# A Fuzzy Logic Control Project For a Real-time Microprocessor Laboratory

**Richard E. Pfile**
**Indiana University-Purdue University at Indianapolis**
**Greg Smith**
**Cummins Engine Company**

## Abstract

An advanced microprocessor course was revised to use the new Motorola M-Core 32-bit RISC processor. A series of laboratories were developed for the course that implements fuzzy logic control of an inverted pendulum. The interface hardware was intentionally kept very simple to force the interface and control functions to be implemented in software. The laboratory provides a platform for exercising many microprocessor/control concepts including using multiple interrupts, reading incremental encoders, implementing PWM control techniques, using an internal timer to calculate rates of change, and interfacing to a commercial fuzzy logic engine.

## I. Introduction

A series of laboratory experiments was added to a microprocessor class in which students balance an inverted pendulum using fuzzy logic control. The problem provides a platform for exercising many microprocessor/control concepts including the using multiple interrupts, directly reading incremental encoders, implementing PWM control techniques, using an internal timer to calculate rates of change, and interfacing to a commercial fuzzy logic engine.

Many Electrical Engineering Technology graduates are employed by automation companies where they are required to develop solutions to computer-based automatic control problems. Technology program curriculums typically do offer courses that provide significant depth in solving classical control systems problems, but graduates working in the automation field still need tools to solve control problems. Fuzzy logic is an effective control tool that can be readily implemented in a technology program.[1] Students with a fuzzy logic background can solve many control problems as long as they know what the expected behavior is for various inputs. In addition students get experience reading incremental encoders and applying pulse-width-modulation; two concepts commonly used in the automation field. Students also enjoy solving the problem because it is readily verifiable and has a dynamic output. Funding for equipment funding to develop this course was provided by the National Science Foundation grant number 9750497.

## II. Background

An inverted-pendulum consists of a mass and a length. It is connected to a base around which it can rotate freely. The base contains a motor, which is used to drive the pendulum to the left or right. The inverted pendulum is naturally unstable. Its tendency is to fall over to the left or right, rather than remain upright much like trying to balance a sharpened pencil on your finger.

Balancing a pendulum requires determining the arm and base position, calculating the arm and base velocity including the direction of the velocity, and generating control signals to a motor that moves the arm along a base in a manner that keeps the arm balanced. A controller trying to balance a pendulum must consider the position and velocity of the arm and base and make appropriate decisions to keep the pendulum upright.

## III. System Hardware

We designed a circuit board to interface the pendulum I/O to a microcontroller. The interface hardware was intentionally kept simple (no hardware PWM or encoder decoding) so students would be forced to write software to determine position information directly from encoder outputs and generate the PWM signals to control the motor inputs.

The encoders on the pendulum arm and base are high-resolution, providing 1000 pulses per revolution. The high-resolution encoders enable the software to accurately determine velocity by providing multiple pulses between each of the control updates whenever the arm or base is moving.

The microcontroller uses pulse-width modulation control to incrementally control the inverted pendulum. A circuit was designed that interfaces microcontroller outputs to power switching transistors to drive the pendulum base dc motor in either direction. Two microcontroller output bits are used to control the motor; one bit for clockwise rotation and another to drive the motor counterclockwise. If neither bit is active, no power is delivered to the motor. A hardware lockout prevents trying to run the motor in both directions at the same time. The interface board also converts the TTL level outputs of the encoders to 3-volt logic used by the microcontroller. A RC lowpass filter was also designed into the interface board to remove high frequency motor noise that was present in the system.

The processor used for this system had to be sufficiently fast to decode encoder signals, execute the fuzzy control software, and generate pulse-width-modulation outputs to control the base motor. The processor used is the new 32-bit Motorola M-Core RISC processor. Our processors had a cycle time of .0625 microseconds with most instructions completed in one machine cycle. The processor interrupt latency is under two microseconds.

## IV. Software

The machine architecture is optimized for high-level languages such as C. C language was used for most of the programming with exception of the interrupt service routines, which are programmed in assembly language.

- PWM Control Techniques

When using PWM the fundamental PWM interval must be determined.[2] Factors that will influence the interval are the inertia of the plant being controlled and the granularity required in the control steps. For this problem it was decided that ten levels of control would be sufficient. The PWM period is broken down into ten intervals and power could be applied during all ten of the intervals for maximum power delivery. It could be turned off during all of the intervals for no power delivery, or be applied in any number of the intervals between 0 and 10 for variable power delivery. The PWM period was set at 10 milliseconds so that control levels could be modified at the rate of 100 times per second. This update rate was conservatively fast considering the dynamics of the arm.

A software flag was set whenever the direction of the control required a change. When this occurred, the current PWM interval was abandoned, a new count was calculated and control was applied in the opposite direction.

- Computing Position and Velocity

Incremental encoders with two output phases provide the pendulum arm and base position information. Software calculates the distance moved by counting pulses. The direction of the movement is determined by examining the phase of the pulses in the pulse train.[3] If the pulse in the second phase is high when the first phase transitions from low to high the movement is clockwise, and if the pulse is low the movement is counter clockwise. Velocity is easily calculated by comparing positions at the current PWM control time and the previous PWM interval time. Since the PWM interval is fixed at 10 milliseconds, velocity is known. High-resolution encoders are needed to accurately calculate velocity. To have a range of velocity readings, it is important that several pulses be detected between PWM update times whenever the base is moving.

The phase is determined using software. The software is setup so that both edges of pulses from one phase generate interrupts and the other phase is examined to determine the direction. It is important that the processor handle interrupts quickly enough to be able to check the phase of the pulse in the other train before it switches to another state. In addition, the software must not lose pulses when the direction changes. If not properly handled, the pulse count will be inaccurate if the first edge of the pulse is detected, and the encoder changes direction before the trailing edge of the pulse is detected. The first edge would have added a count, but the software will not subtract the count when the encoder changes direction since

another rising would not be seen until the next pulse. Software flags are set so the trailing edge of the pulse can subtract a count if a direction reversal has taken place.

- Fuzzy Logic Control

A commercial fuzzy logic code generator is used to generate a fuzzy control output from the base and arm encoder information. The generator outputs C code that is linked into the project file. The code generator is Windows-based and extremely intuitive. Students are able to start using the generator with just a few pages of instructions. Using the commercial code generator allowed students to focus on solving the control problem rather than having to spend excess time writing fuzzy logic generation code.

There are three basic steps in a fuzzy logic control system: fuzzification, rule evaluation, and defuzzification.[4] The fuzzification step maps real crisp inputs into a fuzzy membership function to determine the degree of truth in each function. The result of this mapping is a fuzzy input that is applied to the rule evaluation function. The rule evaluation process uses logic to combine the fuzzy inputs to produce fuzzy outputs that themselves are mapped to membership functions. The various fuzzy outputs from the membership functions are combined (typically using a center of gravity approach) to produce a real crisp output.

The fuzzy generator begins with a series of dialogs in which students define the input variables and fuzzy functions. The variable types are specified as either 8 or16-bit unsigned or double and the range of the variable is specified. At runtime variables can be passed to the fuzzy functions as function arguments or they can be global variables, the choice is selected in a dialog box during development. If global variables are used, the names are generated and shown in a header file produced by the code generator.

The fuzzy generator provides the option of entering custom fuzzification membership functions by allowing the user to specify up to 16 data points to define the function. In addition, the more traditional triangular or trapezoidal functions can be specified. For the laboratory, triangular shaped functions were used. See figure 1. A Windows-based visual editor allows the user to drag points of the membership functions to new locations to quickly make modifications and test the resulting code. Depending on the number of membership functions specified, a set of default names are provided such as: low, medium or high. The user is also permitted to enter unique names. The fuzzy generator code can store the functions as look-up tables (runs faster, requires more memory) or it can calculate outputs at run-time.

The evaluation of fuzzy rules is called fuzzy rule inference. The fuzzy inputs are inputted into the rule evaluation block which uses if-then logic to determine the contributions each of the output membership functions will make in determining a final crisp output.

Rules are entered using a Rule Block editor. All rule combinations are entered into the editor with a default Degree of Support equal to 1. Rules can later be deleted if they are not needed or the degree of support can be reduced from one. AND and OR operators are available for fuzzy inference. The AND operation is used to select the minimum from the linguistic

variables and the OR operation is used to select the maximum from the list of linguistic variables. One of the rules used is:

IF ArmAnglePositiveHigh AND ArmVelocityPositiveHigh THEN MotorSpeedNegativeHigh

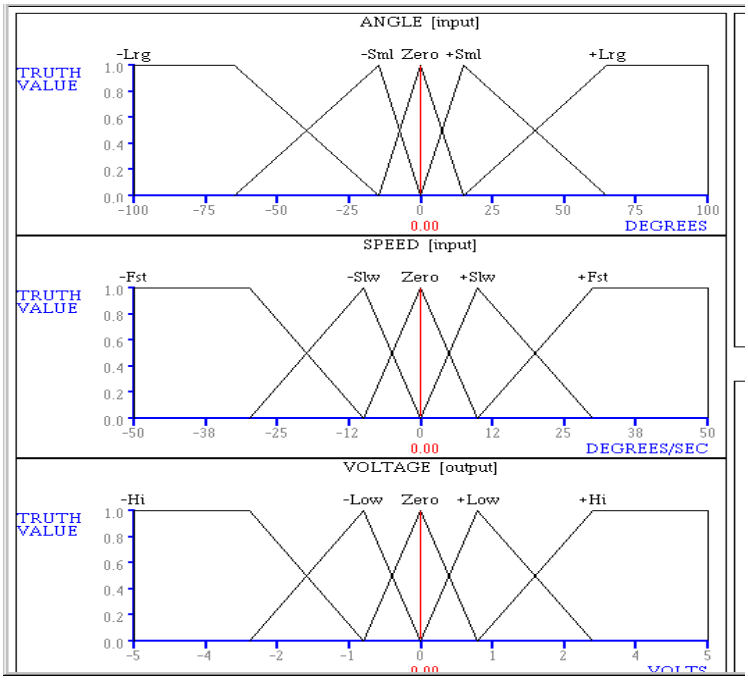


Figure 1. Fuzzy Membership Functions

In the above example the lowest degree of membership from either *ArmAnglePositiveHigh* and *ArmVelocityPositiveHigh* would set the membership level for the output variable *MotorSpeedPositiveHigh*. A typical system will have many such rules whose outputs will be combined to determine the degree of membership functions for the output variables. Combining the outputs is called aggregation. A typical system will use an OR operator to combine outputs, and the maximum membership values from all rules will be used in the final step of determining the degree of membership in an output fuzzy function. The final step is to generate a crisp output by combining the results from all of the output membership functions to generate a single crisp output that is outputted to control the direction and power delivered to the motor.

## V. Conclusion

Students who for the most part had a very limited background in feedback control theory were able to solve a significant control problem using fuzzy logic. In the laboratory experience, students were given a tool that could be applied to a variety of control problems that they are likely to encounter in the design of factory automation equipment and systems.

[1] J. Sibigtroth, "Creating Fuzzy Micros", Embedded Systems Programming, Vol. 4, No. 12, pp. 20-34, December 1991.
[2] Peter Spasov, MIcrocontroller Technology the 68HC11, Prentice Hall, Columbus, Ohio, 1999.
[3] Robert Bateson, Introduction to Control System Technology, Prentice Hall, Englewood Cliffs, New Jersey, 1996.
[4] Ted Heske, Fuzzy Logic for Real World Design, Annabooks, San Diago, CA, 1996

Richard E. Pfile is a professor of Electrical Engineering Technology at IUPUI. He received his B.S. from the University of Louisville and his M.Eng. from the University of Michigan. He has won the Outstanding Teaching Award and has received Teaching Excellence awards from the School of Engineering and Technology at IUPUI. He teaches courses in microprocessor systems, computer networks and digital signal processing. He has 16 years of teaching experience and eight years of industrial experience, including three years as a systems engineer.