-C requires modest system resources: DOS or Windows and less than 1 megabyte of memory. Fuzz-C works with *make* and other industry-standard build systems. Complete documentation is included.



Fuzzy Logic

in Embedded Microcomputers and Control Systems

Walter Banks / Gordon Hayward

Published by

BYTE CRAFT LIMITED

A2-490 Dutton Drive

Waterloo, Ontario

Canada • N2L 6H7

Sales Information and Customer Support:

BYTE CRAFT LIMITED

A2-490 Dutton Drive
Waterloo, Ontario
Canada NL 6H7

Phone	(519) 888-6911
FAX	(519) 746-6751
Web	www.bytecraft.com

Printed in Canada October, 2002

First Web Release October 2002

Forward

This booklet started as a result of the rush of people who asked for copies of the overhead slides I used in a talk on **Fuzzy Logic For Control Systems** at the 1993 Embedded Systems Show in Santa Clara.

A fuzzy logic tutorial

There is a clear lack of basic tutorial materials for fuzzy logic. I decided that I did have enough material to create a reasonable tutorial for those beginning to explore the possibilities of fuzzy logic. In addition to the material presented at the embedded systems conference I have added additional chapters.

Clear thinking on fuzzy linguistics

The first chapter essentially consists of the editorial I wrote for *Electronic Engineering Times* (printed on October 4, 1993). The editorial presented a case for the addition of linguistic variables to the programmer's toolbox.

Fuzzy logic implementation on embedded microcomputers

The second chapter is based upon a paper I presented at **Fuzzy Logic '93** by Computer Design in Burlingame, CA (in July of 1993). This paper described the implementation considerations of fuzzy logic on conventional, small, embedded micro-computers. Many of the paper's design considerations were essential to the development of our ***Fuzz-C***® preprocessor. I have created most of the included examples in ***Fuzz-C*** and although you don't need to use ***Fuzz-C*** to implement a fuzzy logic system, you will find it useful to understand some of its design.

Software Reliability and Fuzzy Logic

Originally part of the implementation paper, this chapter presents what is actually a separate subject. The inherent reliability and self scaling aspects of fuzzy logic are becoming important and may in fact be the over riding reason for the use of fuzzy logic.

Appendix

The appendix contains, in addition to copies of the slides, the actual code for a fuzzy PID controller as well as the block diagram of the PID controller used in my Santa Clara talk entitled **Fuzzy Logic For Control Systems**.

Adjusting to fuzzy design

While presenting the paper in Santa Clara, much of the discussion touched on provable control stability. This final issue has discouraged many engineers from employing fuzzy logic in their designs. Despite the great incentive to use fuzzy logic, I found it took me about a year and a half to feel comfortable with the addition of linguistic variables to my software designs.

Fuzzy logic is not magic, but it has made many problems much easier to visualize and implement. Debugging has generally been straight forward in my own code, and I think that most who have implemented fuzzy logic applications share this opinion.

I have tried to make the material presented both in this booklet, and in my presentations in public, as non-commercial as possible. The purpose here is to inform and educate. Some of the slide material came from Dr. Gordon Hayward of the University of Guelph. Gord is a friend and colleague dating back more than twenty years. Gord was the first to look at fuzzy logic through transfer functions. The slides of the actual control system response were generated by a student of Dr. Hayward's in a report (L. Seed 05-428 Project, Winter 1993). I thank both of them for this material.

Much material has been published on fuzzy logic and linguistic variables. Most of the literature available in the English-speaking world was written primarily by and for mathematicians, with few papers and articles written for computer scientists or system implementors. This work started with a paper by Lotfi Zadeh more than a quarter century ago ("Fuzzy Sets", Information and Control 8, pp. 338-353, 1965). Professor Zadeh has remained a tireless promoter of the technology.

At the 1992 Embedded Systems Conference in Santa Clara, the genie was finally let out of the bottle, and fuzzy logic came into its own with wide interest. Jim Sibigtroth's article in *Embedded Systems Programming* magazine in December, 1991 cracked the bottle, describing for the first time a widely available, understandable implementation of a fuzzy logic control system workable for general purpose microprocessors. Jim Sibigtroth has been working on the promotion of fuzzy logic control systems to the point of personal passion. As developers began to understand the real power of using linguistic variables in control applications, the negative implications of the name *fuzzy logic* have given way to a deep understanding that this is a powerful tool backed by solid mathematical principles.

I thank all those who work with me at Byte Craft Limited for their efforts. A special thanks to Viktor Haag who gets to do much of the hard work for our printed material and far too little credit. For me I accept responsibility for all of the errors and inconsistencies.

Walter Banks
October 28, 1993.

Clear thinking on fuzzy linguistics

I have had a front row seat, watching a computing public finding uses for an almost 30 year-old *new* technology.

Personally, I struggled with finding an application to clearly define what all the magic was about, until I switched the question around and looked at how an ever increasing list of fuzzy logic success stories might be implemented. I then looked at the language theory to see why *linguistic variables* were important in describing and solving problems on computers.

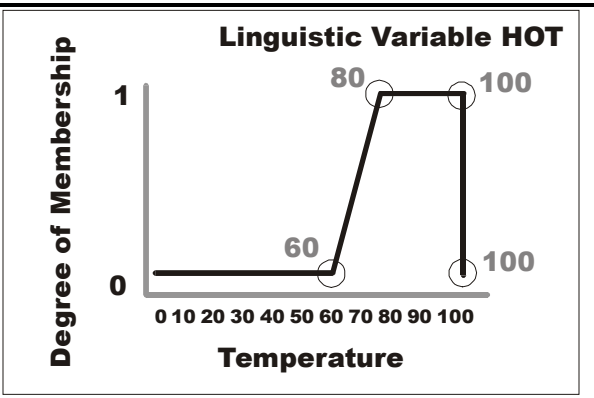
Linguistic variables are central to fuzzy logic manipulations. Linguistic variables hold values that are uniformly distributed between 0 and 1, depending on the relevance of a context-dependent linguistic term. For example; we can say *the room is hot* and *the furnace is hot*, and the linguistic variable *hot* has different meanings depending on whether we refer to the room or the inside of the furnace.

The assigned value of 0 to a linguistic variable means that the linguistic term is not true and the assigned value of 1 indicates the term is true. The "linguistic variables" used in everyday speech convey relative information about our environment or an object under observation and can convey a surprising amount of information.

The relationship between crisp numbers and linguistic variables is now generally well understood. The linguistic variable **HOT** in the following graph has a value between 0 and 1 over the crisp range 60-80 (where 0 is not hot at all and 1 is undeniably hot). For each crisp number in a variable space (say room), a number of linguistic terms may apply. Linguistic variables in a computer require a formal way of describing a linguistic variable in the crisp terms the computer can deal with.

The following graph shows the relationship between measured room temperature and the linguistic term *hot*. In the space between *hot* and *not hot*, the temperature is, to some degree, a bit of both.

The horizontal axis in the following graph shows the measured or crisp value of temperature. The vertical axis describes the degree to which a linguistic variable fits with the crisp measured data.



Most fuzzy logic support software has a form resembling the following declaration of a linguistic variable. In this case, a crisp variable *room* is associated with a linguistic variable *hot*, defined using four break points from the graph.

```
LINGUISTIC room TYPE unsigned int MIN 0 MAX 100
 {
     MEMBER HOT { 60, 80, 100, 100 }
 }
```

We often use linguistic references enhanced with crisp definitions.

Cooking instructions are linguistic in nature: "Empty contents into a saucepan; add 4½ cups (1 L) cold water." This quote from the instructions on a Minestrone soup mix packet shows just how common linguistic references are in our descriptive language. These instructions are in both the crisp and fuzzy domains.

The linguistic variable "saucepan", for example, is qualified by the quantity of liquid that is expected. One litre (1 L) is not exactly 4½ cups but the measurement is accurate enough (within 6.5%) for the job at hand. "Cold water " is a linguistic variable that describes water whose temperature is between the freezing point (where we all agree it is cold) to some higher temperature (where it is cold to some degree).

The power of any computer language comes from being able to describe a problem in terms that are relevant to the problem. Linguistic variables are relevant for many applications involving human interface. Fuzzy logic success stories involve implementations of tasks commonly done by humans but not easily described in crisp terms.

Rice cookers, toasters, washing machines, environment control, subway trains, elevators, camera focusing and picture stabilization are just a few examples. Linguistic variables do not simplify the application or its implementation but they provide a convenient tool to describe a problem.

Applications may be computed in either the fuzzy linguistic domain or the conventional crisp domain. Non-linear problems, such as process control in an environment that varies considerably from usage to usage, yield very workable results with impressively little development time when solved using fuzzy logic. Although fuzzy logic is not essential to solving this type of non-linear control problem, it helps in describing some of the possible solutions.

Dr. Lotfi Zadeh, the originator of fuzzy logic, noted that ordinary language contains many descriptive terms whose relevance is context-specific. I can, for example, say that *the day is hot*. That statement conveys similar information to most people. In some ways, it conveys better information than saying *the temperature is 35 degrees*, which implies hot in most European countries and quite cool in the United States.

The day is muggy implies two pieces of information: the day is hot and the relative humidity is high. We can have a day that is hot or muggy or cold or clammy. In common usage linguistic variables are often overlapping.

Muggy implies both high humidity and hot temperatures. The variable *day* may have an extensive list of linguistic values computed in the fuzzy domain associated with it (**MUGGY, HUMID, HOT, COLD, CLAMMY**). If *day* is a linguistic variable, it doesn't have a crisp number associated with it so that although we can say the day is **HOT** or **MUGGY**, assigning a value to *day* is meaningless. All of the linguistic members associated with *day* are based on fuzzy logic equations.

When fuzzy logic is used in an application program, it adds linguistic variables as a new variable type. We might implement an air conditioner controller with a single fuzzy statement

```
IF room IS hot THEN air_conditioner is on;
```

We can extend basic air conditioner control to behave differently depending on the different types of *day*.

The math developed to support linguistic variable manipulation conveniently implements an easy method to switch smoothly from one possible solution to another. This means that, unlike a conventional control system that easily implements a single well behaved control of a system, the fuzzy logic design can have many solutions (or rules) which apply to a single problem and the combined solutions can be appropriately weighted to a controlling action.

Computers–especially those in embedded applications–can be programmed to perform calculations in the fuzzy domain rather than the crisp domain. Fuzzy logic manipulations take advantage of the fact that linguistic variables are only resolved to crisp values at the resolution of the problem, a kind of self scaling feature that is objective-driven rather than data-driven.

To keep a room comfortable, the temperature and humidity need to be kept only within the fuzzy comfort zone. Any calculations that have greater accuracy than the desired result are redundant, and require more computing power than is needed. Fuzzy logic is not the only way to achieve reductions in computing requirements but it is the best of the methods suggested so far to achieve this goal.

Linguistic variable types are taking their place alongside such other data types as *character*, *string*, *real* and *float*. They are, in some ways, an extension to the already familiar enumerated data types common in many high level languages. In my view, the linguistic domain is simply another tool that application developers have at their disposal to communicate clearly. When applied appropriately, fuzzy logic solutions are competitive with conventional implementation techniques with considerably less implementation effort.

Fuzzy logic implementation on embedded microcomputers

Fuzzy logic operators provide a formal method of manipulating linguistic variables. It is a reasonable comment to describe fuzzy logic as just another programming paradigm. Fuzzy logic critics are correct in stating that they can do with conventional code everything that fuzzy logic can do. For that matter, so can machine code, but I am not going to argue the point.

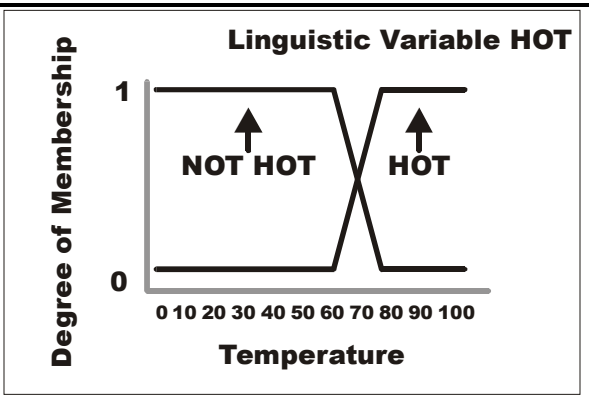
Central to fuzzy logic manipulations are *linguistic variables*. Linguistic variables are non-precise variables that often convey a surprising amount of information. We can say, for example, that it is warm outside or that it is cool outside. In the first case we may be going outside for a walk and we want to know if we should wear a jacket so we ask the question, *what is it like outside?*, and the answer is *it is warm outside*.

Experience has shown that a jacket is unnecessary if it is warm and it is mid-day; but, warm and early evening might mean that taking a jacket along might be wise as the day will change from warm to cool. The linguistic variables so common in everyday speech convey information about our environment or an object under observation.

In common usage, linguistic variables often overlap. We can have a day in Boston that is, *hot and muggy,* indicating high humidity and hot temperatures. Again, I have described one linguistic variable in linguistic variable terms. The description *hot and muggy* is quite complex. *Hot* is simple enough as the following description shows.

Linguistic variables in a computer require a formal way of describing a linguistic variable in crisp terms the computer can deal with. The following graph shows the relationship between measured temperature and the linguistic term *hot*. Although each of us may have slightly differing ideas about the exact temperature that *hot* actually indicates, the form is consistent.

At some point all of us will say that *it is not hot* and at some point we will agree that *it is hot*. The space between *hot* and *not hot* indicates a temperature that is, to some degree, a bit of both. The horizontal axis in the following graph shows the measured or *crisp value* of temperature. The vertical axis describes the degree to which a linguistic variable fits with the crisp measured data.



We can describe temperature in a non-graphical way with the following declaration. This declaration describes both the crisp variable **Temperature** as an *unsigned int* and a linguistic member **HOT** as a trapezoid with specific parameters.

```
LINGUISTIC Temperature TYPE unsigned int MIN 0 MAX 100
 {
     MEMBER HOT { 60, 80, 100, 100 }
 }
```

To add the linguistic variable **HOT** to a computer program running in an embedded controller, we need to translate the graphical representation into meaningful code. The following ***C*** code fragment gives one example of how we might do this. The function **Temperature_HOT** returns a degree of membership, scaled between 0 and 255, indicating the degree to which a given temperature could be **HOT**. This type of simple calculation is the first tool required for calculations of fuzzy logic operations.

```
unsigned int Temperature; /* Crisp value of Temperature */
unsigned char Temperature_HOT (unsigned int __CRISP)
 {
     if (__CRISP < 60) return(0);
     else
     {
         if (__CRISP <= 80) return(((__CRISP - 60) * 12) + 7);
         else
         {
             return(255);
          }
 }
 }
```

The same code can be translated to run on many different embedded micros, as displayed in the next two examples.

Code for National COP8

```
0008                                unsigned int Temperature ;
                                    unsigned int Temperature_HOT
                                    (unsigned int __CRISP)
0009                                {
0005 56       LD B,#09      < 1 >
0006 A6       X A,[B]       < 1 >
0007 AE       LD A,[B]      < 5 >     if (__CRISP < 60) return(0);
0008 93 3B    IFGT A,#03B   < 2 >
000A 02       JP 0000D      < 3 >
000B 64       CLRA          < 1 >
000C 8E       RET           < 5 >
                                      else
                                      {
000D 56       LD B,#09      < 1 >       if (__CRISP <= 80)
                                        return (((__CRISP - 60) * 12) + 7);
000E AE       LD A,[B]      < 5 >
000F 93 50    IFGT A,#050   < 2 >
0011 10       JP 00022      < 3 >
0012 AE       LD A,[B]      < 5 >
0013 94 C4    ADD A,#0C4    < 2 >
0015 9C 00    X A,000       < 3 >
0017 BC 01 0C LD 001,#0C    < 3 >
001A AD 00 26 JSRL 00026    < 4 >
001D 9D 01    LD A,001      < 3 >
001F 94 07    ADD A,#007    < 2 >
0021 8E       RET           < 5 >
                                        else
                                        {
0022 98       FF LD A,#0FF  < 2 >         return(255);
0024 8E       RET           < 5 >
                                        }
                                      }
```

Code for Motorola MC68HC08

```
0050                                unsigned int Temperature ;

                                    unsigned int Temperature_HOT
                                    (unsigned int __CRISP)
0051                                {
0100 B7 51    STA $51     < 3 >
0102 A1 3C    CMP #$3C    < 2 >       if (__CRISP < 60) return(0);
0104 24 02    BCC $0108   < 3 >
0106 4F       CLRA        < 1 >
0107 81       RTS         < 4 >
                                      else
                                      {
0108 B6 51    LDA $51     < 3 >         if (__CRISP <= 80)
                                        return(((__CRISP - 60) * 12) + 7);
010A A1 50    CMP #$50    < 2 >
010C 22 08    BHI $0116   < 3 >
010E A0 3C    SUB #$3C    < 2 >
0110 AE 0C    LDX #$0C    < 2 >
0112 42       MUL         < 5 >
0113 AB 07    ADD #$07    < 2 >
0115 81       RTS         < 4 >
                                        else
                                        {
0116 A6 FF    LDA #$FF    < 2 >           return(255);
0118 81       RTS         < 4 >
                                        }
                                      }
                                    }
```

Central to the manipulation of fuzzy variables are fuzzy logic operators that parallel their boolean logic counterparts; *f_and, f_or* and *f_not*. We can define these operators as three macros to most embedded system *C* compilers as follows.

```
#define f_one 0xff
#define f_zero 0x00
#define f_or(a,b) ((a) > (b) ? (a) : (b))
#define f_and(a,b) ((a) < (b) ? (a) : (b))
#define f_not(a) (f_one+f_zero-a)
```

The linguistic variable **HOT** is straight forward in meaning; as the temperature rises, our perceived degree of **HOT**ness also rises, until and at some point we simply say *it is hot*.

Our description of the linguistic variable **MUGGY** is, however, more complex. Typically, we think of the condition **MUGGY** as a combination of **HOT** and **HUMID**.

We can describe a controlling parameter for an air conditioner with the following equation.

```
IF Temperature IS HOT AND Humidity IS HUMID THEN ACcontrol is MUGGY;
```

Different variables can have the same linguistic member names. Like members of a structure or enumerated type in most programming languages, they do not have to be unique. It is important to note that many of the linguistic conclusions are a result of the general form of the above equation.

We have linguistic definitions of the variable **day**. The variable **day** can have a number of linguistic terms associated with it. { **MUGGY**, **HUMID**, **HOT**, **COLD**, **CLAMMY**}. This list may be extensive.

What is interesting, is that **day,** although a linguistic variable, doesn't have a crisp number associated with it. For example we can say that the **day** is **HOT** or that the **day** is **MUGGY**, but saying that the **day = 29** is meaningless; **day** is a *void* variable.

All **day**'s members are based on fuzzy logic equations. The following is a complete description of **day**.

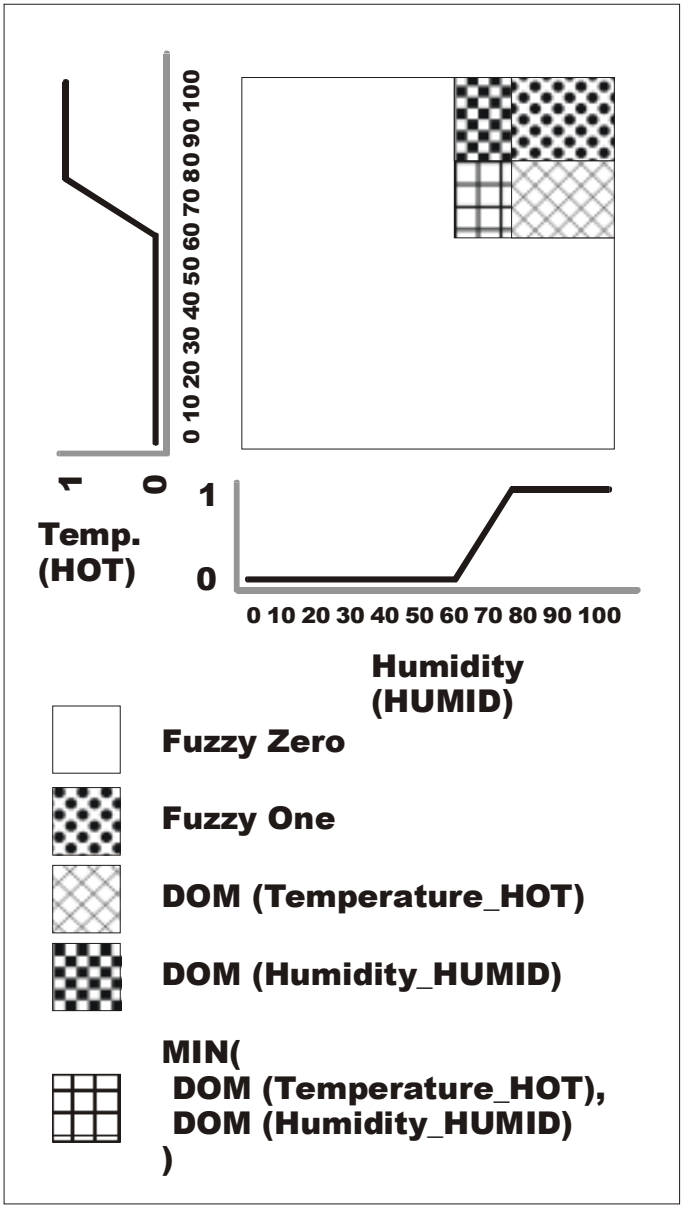
```
LINGUISTIC day TYPE void
 {
     MEMBER MUGGY { FUZZY ( Temperature IS HOT AND Humidity IS HUMID ) }
     MEMBER HOT { FUZZY Temperature IS HOT }
     MEMBER HUMID { FUZZY Humidity IS HUMID }
     MEMBER COLD { FUZZY Temperature IS COLD }
     MEMBER CLAMMY { FUZZY ( Temperature IS COLD AND Humidity IS HUMID ) }
 }
```

To calculate the **Degree of Membership** (*DOM*) of **MUGGY** in **day**, we need to calculate the *DOM* of **HOT** in Temperature and **HUMID** in Humidity, and then combine them with the fuzzy **AND** operator.

The following code fragment shows implementation of **day is MUGGY**. For each of the linguistic members of **day** a similar equation needs to be generated.

```
unsigned int day_MUGGY( unsigned int__CRISP)
 {
   return (f_and(Temperature_HOT(__CRISP), Humidity_HUMID(__CRISP)));
 }
```

Even more important is the calculation for **day is MUGGY**. This can be quite straightforward, as the following graph shows.



Temperature/Humidity Graph

At first look, this doesn't seem much different from the evaluation of the two membership functions followed by the execution of the *f_and* function. After all, the *f_and* function is simple, and not computationally intensive. The graphical solution suggests that if the *f_and* evaluation were mixed with the evaluation of the membership function, substantial savings in execution time could result–in the worst case, the execution time would be the same as in the equation above.

Each of the rectangles in the above graph have their own unique computational requirements. The area that has a value of *fuzzy_zero* requires that either *Temperature is less than 60* or *Humidity is less than 75 %.* Similarly, if *Temperature is greater than 80* and *Humidity is greater than 90%* then the result is a *fuzzy_one*.

Even eliminating these areas won't necessarily require computation of both membership functions. In two of the areas in the graph *f_and* produces a minimum of *fuzzy_one* and either a function of *Temperature* or *Humidity*. In these cases, the minimum calculation requires a single membership function.

In my experience, it is very common to combine two linguistic terms and define a new linguistic variable, or find that a fuzzy rule is actually the simple combination of two linguistic variables. The above diagram displays that it is at least possible that calculations combining two linguistic variables may be considerably less complicated than suggested by earlier equations. In much of the current application base, membership functions are some variation on simple trapezoids. The above graphic representation makes calculations in these cases easy.

Some of the better implementation tools using fairly standard compiler technology can now recognize and implement this simplification when appropriate. The resulting execution speed increase can be impressive, even on simple 8 bit microcomputers.

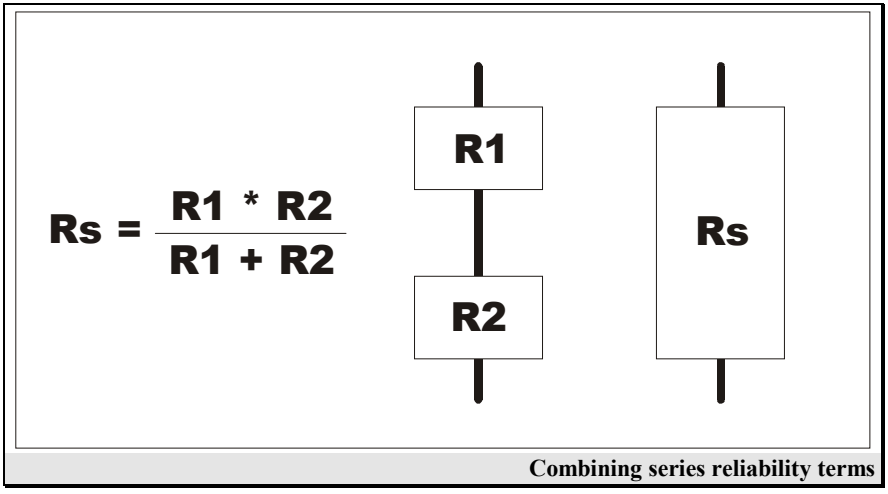
Software reliability and fuzzy logic

Let us look at the lessons we can learn by applying high reliability principles to software development. This approach tends to draw less specific conclusions, but can form the basis for subjective evaluations of competing software designs, and can provide an effective tool for software engineering.

Simple systems are components combined in series and parallel terms. Complex systems result from the combination of simple systems. Real systems are rarely as simple as a few components with easily identified relationships. Most reliability calculations, especially in software, are at best good estimates based on individual component information and some hard data measured from the system.

The math behind all system reliability calculations is based on combining individual components (in software individual instructions or functions) using two basic formulas.

Given two components in a system with (Mean Time Between Failures) *MTBF*'s of *R1* and *R2*, they can be combined into a single component whose reliability is given by the following example.



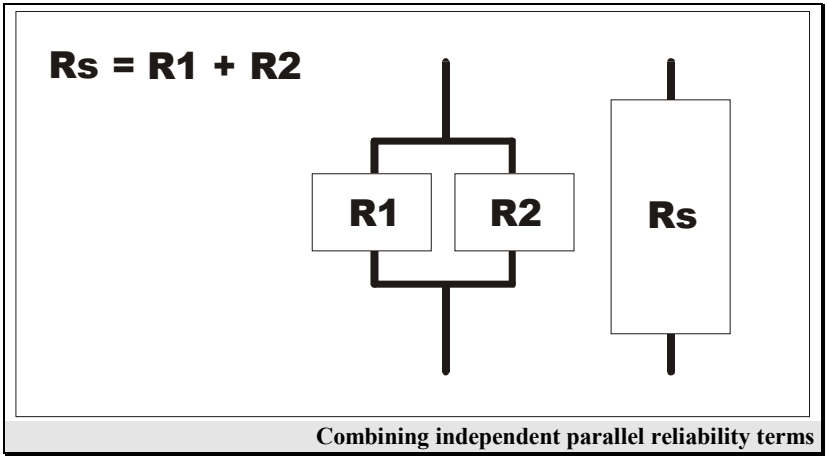
Combining series reliability terms

The reliability *Rs* is indistinguishable from the reliability of the two components *R1* and *R2*. The units used in each of the reliabilities is *time*, usually measured in hours. In a practical system, reliabilities are the combination of series and parallel terms. Although software cannot place actual times on *MTBF* calculations, we can learn a lot about our system if we look at the relative reliability of software structured in different ways.

If two components in a system function independently, and the system can continue to function despite the failure of either component, then we can show the combined system reliability with the following diagram. The reliability *Rs* is indistinguishable from the reliability of the two components *R1* and *R2*.

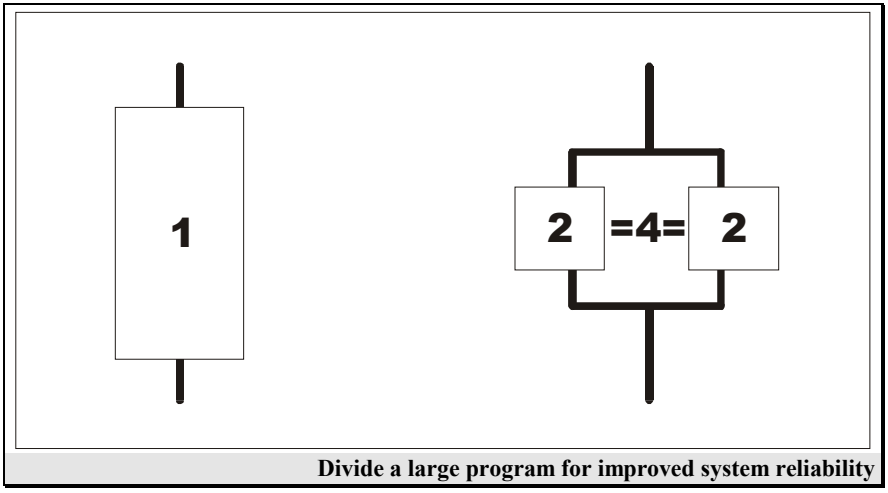
Take a program and give it a dimensionless reliability of unity. Now divide the program into two parts such that each part performs a separate operation. This is often possible, because few programs contain code for a single operation. Re-configure the program to function as two independent tasks. You can then measure the reliability of the resulting two-task system.

Each of these tasks will be half as long as the original, meaning that if our original assumption that the task reliability is a function of the code length is correct, each task will probably fail half as often as the original program.



Combining independent parallel reliability terms

Each task then has a reliability of 2. If the correct operation of each half keeps the original system running, what we have are parallel independent components. Two independent parts, each with a reliability of 2, will improve the software reliability by a factor of 4. There may be some overhead in additional system code, which should be factored in. Even accounting for the additional code, the results are spectacular.



Divide a large program for improved system reliability

Three essential assumptions are necessary to justify the above scenario.

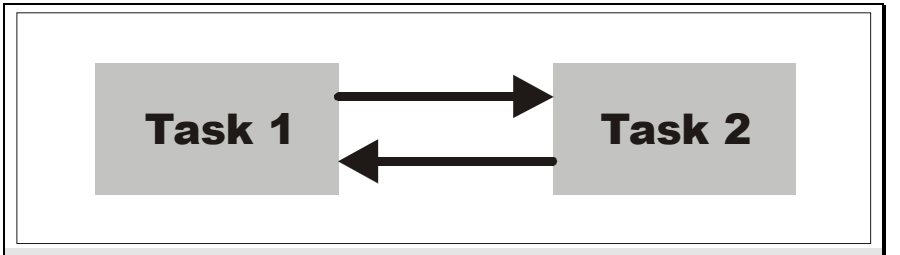
- the program really does have to perform two tasks
- the program's tasks can be divided
- a failure of either task will not cause a system failure

There are many systems that satisfy these conditions.

Here's a practical example involving a high-end product with an unacceptable number of failures. We reorganized the task scheduler from round robin to non-preemptive with many independent tasks. We made each task's execution independent rather than depending on other tasks in the loop. The customer reported failures went essentially to zero, and less than one percent of the code in the system was re-written!

For a moment, assume reliability is essentially the same for all instructions. Assume also the reliability of a single task is essentially a function of the size of the task. In an isolated task this is true, however, in the real world a task takes on arguments and returns results. This adds an assumption that a task can cope with all of the possible arguments presented.

What if the arguments presented to the task could in some way cause the task to fail? Then the arguments themselves would be a part of the reliability of the task. This would indicate that the reliability of a task is a function of the number of arguments presented to it. It also means that anything that is capable of altering the value of the arguments presented to a task is now a part of the series terms in an individual task's reliability.



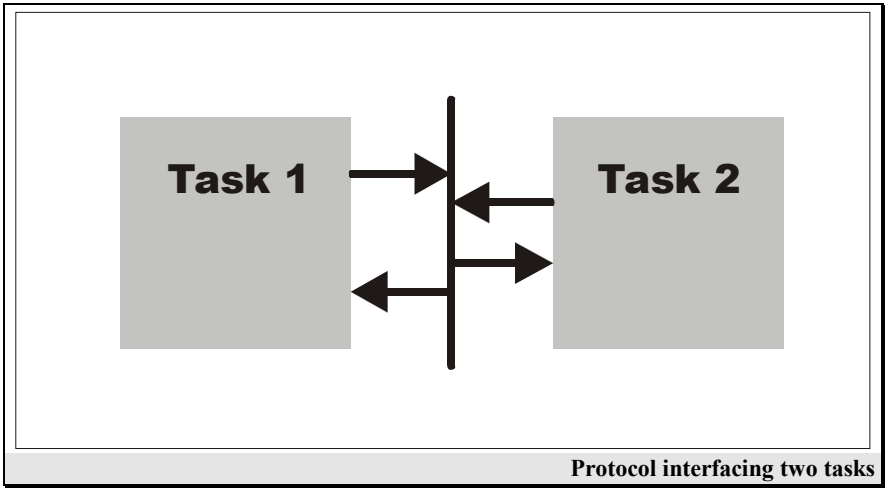
Tasks directly interfacing with each other

Consider the following example: two tasks exist in a system. One calls the second which executes its code and returns a result.

As time passes, it is found the second task is useful for other things, and is called by a third task. The third task requires a minor change that we feel the first task is unlikely to notice. Now, as fate would have it, the first task calls the revised second task and the returned result causes the previously functioning first task to fail.

This suggests that a task's reliability requires its interaction with other tasks be conducted through a well defined interface. In fact, a task should not communicate directly with other tasks at all, but through some abstract protocol. This would mean that a task could then be isolated from its environment; as long as it responds to requests from the protocol it could be implemented in any manner without affecting other tasks making requests of it through the protocol.

The following figure depicts this implementation. The protocol provides the isolation needed to protect the tasks. Each task communicates solely with the protocol, which makes calls to tasks and receives their output. The protocol contains the list of expected responses for a given set of arguments.



Protocol interfacing two tasks

Sound familiar? In implementing fuzzy logic systems the fuzzy logic rules operate independently from each other to the degree that the rules in a block can usually be executed in any order, and the result will still be the same. Each rule is small and may be implemented in a few instructions. Fuzzy logic rules call membership functions through a well-defined interface providing isolation and further parallelism. After a fuzzy logic rule is evaluated it calls *CONSEQUENCE* functions through a well-defined interface.

As in our earlier example of dividing a problem to achieve improved reliability, the fuzzy logic solution naturally breaks a problem into its component parts. There are other ways to visualize the reliability of the system. The focus of the fuzzy logic rules is on a very different level of detail than is the focus of the membership functions.

This reduces a problem's solution to its component parts. Compilers may reassemble the code for effective execution on some target, but at the programmer level the problem is a number of simple tasks.

Without trying, the implementation of a fuzzy logic system naturally follows a coding style that lends itself to producing reliable code. Fuzzy logic is inherently robust, and this is the reason.

Bibliography

There are a number of names that consistently appear in the fuzzy logic literature. In your search for reading material, the following authors have much to offer both in their technical content and their presentation. This list is not exhaustive, but it is a reasonable place to start a literature search in most good libraries.

- David I. Brubaker, *The Huntington Group.*
- Madan M. Gupta, *University of Saskatchewan, Saskatoon*.
- Bart Kosko, *University of Southern California, Los Angeles*.
- Jim Sibigtroth, *Motorola Semiconductor, Austin, Texas*.
- Lotfi Zadeh, *University of California at Berkeley*.
- Hans-Jurgen Zimmermann.

Reading List

Bandemer, Hans, ed., "Some applications of fuzzy set theory in data analysis", 1. Aufl., Leipzig: VEB Deutscher Verlag fur Grundstoffindustrie, c1988, ISBN 3-34-200985-3 (pbk.), (Summaries in English, German and Russian).

Bezdek, James C. and Pal, Sankar K., "Fuzzy models for pattern recognition: methods that search for structures in data", New York: IEEE Press, 1992, ISBN 0-78-030422-5.

Billot, Antoine, "Economic theory of fuzzy equilibria: an axiomatic analysis", Berlin: Springer-Verlag, c1992, ISBN 3-54-054982-X (Berlin), ISBN 0-38-754982-X (New York).

Dubois, Diddier and Prade, Henri, "Possibility Theory, An Approach to Computerized Processing of Uncertainty".

Fedrizzi, Mario; Kacprzyk, Janusz and Roubens, Marc, ed. "Interactive fuzzy optimization", Berlin: Springer-Verlag, c1991, ISBN 0-38-754577-8, ISBN 0-38-754577-8 (U.S.).

Gupta, Madan M. and Sanchez, Elie, ed., "Approximate reasoning in decision analysis", Amsterdam: North-Holland Pub. Co., 1982, ISBN 0-44-486492-X (U.S.).

Gupta, Madan M. Gupta, et. al., ed., "Approximate reasoning in expert systems", Amsterdam: North-Holland, 1985, ISBN 0-44-487808-4 (U.S.).

Gupta, Madan M. and Sanchez, Elie, ed., "Fuzzy information and decision processes", Amsterdam: North-Holland, c1982, ISBN 0-44-486491-1. (Companion vol.: Approximate reasoning in decision analysis).

Janko, Wolfgang H.; Roubens, Marc and Zimmermann, H.-J., ed., "Progress in fuzzy sets and systems", *Proceedings of the Second Joint IFSA-EC and EURO-WGFS Workshop on Progress in Fuzzy Sets in Europe held on April 6-8, 1989, in Vienna, Austria*, Dordrecht ; Boston: Kluwer Academic Publishers, c1990, ISBN 0-79-230730-5.

Kacprzyk, Janusz and Fedrizzi, Mario, ed., "Combining fuzzy imprecision with probabilistic uncertainty in decision making", Berlin: Springer-Verlag, c1988, ISBN 3-54-050005-7 (German), ISBN 0-38-750005-7 (U.S.).

Kacprzyk, Janusz and Yager, Ronald R., ed., "Management decision support systems using fuzzy sets and possibility theory",

Kaufmann, Arnold and Gupta, Madan M., "Introduction to Fuzzy Arithmetic, Theory and Applications", Von Nostrand Reinhold, 1984, ISBN 0-442-23007-9.

Kaufman, Arnold and Gupta, Madan M., "Fuzzy mathematical models in engineering and management science", Amsterdam: North-Holland, c1988, ISBN 0-44-470501-5.

Koln: Verlag TUV Rheinland, c1985, ISBN 3-88-585143-1 (pbk.).Kacprzyk, Janusz and Fedrizzi, Mario, ed.,"Multiperson decision making models using fuzzy sets and possibility theory", Dordrecht: Kluwer Academic Publishers, 1990 ISBN 0-79-230884-0.

Kacprzyk, Janusz, "Multistage decision-making under fuzziness: theory and applications Koln: Verlag TUV Rheinland, 1983, ISBN 3-88-585093-1.

Kacprzyk, J. and S.A. Orlovski, S.A., ed., "Optimization models using fuzzy sets and possibility theory", Dordrecht: D. Reidel: International Institute for applied Systems Analysis, c1987, ISBN 9-02-772492-X.

Kandel, Abraham, "Fuzzy mathematical techniques with applications", Reading, Mass. ; Don Mills, Ont.: Addison-Wesley, c1986, ISBN 0-20-111752-5.

Kandel, Abraham, "Fuzzy techniques in pattern recognition", *A Wiley-Interscience publication* New York ; Toronto: Wiley, c1982, ISBN 0-47-109136-7.

Kandel, Abraham, ed., "Fuzzy expert systems", Boca Raton, FL: CRC Press, c1992, ISBN 0-84-934297-X.

Karwowski, Waldemar and Mital, Anil, ed., "Applications of fuzzy set theory in human factors", *Advances in human factors/ergonomics; v. 6* Amsterdam: Elsevier, 1986, ISBN 0-44-442723-6.

Klir, George J. and Folger, Tina A., "Fuzzy sets, uncertainty, and information", Englewood Cliffs, N.J.: Prentice Hall, c1988, ISBN 0-13-345984-5.

Kosko, Bart, "Neural Networks and Fuzzy Systems, A Dynamical Systems Approach to Machine Intelligence", Prentice Hall, 1992, ISBN 0-13-611435-0.

Kosko, Bart, "Fuzzy Thinking", Hyperion, 1993, ISBN 1-56282-839-8

Kruse, Rudolf and Meyer, Klaus Dieter, ed., "Statistics with vague data", Dordrecht: D. Reidel, c1987, ISBN 9-02-772562-4.

Kruse, R.; Schwecke, E. and Heinsohn, J., "Uncertainty and vagueness in knowledge based systems: numerical methods", Berlin: Springer-Verlag, c1991, ISBN 3-54-054165-9, ISBN 0-38-754165-9 (New York).

Leung, Yee, "Spatial analysis and planning under imprecision", Amsterdam: North Holland, 1988, ISBN 0-44-470390-X (U.S.).

Mamdani, E. H. and Gaines, B. R., ed., "Fuzzy reasoning and its applications", London ; Toronto: Academic Press, 1981, ISBN 0-12-467750-9.

McNeil, D. and Freiberger, P., "Fuzzy Logic", New York: Simon and Schuster, 1993, ISBN 0-671-73843-7

Miyamoto, Sadaaki, "Fuzzy sets in information retrieval and cluster analysis", Dordrecht: Kluwer Academic Publishers, c1990, ISBN 0-79-230721-6.

Negoita, Constantin V. and Ralescu, Dan, "Simulation, knowledge-based computing, and fuzzy statistics", New York: Van Nostrand Reinhold, c1987, ISBN 0-44-226923-4.

Novak, Vilem, "Fuzzy sets and their applications", Bristol: Hilger, 1989, ISBN 0-85-274583-4.

Pal, Sankar K. and Majumder, Dwijesh K. Dutta, "Fuzzy mathematical approach to pattern recognition", *A Halsted Press book*, New York ; Toronto: Wiley, c1986, ISBN 0-47-027463-8.

Pienkowski, Andrew E. K., "Artificial colour perception using fuzzy techniques in digital image processing", Koln: TUV Rheinland, c1989, ISBN 3-88-585640-9.

Schefe, Peter, "On foundations of reasoning with uncertain facts and vague concepts", Hamburg: Fachbereich Informatik, Universitat Hamburg, 1979.

Schmucker, Kurt J., (foreword by Lotfi A. Zadeh), "Fuzzy sets, natural language computations, and risk analysis", Rockville, Md.: Computer Science Press, c1984, ISBN 0-91-489483-8.

Seo, Fumiko and Sakawa, Masatoshi, "Fuzzy multiattribute utility analysis for collective choice", Laxenburg, Austria: International Institute for Applied Systems Analysis, 1987, (Reprinted from IEEE Transactions on systems, man, and cybernetics, volume 15 (1985)).

Terano, Toshiro; Asai, Kiyoji and Sugeno, Michio, "Fuzzy systems theory and its applications", San Diego, CA: Academic Press, 1991, (Translation of: "Fajii shisutemu nyumon"), ISBN 0-12-685245-6.

Verdegay, Jose-Luis and Delgado, Miguel ed., "The interface between artificial intelligence and operations research in fuzzy environment", Koln: Verlag TUV Rheinland, 1989, ISBN 3-88-585702-2.

Yager, Ronald R. and Zadeh, Lotfi A., ed., "An Introduction to Fuzzy Logic Applications in Intelligent Systems", Kluwer Academic Publishers, 1992, ISBN 0-7923-9191-8.

Zadeh, Lotfi and Kacprzyk, Janusz, ed., "Logic for the Management of Uncertainly", John Wiley and Sons, 1992, ISBN 0-471-54799-9.

Zetenyi, Tamas, ed., "Fuzzy sets in psychology", Amsterdam: North-Holland, 1988, ISBN 0-44-470504-X (U.S.).

Zimmermann, Hans-Jurgen; Zadeh, L.A. and Gaines, B.R., ed., "Fuzzy sets and decision analysis", Amsterdam: North-Holland, c1984, ISBN 0-44-486593-4.

Zimmermann, Hans-Jurgen, "Fuzzy set theory and its applications", Boston: Kluwer-Nijhoff Pub., c1985, ISBN 0-89-838150-9.

Zimmermann, H.-J., "Fuzzy set theory-and its applications", 2nd rev. ed., Boston: Kluwer Academic Publishers, c1991, ISBN 0-79-239075-X.

Journals on Fuzzy Logic

"Fuzzy Sets and Systems", North-Holland, Amsterdam, (started in 1978).

Article References

"Fuzzy sets as a basis for a theory of possibility.", Lofti Zadeh, Fuzzy Sets and Sytems I, 3-28

"Designing with Tolerance.", Walter Banks, Embedded Systems Programming June 1990

"Software Reliability", H.Kopetz, Springer_Verlag 1979

"Software Reliability", John D. Musa, Anthony Iannino, Kazuhira Okumoto McGraw-Hill 1990

"Reliability Principles and Practices", S.R. Calabro, McGraw Hill

Brubaker, David I., "Fuzzy-logic Basics: Intuitive Rules Replace Complex Math", EDN Asia, August, 1992, pp. 59-63.

Khan, Emdad and Venkatupuram, Prahlad, "Fuzzy Logic Design Algorithms Using Neural Net Based Learning", Embedded Systems Conference, September, 1992 Santa Clara, CA.

Sibigtroth, James M., "Creating Fuzzy Micros", December, 1991, Emedded Systems Programming, pp. 20-31.

Sibigtroth, James M., "Fuzzy Logics", April, 1992, AI Expert.

Williams, Tom, "Fuzzy Logic Is Anything But Fuzzy", April, 1992, Computer Design, pp. 113-127.

About the authors

Walter Banks is the president of *Byte Craft Limited*, a company specializing in code creation tools for embedded systems. One of these products, **Fuzz-C® Preprocessor for Fuzzy Logic**, adds linguistic variables and fuzzy logic operators to programs written in ***C*** . **Fuzz-C** is implemented as a preprocessor, making it compatible with most ***C*** compilers.

Gordon Hayward is an associate professor in the School of Engineering at the University of Guelph. His research is primarily in the field of instrument design, and his current projects include the design of sensors to monitor environmental contaminants and the use of piezoelectric crystals as chemical sensors. In many applications, the required information can not be measured precisely (flavour is a good example), hence his interest in fuzzy logic.

Appendix

This appendix contains a selection of pages employed as overhead slides used in my talk on **Fuzzy Logic For Control Systems**, at the 1993 Embedded Systems Show in Santa Clara.

In addition to copies of the slides, I've included the actual code for a fuzzy PID controller as well as the block diagram of the PID controller used in Santa Clara.

Example of PID Controller:

```
Int OldError,SumError;
int process(void);

LINGUISTIC Error  TYPE int  MIN -90  MAX 90
 {
 MEMBER LNegative   { -90, -90, -20,  0  }
 MEMBER normal      { -20,   0,  20      }
 MEMBER close       {  -3,   0,   3      }
 MEMBER LPositive   {   0,  20,  90,  90 }
 }

LINGUISTIC DeltaError  TYPE int  MIN -90  MAX 90
 {
 MEMBER Negative    { -90, -90, -10,  0  }
 MEMBER Positive    {   0,  10,  90,  90 }
 }

LINGUISTIC SumError  TYPE int  MIN -90  MAX 90
 {
 MEMBER LNeg    { -90, -90,  -5,  0  }
 MEMBER LPos    {   0,   5,  90,  90 }
 }

CONSEQUENCE  ManVar TYPE int   MIN -20  MAX 20 DEFUZZ cg
{
    MEMBER  LNegative  {  -18 }
    MEMBER  SNegative  {   -6 }
    MEMBER  SPositive  {    6 }
    MEMBER  LPositive  {   18 }
}

FUZZY pid
  {
   IF Error IS LNegative THEN ManVar IS LPositive

   IF Error IS LPositive THEN ManVar IS LNegative

   IF Error IS normal AND DeltaError IS Positive
     THEN ManVar IS SNegative

   IF Error IS normal AND DeltaError IS Negative
     THEN ManVar IS SPositive
```

```
    IF Error IS close AND SumError IS LPos
      THEN ManVar IS SNegative

    IF Error IS close AND SumError IS LNeg
      THEN ManVar IS SPositive

   }

void main (void)
   {
    while(1)
      {
        OldError = Error;
        Error = Setpoint - Process();
        DeltaError = Error - OldError;
        SumError := SumError + Error;
        pid();
       }
    }
```

Example of Code for a fuzzy PID controller:

```
#include "fuzzc.h"
char __IDOM[2];
 Int OldError,SumError;
 int process(void);
/*  LINGUISTIC Error  TYPE int  MIN -90  MAX 90 */
/*   { */
int  Error ;
/*   MEMBER LNegative   { -90, -90, -20,  0  } */
/*

  1-| .............
    | .            .
    | .             .
    | .              .
  0-| .               ..................
    ----------------------------------
    -90     -45       0      45      90

*/
char Error_LNegative (int __CRISP)
  {
    {
    if (__CRISP <= -20) return(255);
    else
      {
      if (__CRISP <= 0)
        return(( - __CRISP * 12) + 7);
      else
        return(0);
      }
    }
  }
/*   MEMBER normal      { -20,   0,  20      } */
/*

  1-|                 .
    |                . .
    |               .   .
    |              .     .
  0-| ..............      ..............
    ----------------------------------
    -90     -45       0      45      90

*/
char Error_normal (int __CRISP)
  {
  if (__CRISP < -20) return(0);
  else
    {
    if (__CRISP <= 0) return(((__CRISP + 20) * 12) + 7);
    else
      {
        {
        if (__CRISP <= 20)
          return((( + 20 - __CRISP) * 12) + 7);
        else
          return(0);
        }
      }
    }
  }
/*   MEMBER close       {  -3,   0,   3      } */
/*
```

```
  1-|                 .
    |                 .
    |                 .
    |                ..
  0-| ................................
     ----------------------------------
    -90     -45       0      45      90

*/
char Error_close (int __CRISP)
  {
  if (__CRISP < -3) return(0);
  else
    {
    if (__CRISP <= 0) return((__CRISP + 3) * 85);
    else
      {
        {
        if (__CRISP <= 3)
          return(( + 3 - __CRISP) * 85);
        else
          return(0);
        }
      }
    }
  }
/*   MEMBER LPositive   {   0,  20,  90,  90 } */
/*

  1-|                    ..............
    |                    .            .
    |                   .             .
    |                  .              .
  0-| ..................               .
     ----------------------------------
    -90     -45       0      45      90

*/
char Error_LPositive (int __CRISP)
  {
  if (__CRISP < 0) return(0);
  else
    {
    if (__CRISP <= 20) return((__CRISP * 12) + 7);
    else
      {
        return(255);
      }
    }
  }
/*   } */
/*

     Fuzzy Sets for Error

  1-| ..............   .  ..............
    | .             . ... .             .
    | .              . . .              .
    | .             . ... .             .
  0-| ..............  ..  ..............
     ----------------------------------
    -90     -45       0      45      90


*/
/*  LINGUISTIC DeltaError  TYPE int  MIN -90  MAX 90 */
/*   { */
int  DeltaError ;
```

```
/*   MEMBER Negative    { -90, -90, -10,  0  } */
/*

  1-| ..............
    | .              .
    | .              .
    | .               .
  0-| .                ..................
    ------------------------------------
    -90     -45       0      45      90

*/
char DeltaError_Negative (int __CRISP)
  {
    {
    if (__CRISP <= -10) return(255);
    else
      {
      if (__CRISP <= 0)
        return(( - __CRISP * 25) + 2);
      else
        return(0);
      }
    }
  }
/*   MEMBER Positive    {   0,  10,  90,  90 } */
/*

  1-|                   .................
    |                   .               .
    |                   .               .
    |                  .                .
  0-| .................                 .
    ------------------------------------
    -90     -45       0      45      90

*/
char DeltaError_Positive (int __CRISP)
  {
  if (__CRISP < 0) return(0);
  else
    {
    if (__CRISP <= 10) return((__CRISP * 25) + 2);
    else
      {
        return(255);
      }
    }
  }
/*   } */
/*

     Fuzzy Sets for DeltaError

  1-| ..............  .................
    | .              . .               .
    | .              . .               .
    | .               .                .
  0-| ..................................
    ------------------------------------
    -90     -45       0      45      90


*/
/*  LINGUISTIC SumError  TYPE int  MIN -90  MAX 90 */
/*   { */
int  SumError ;
/*   MEMBER LNeg    { -90, -90,  -5,  0  } */
```

```
/*

  1-| .................
    | .               .
    | .                .
    | .                .
  0-| .                ..................
     ----------------------------------
    -90     -45       0      45      90

*/
char SumError_LNeg (int __CRISP)
  {
    {
    if (__CRISP <= -5) return(255);
    else
      {
      if (__CRISP <= 0)
        return( - __CRISP * 51);
      else
        return(0);
      }
    }
  }
/*   MEMBER LPos    {   0,   5,  90,  90 } */
/*

  1-|                  .................
    |                  .               .
    |                  .               .
    |                  .               .
  0-| .................                .
     ----------------------------------
    -90     -45       0      45      90

*/
char SumError_LPos (int __CRISP)
  {
  if (__CRISP < 0) return(0);
  else
    {
    if (__CRISP <= 5) return(__CRISP * 51);
    else
      {
        return(255);
      }
    }
  }
/*   } */
/*

     Fuzzy Sets for SumError

  1-| ................................
    | .               ..               .
    | .                .               .
    | .                .               .
  0-| ................................
     ----------------------------------
    -90     -45       0      45      90


*/
/*  CONSEQUENCE  ManVar TYPE int   MIN -20  MAX 20 DEFUZZ cg */
/*  { */
int   ManVar ;
   long fa_@ConsName, fc_@ConsName;
/*      MEMBER  LNegative  {  -18 } */
```

```
/*

  1-|  .
    |  .
    |  .
    |  .
  0-| .*.............................
    ---------------------------------
     -20     -10       0      10      20


*/
void ManVar_LNegative (int  __DOM)
  {
     fc_@ConsName += @ConsVol;
     fa_@ConsName += (@ConsVol * (@ConsPoint));
  }
/*      MEMBER  SNegative  {   -6 } */
/*

  1-|            .
    |            .
    |            .
    |            .
  0-| ...........*.....................
    ---------------------------------
     -20     -10       0      10      20


*/
void ManVar_SNegative (int  __DOM)
  {
     fc_@ConsName += @ConsVol;
     fa_@ConsName += (@ConsVol * (@ConsPoint));
  }
/*      MEMBER  SPositive  {    6 } */
/*

  1-|                     .
    |                     .
    |                     .
    |                     .
  0-| ..................*.............
    ---------------------------------
     -20     -10       0      10      20


*/
void ManVar_SPositive (int  __DOM)
  {
     fc_@ConsName += @ConsVol;
     fa_@ConsName += (@ConsVol * (@ConsPoint));
  }
/*      MEMBER  LPositive  {   18 } */
/*

  1-|                                .
    |                                .
    |                                .
    |                                .
  0-| ..............................*..
    ---------------------------------
     -20     -10       0      10      20


*/
void ManVar_LPositive (int  __DOM)
  {
```

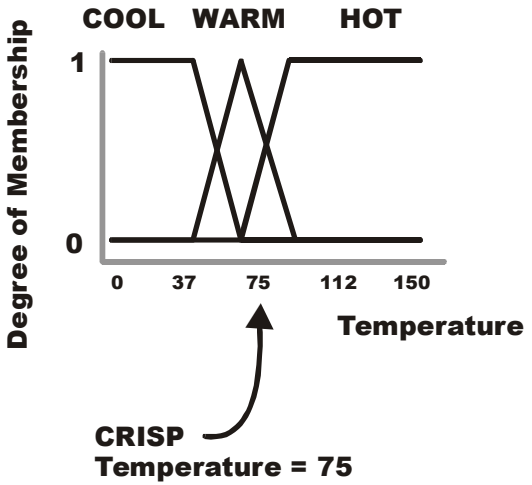
```
      fc_@ConsName += @ConsVol;
      fa_@ConsName += (@ConsVol * (@ConsPoint));
   }
/*  } */
/*  FUZZY pid */
void pid (void)
   {
      fa_@ConsName = 0;
      fc_@ConsName = 0;
/*    { */
/*      IF Error IS LNegative THEN ManVar IS LPositive */
   ManVar_LPositive( Error_LNegative(Error)  );
/*      IF Error IS LPositive THEN ManVar IS LNegative */
   ManVar_LNegative( Error_LPositive(Error)  );
/*      IF Error IS normal AND DeltaError IS Positive */
/*        THEN ManVar IS SNegative */
   __IDOM[1] = Error_normal(Error) ;
   __IDOM[0] = DeltaError_Positive(DeltaError) ;
   __IDOM[0] = F_AND(__IDOM[1],__IDOM[0]);
   ManVar_SNegative( __IDOM[0] );
/*      IF Error IS normal AND DeltaError IS Negative */
/*        THEN ManVar IS SPositive */
   __IDOM[1] = Error_normal(Error) ;
   __IDOM[0] = DeltaError_Negative(DeltaError) ;
   __IDOM[0] = F_AND(__IDOM[1],__IDOM[0]);
   ManVar_SPositive( __IDOM[0] );
/*      IF Error IS close AND SumError IS LPos */
/*        THEN ManVar IS SNegative */
   __IDOM[1] = Error_close(Error) ;
   __IDOM[0] = SumError_LPos(SumError) ;
   __IDOM[0] = F_AND(__IDOM[1],__IDOM[0]);
   ManVar_SNegative( __IDOM[0] );
/*      IF Error IS close AND SumError IS LNeg */
/*        THEN ManVar IS SPositive */
   __IDOM[1] = Error_close(Error) ;
   __IDOM[0] = SumError_LNeg(SumError) ;
   __IDOM[0] = F_AND(__IDOM[1],__IDOM[0]);
   ManVar_SPositive( __IDOM[0] );
/*    } */
      @ConsName = fa_@ConsName / fc_@ConsName;
    }
 void main (void)
    {
     while(1)
       {
         OldError = Error;
         Error = Setpoint - Process();
         DeltaError = Error - OldError;
         SumError := SumError + Error;
         pid();
        }
     }
```

Fuzzy Logic Presentation Slides

These slides accompanied the presentation adapted for the first part of this book, Fuzzy logic implementation on embedded microcomputers. The quality of some slides reflects their origin as transparencies.


Linguistic
Variables
Minestrone
Soup Mix
Directions:
1. Empty contents into saucepan;
add 4½ cups (1 L) cold water.
2. Bring to a boil, stirring constantly.
3. Reduce heat; partially cover
and simmer for 15 minutes,
stirring occasionally.
4 to 6 servings, 4½ cups (1 L)

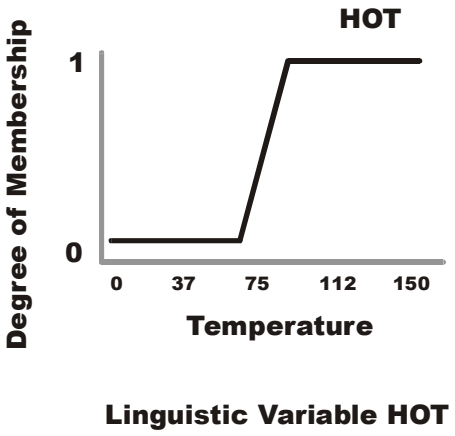
Linguistic Variables



Given a crisp temperature of 75:

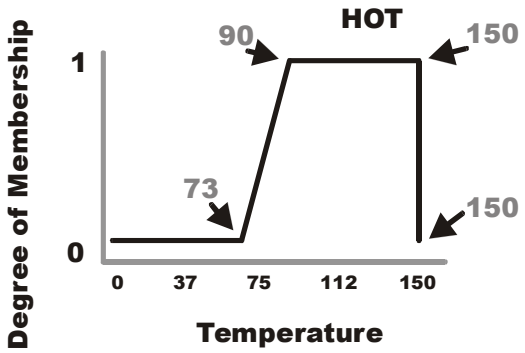
- Degree of Membership(COOL) = 0.0
- Degree of Membership(WARM) = 0.7
- Degree of Membership(HOT) = 0.23

Linguistic Variables



Linguistic Variable HOT

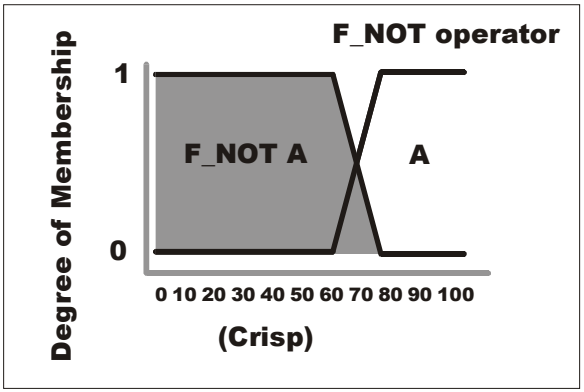
Linguistic Variables

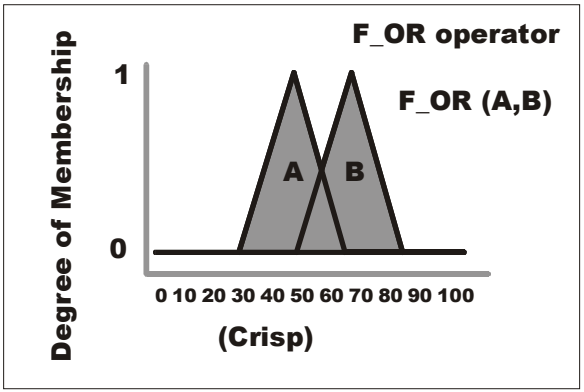


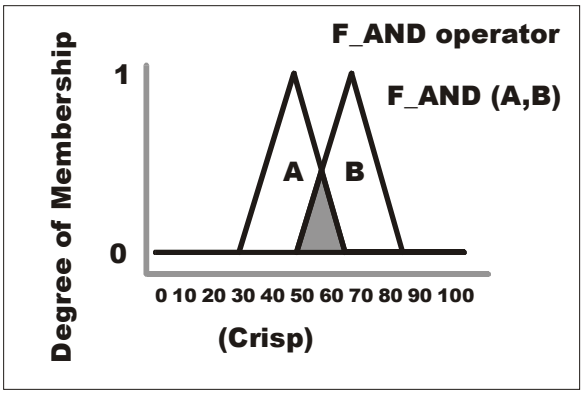
Linguistic Variable HOT

```
LINGUISTIC Temperature TYPE unsigned int MIN 0 MAX 150
   {
     MEMBER HOT {73, 90, 150, 150}
   }
```

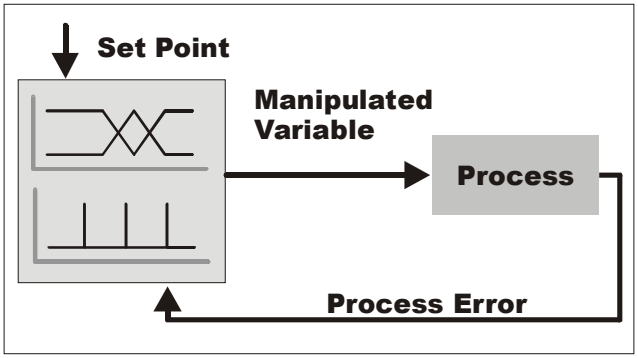
Fuzzy Operators





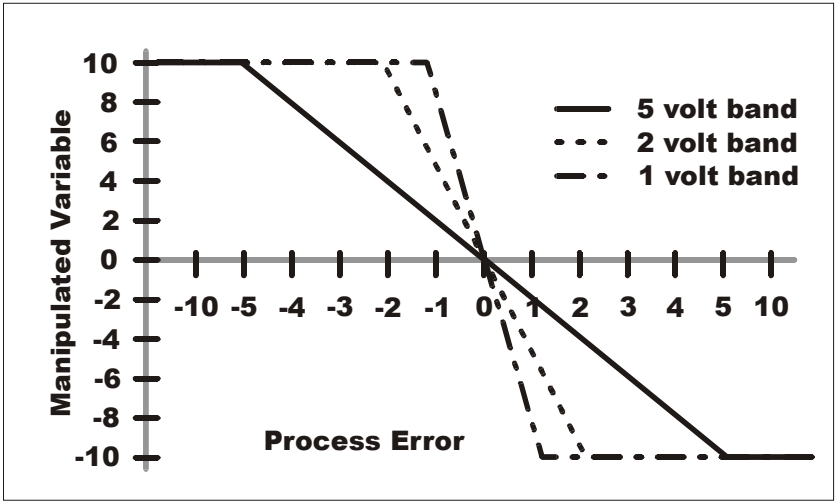


Fuzzy Proportional Contoller

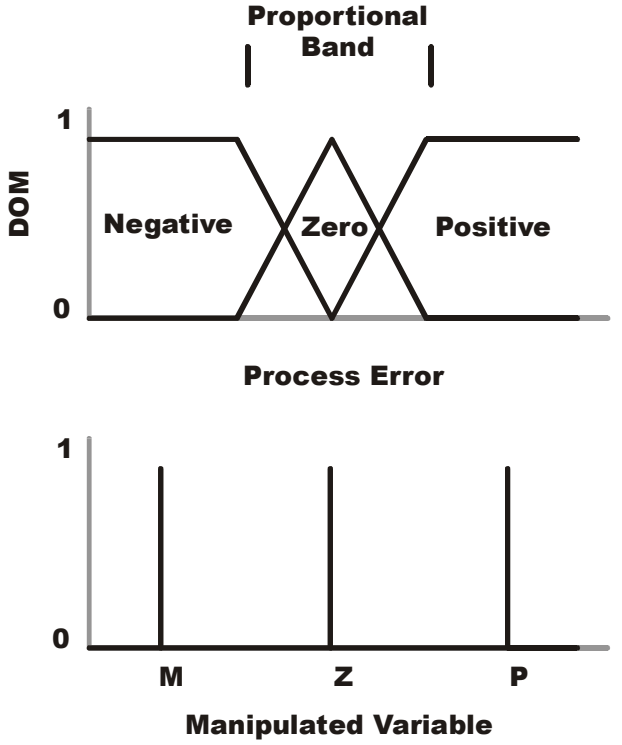


Process Proportional Band

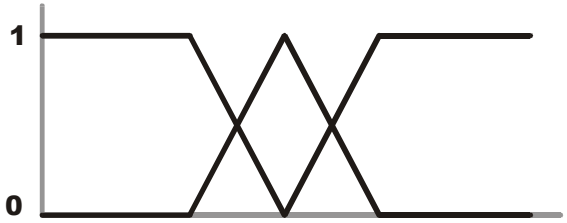
- Range of error to give full-scale proportional output



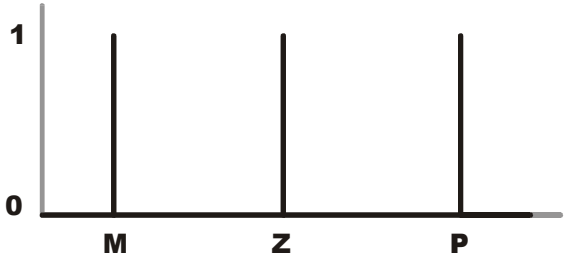
Fuzzy Proportional Controller



Fuzzy Proportional Controller



Process Error

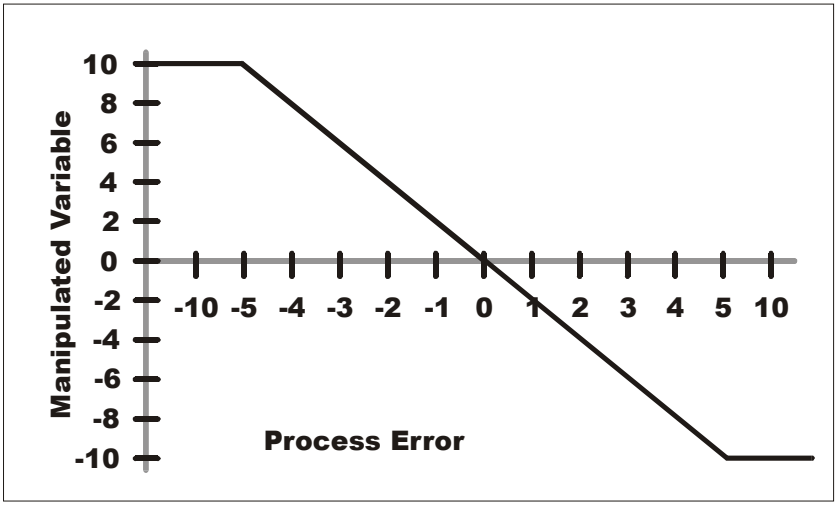


Rules:

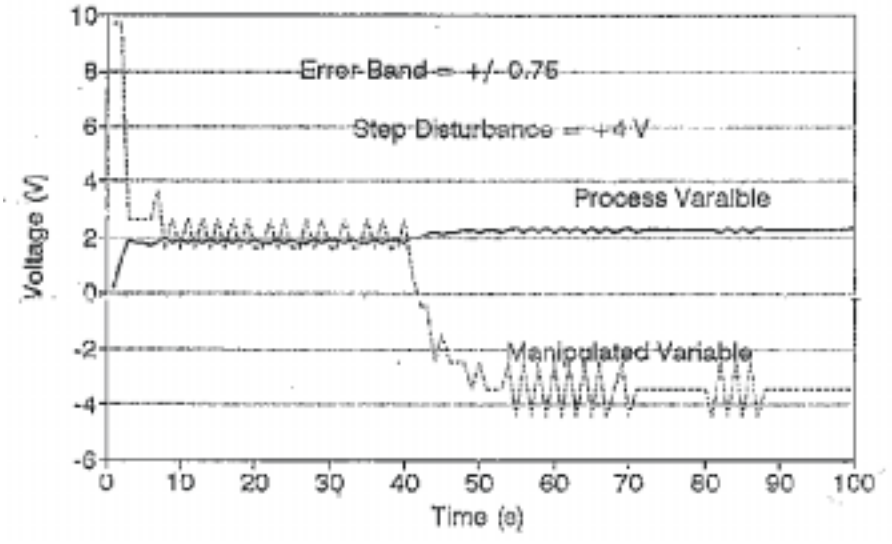
- if pe is POSITIVE THEN mv IS M
- if pe is NEGATIVE THEN mv IS P
- if pe is ZERO THEN mv is Z

Fuzzy Proportional Controller

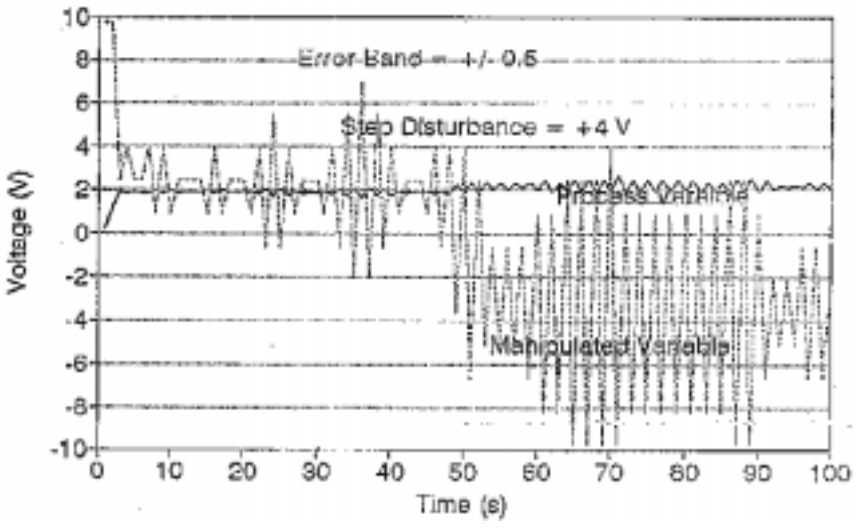
- Transfer Function



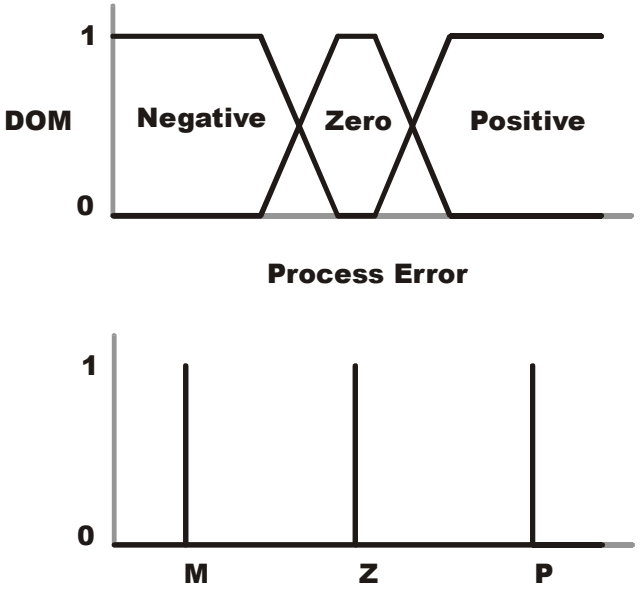
Fuzzy Proportional Controller Performance



Fuzzy Proportional Controller Performance

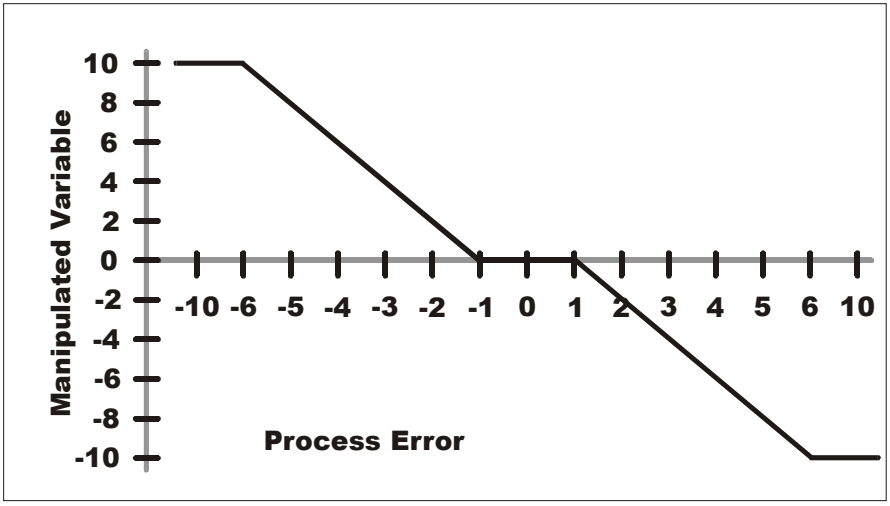


Fuzzy Proportional Controller

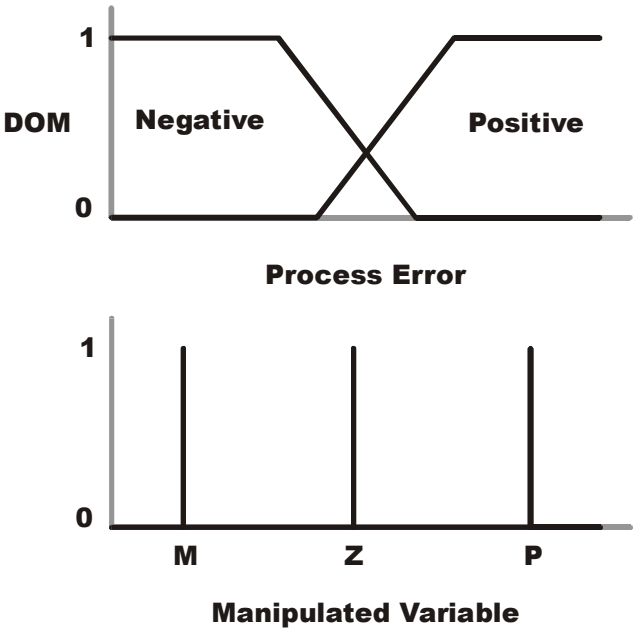


Fuzzy Proportional Controller

- Transfer Function

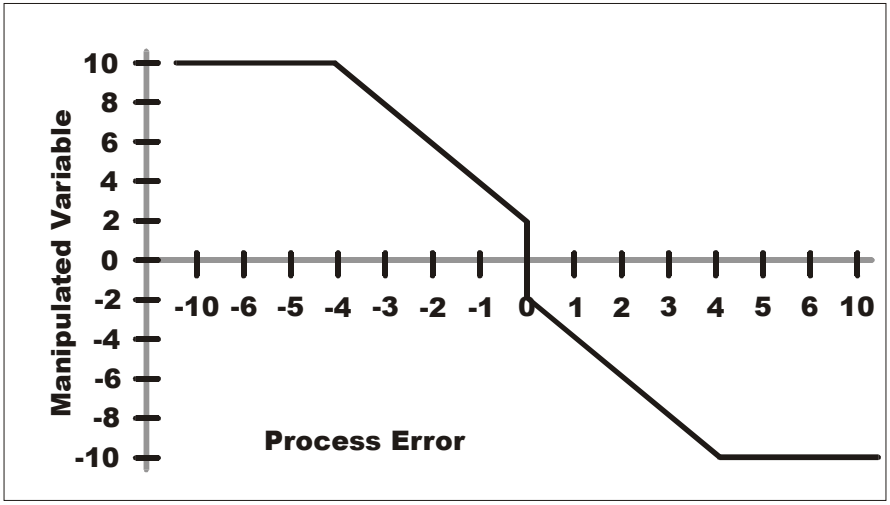


Fuzzy Proportional Controller



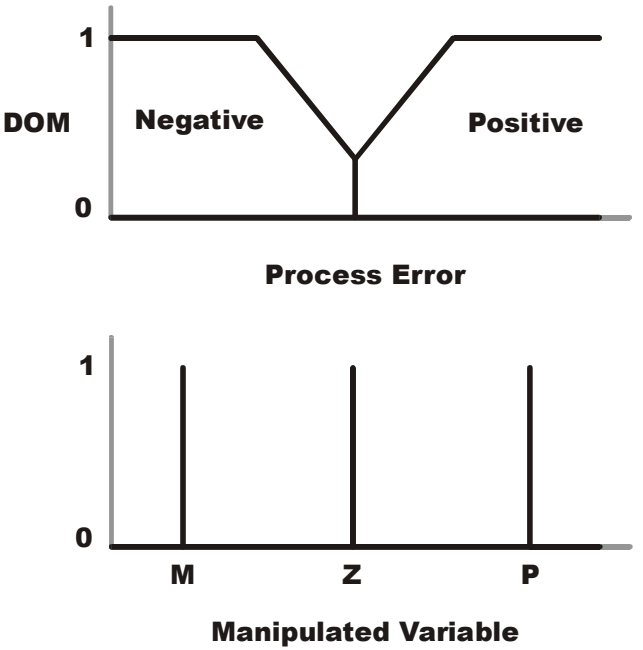
Fuzzy Proportional Controller

- Transfer Function

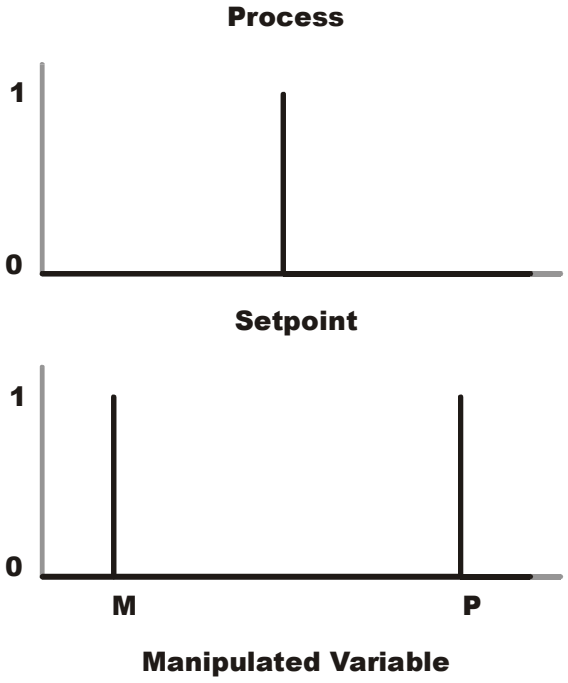


Fuzzy Proportional Controller

- Computationally less intensive



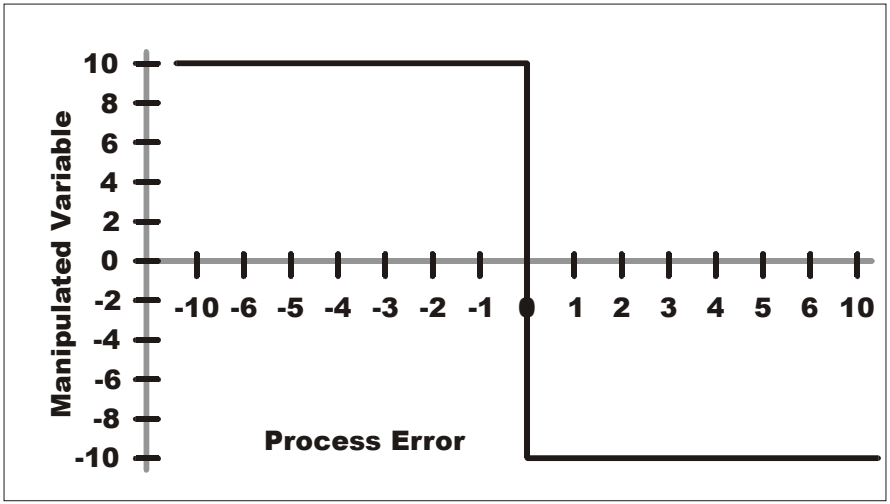
Bang Bang Controller



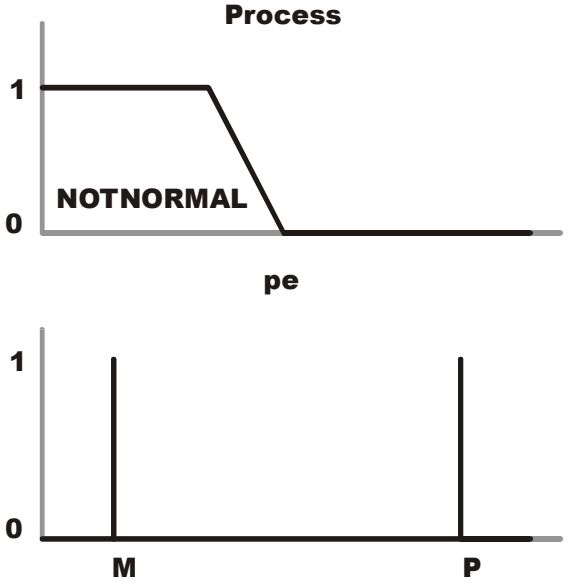
IF (process < setpoint)
  THEN mv is P
  ELSE mv is M

Bang Bang Controller

- Transfer Function



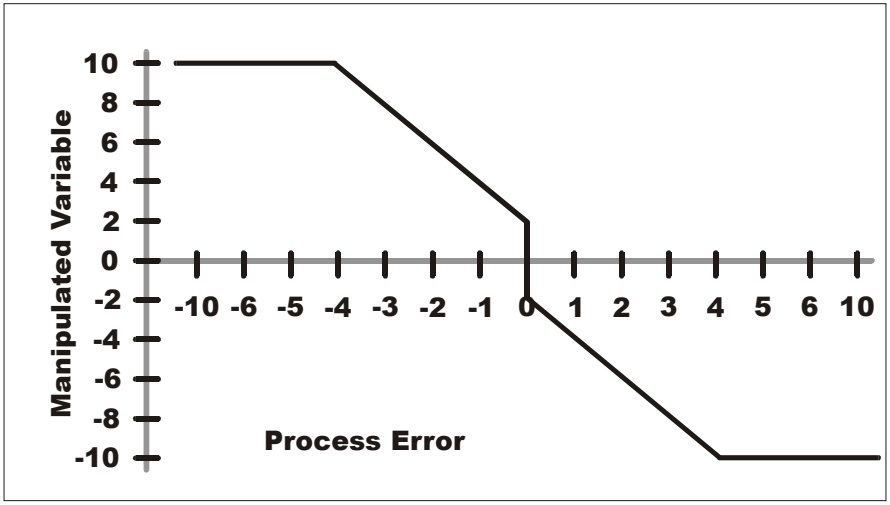
Fuzzy Bang Bang



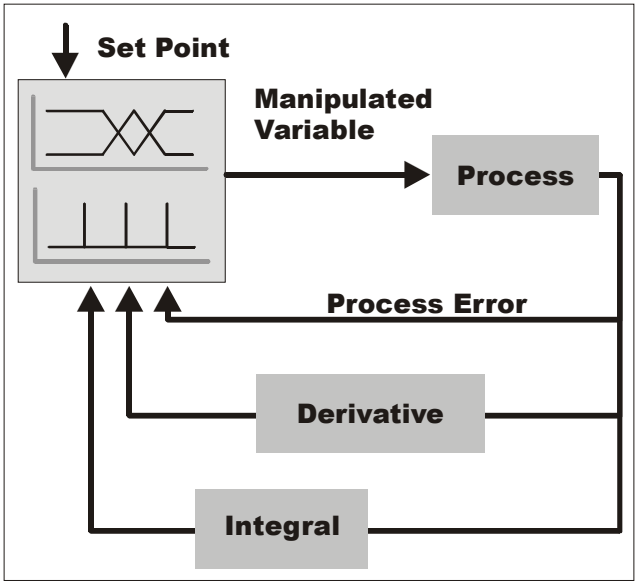
RULES:
IF pe IS NOTNORMAL
  THEN mv is P
  ELSE mv is M

Fuzzy Bang Bang

- Transfer Function



Fuzzy PID Controller



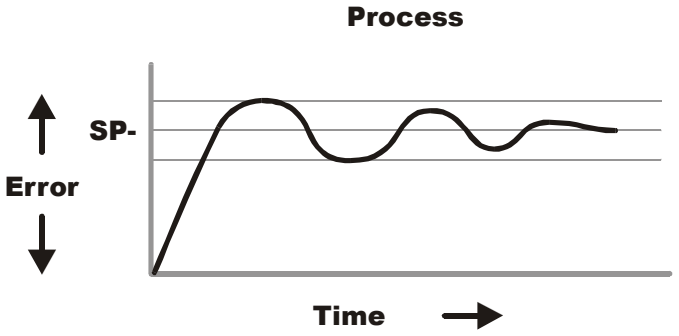
Fuzzy PID Controller

- Classical PID
  - Manipulated variable is the sum of three terms:

$$mv = (pe \times K1) + \frac{d\,pe}{d\,t} \times K2 + (\Sigma\, pe \times K3)$$

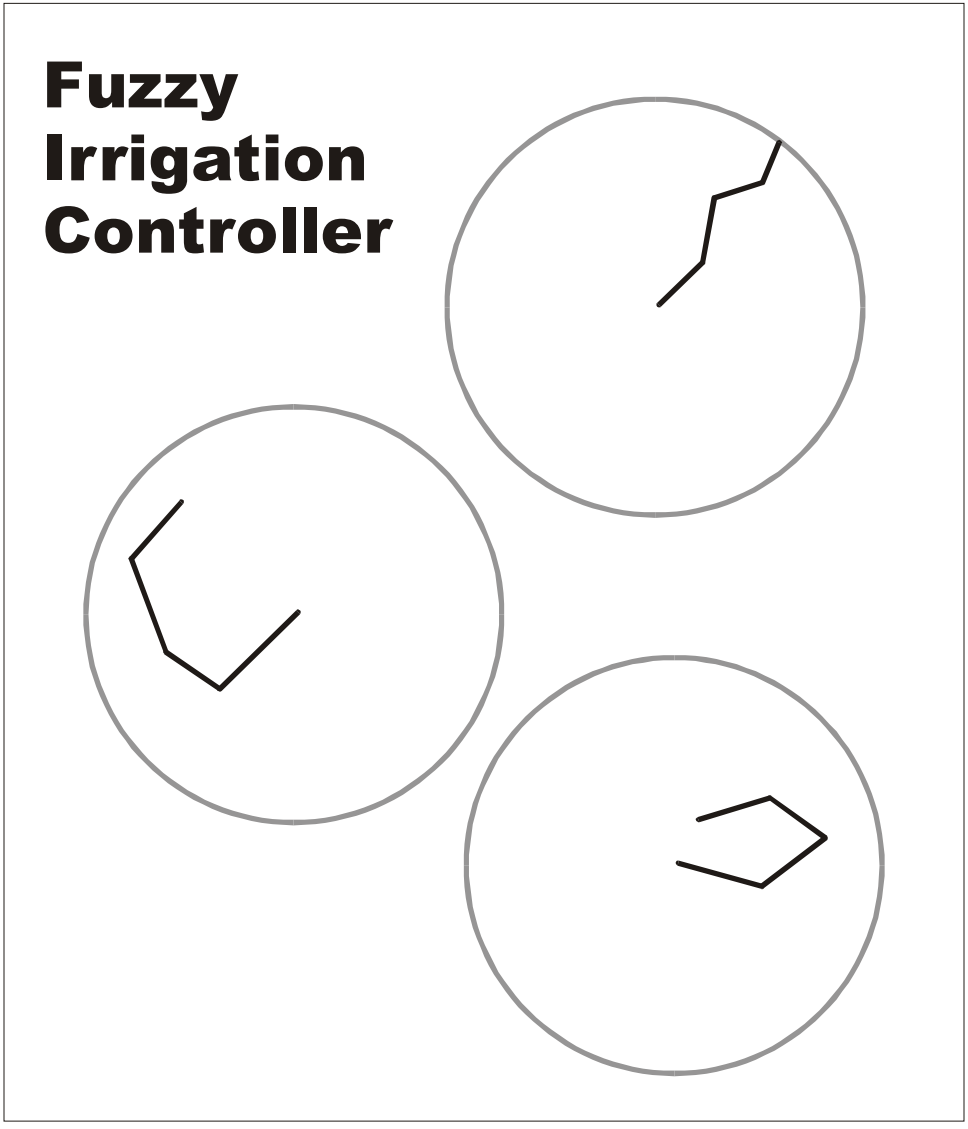
Fuzzy PID Controller

- Break problem into separate control zones.
- Solve each part of the problem individually.



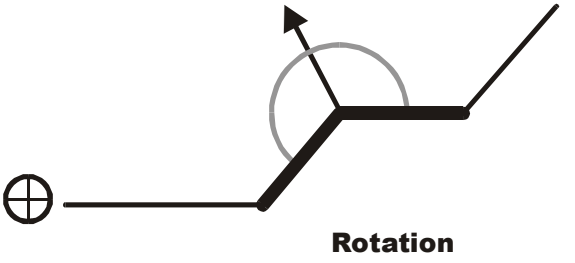
Fuzzy PID Controller

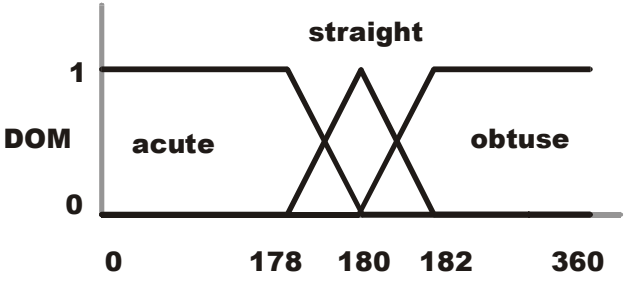
- Separate rules for
  - Error term
  - Derivative Term
  - Integral Term


Fuzzy
Irrigation
Controller

Fuzzy Irrigation Controller

- Making the rules





Fuzzy Irrigation Controller

- Each node has its own rules

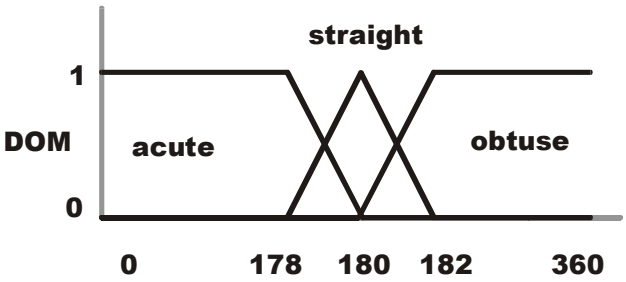
IF angle IS straight
THEN speed IS same
IF angle IS acute
THEN speed IS slowdown
IF angle IS obtuse
THEN speed IS speedup

Fuzzy Irrigation Controller

- Differential Control
  - speedup
  - slowdown
- CONSEQUENCE functions may be non-linear

Fuzzy Irrigation Controller

- Normalized control values
  - Degree of membership range is two degrees



The Fuzzy Advantage

- Normalized number system
- Natural smooth transition between different strategies
- Focus on problem solution, not problem analysis
- Works well on conventional embedded microprocessors
- Can easily be combined with conventional software

Index

-A-

arguments, 16

-C-

CONSEQUENCE functions, 17
crisp, 1, 4, 5

-D-

data types, 4
Degree of Membership, 9

-F-

f_and, 8, 10, 11
f_not, 8
f_or, 8
fuzzy logic rule, 17
fuzzy_one, 11
fuzzy_zero, 11

-I-

interface, 16

-L-

linguistic variables, 1, 5

-M-

Mean Time Between Failures, 13
membership functions, 10, 11, 17

-O-

operators, 5, 8

-R-

reliability, 13

-T-

tasks, 14
trapezoid, 11

Byte Craft Limited CATALOG

05/2001

Code Development Systems

Byte Craft Limited specializes in embedded systems software development tools for single-chip microcontrollers. Byte Craft Limited was the first company to develop a C compiler for the Motorola 68HC05 and the National Semiconductor COP8™. Our compilers and related development tools are now being used by a wide range of design engineers and manufacturers in areas of Commerce, Industry, Education, and Government.

MPC

Supports all Microchip PIC 12x/14x/16x/17x families, 8K and Flash parts
Named address space supports variable grouping
Works with Microchip's PICMASTER, ICE 2000 emulator, MPLAB-SIM simulator, Advanced Transdata, Tech-Tools Mathias, Clearview, iSystem
Supports setting configuration fuses through C
Demo at www.bytecraft.com/impc.html

C6808

Supports all 68HC08 variants
Supports LOCAL memory reuse, SPECIAL memory through software
Supports 6808 extended addressing, instructions
Support for symbolic debugging with many emulators including Motorola MMDS08 and MMEVS08, and the Ashling CT68HC08
Supports setting Mask Option Register through C
Demo at www.bytecraft.com/i08.html

for DOS or Windows COP8C

Supports the Feature Family, and SGR/SGE
Supports LOCAL memory reuse, SPECIAL memory through software
Supports SREG memory management
Support for symbolic debugging with emulators including MetaLink
Supports setting configuration fuses through C
Demo at www.bytecraft.com/icop.html

for DOS or Windows SXC

Supports all SX variants, including SX48 and SX52
Supports LOCAL memory reuse, SPECIAL memory through software
Supports virtual device drivers within C
Data types include bit, bits, char, short, int, int8/16/24/32, long, float and fixed point
Support for assembly source-level debugging with Parallax SX-Key
Demo at www.bytecraft.com/isxc.html

C6805

Supports all 68HC05 variants
Supports LOCAL memory reuse, SPECIAL memory through software
Support for symbolic debugging with many emulators including MMDS05, MMEVS, and Metalink iceMASTER
E6805 available to support Motorola EVM, EVS
Supports setting Mask Option Register through C
Demo at www.bytecraft.com/i05.html

Z8C

Supports all Zilog Z8 and Z8+ variants
Supports instruction set variants C94, C95, HALT, MUL, STOP, WAIT
Supports processor-specific instructions DI, EI, HALT, NOP, RCF, SCF, STOP, WAIT, WDT, WDH
Generates information required for source-level debugging
Demo at www.bytecraft.com/iz8c.html

C38

Supports all MELPS740 variants, including 7600 series, M509xx, M371xx, M374xx and M38xxx
Supports MUL, 7600
Supports processor-specific instructions BRK, CLC, CLD, CLI, CLT, CLV, NOP, PHA, PLA, PLP, ROL, ROR, RRF, SEC, SED, SEI, SET, STP, WIT
Allows direct access to AC, X, Y, CC registers
Demo at www.bytecraft.com/ic38.html

Fuzz-C™

Transforms fuzzy logic to plain C; call between C and fuzzy functions
Accepts fuzzy logic rules, membership functions and consequence functions
Standard defuzzification methods provided; add new defuzzification methods easily
Includes plots of membership and consequence functions in generated comments
Works with all Code Development Systems

Features

Both DOS and Windows versions include an **Integrated Development Environment**. The DOS IDE provides source-level error reporting. The Windows IDE maintains projects, gives access to online help, and can control third-party tools.

The compilers generate tight, fast, and efficient executables, as well as listing files that match the original C source to the code generated. Several optional reports (symbol information, nesting level, register contents) can appear in the listing file.

Header files describe each processor derivative. **#pragma** statements configure the compiler for available interrupts, memory resources, ports, and configuration registers. Convenient **#define**s make your programs portable between members of a processor family.

C extensions include: **bit** and **bits** data types, binary constants, **case** statement extensions, direct register access in C, embedded assembly, initialization control, direct variable placement, interrupt support in C.

Two forms of linking are available: **Absolute Code Mode** links library modules into the executable during compilation. The **BClink linker** uses a more traditional linker command file and object files. Either route provides optimization at final code generation.

You can include **Macro Assembler** instructions within C code, or as separate source files. Embedded assembly code can call C functions and access C variables directly. You can also pass arguments to and from assembly code.

Availability

Byte Craft Limited products are available world-wide, both directly from Byte Craft Limited and through our distributors. Demonstration versions of the Code Development System are available.

For more information, see **www.bytecraft.com**.

Upgrade Policy

Registered customers receive free upgrades and technical support for the first year. All other **registered users** may purchase major releases for a fraction of the full cost. Along with our version upgrades, Byte Craft Limited remains committed to maintaining a high level of technical support.

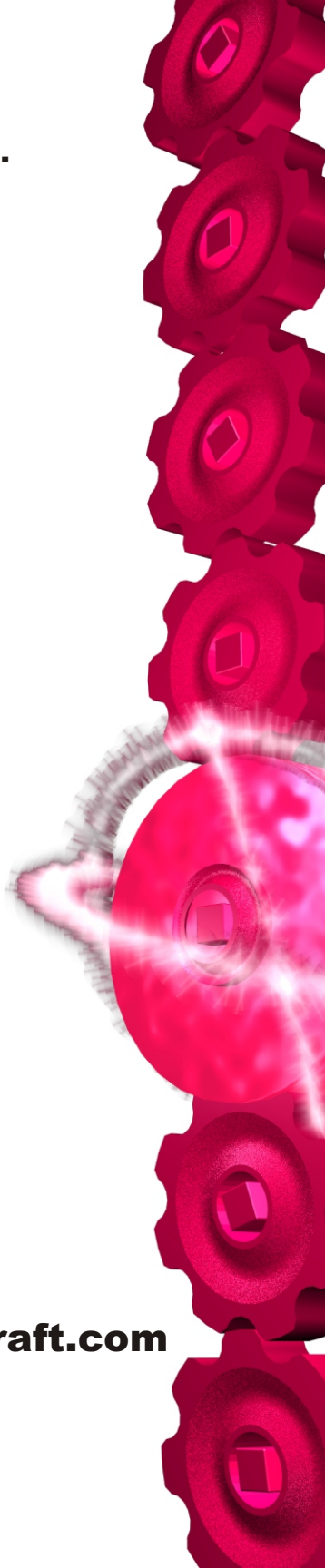
BYTE CRAFT LIMITED

Byte Craft Limited
A2-490 Dutton Drive
Waterloo, Ontario
Canada • N2L 6H7
phone: 519-888-6911
fax: 519-746-6751

info@bytecraft.com
www.bytecraft.com

Byte Craft Limited Publishing brings Embedded Systems information and technology to developers.

Byte Craft Limited has more than twenty years' experience in Embedded Systems, and significant experience in publishing and document management. We are devoting our efforts to putting knowledge into developers' hands.



Byte Craft Limited
A2-490 Dutton Drive
Waterloo, Ontario, Canada
N2L 6H7
phone: +1 519.888.6911
fax : +1 519.746.6751
<info@bytecraft.com>

www.bytecraft.com