Real-Time Implementation of a Fuzzy Logic Speed Control System for Error Minimization in Two-Axis Motion Control

David McAfee
University Undergraduate Fellow, 1993-1994
Texas A&M University
Department of Mechanical Engineering

APPROVED BY:

Fellows Advisor R. Langari

Honors Director Dale T. Knobel

ABSTRACT

In manufacturing systems, there exists a trade-off between speed and accuracy. The faster a system is allowed to move, the higher the rate of production, however this is at the expense of decreased quality. This project attempts to develop a hardware/software digital control system to optimize this relationship for a two-axis manufacturing platform using a fuzzy logic speed control algorithm.

The fuzzy logic speed controller maintains accuracy by reducing the tracking speed around sharp corners. This action has been compared to the thought process of a person driving a car down a winding road. When the road begins to turn sharply, the driver reduces the speed to maintain control of the vehicle, but as the road begins to straighten out, the driver once again increases the speed of the automobile. Just like a human operator, the Self Paced Fuzzy Tracking Controller has the ability to optimize tracking speed based on the current conditions of the path it is following.

The development of the control system was based on the following need statement: to develop an accurate control system for a two-dimensional positioning system, using the Motorola 6811 microcontroller, a software implementation of the Self Paced Fuzzy Tracking Controller, and the Aerotech X-Y table. The system was required to perform several functions, including path storage and generation, current motor status sensing, local position control, fuzzy logic speed control, and actuator signal conversion. Many of these functions were incorporated into hardware components, which resulted in a multi-processor control system with one dedicated to each motor, and a Motorola 6811 microcontroller to coordinate data transfer.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

I. Introduction

In manufacturing systems, there exists a trade-off between speed and accuracy. The faster a system is allowed to move, the higher the rate of production, however this is at the expense of decreased quality. This project attempts to develop a hardware/software digital control system to optimize this relationship for a two-axis manufacturing platform using a fuzzy logic speed control algorithm. The purpose of the fuzzy logic controller is to maximize the speed that the system is allowed to move while at the same time minimizing the incurred contouring error.

Following a basic discussion of the fundamentals of fuzzy logic, Section III gives an introduction to the theory of operation of the fuzzy logic speed controller which was incorporated into the control code. Section IV provides an overview of the existing system hardware which was used in the experiment, including the two-axis positioning system, the electric motors and the position encoder system, and the Motorola 6811 microcontroller used to run the control software.

Section V presents an overview of the needs and requirements of the system, including detailed discussions of the systems which must be included in the final design and methods to realize these systems. The following two sections deal with the specific details involved in the development of the hardware and software system used to drive the motors. Finally, Section VIII gives a summary of the results of the experiment.

II. Introduction to Fuzzy Logic

Is a temperature of 80 °F warm or hot? Within the confines of Boolean logic, where a proposition is either true or false, it would be a difficult to come to a satisfactory conclusion. On the other hand, fuzzy logic is multi-valued, where instead of being totally true or false, it is possible to be partially true and partially false at the same time. So an acceptable reply to a fuzzy logic system would be "some of both". To gain a better understanding of fuzzy systems, one must be familiar with the basic concepts associated with fuzzy logic. Fuzzy logic processing proceeds in three basic steps: fuzzification, rule evaluation, and defuzzification.

The first step in fuzzy logic processing is fuzzification. In this step, the input variables are transformed from "crisp" inputs to "fuzzy" inputs. A "crisp" number is one that would occur in the real world. For example, a "crisp" temperature would be 80 °F. First, fuzzy labels must be assigned to cover the range of all possible values applicable to a system variable, also known as the universe of discourse for a variable. Figure 1 shows a possible fuzzy label assignment for a temperature input variable.

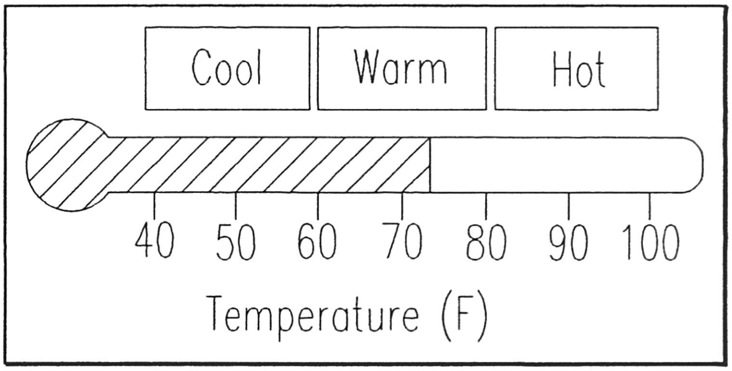


Figure 1: Possible Fuzzy Labels for a Temperature Input Variable

Next, membership functions are established to give numerical meaning to each fuzzy label. A membership function defines the range of input values that correspond to a label. Fuzzy membership functions do not define cut-off points where a label applies fully on one side and not at all on the other, as is the case in Boolean logic. Instead they define a region where a particular fuzzy label gradually changes from being fully applicable to completely inapplicable. A set of possible membership functions for the temperature system is shown in Figure 2.

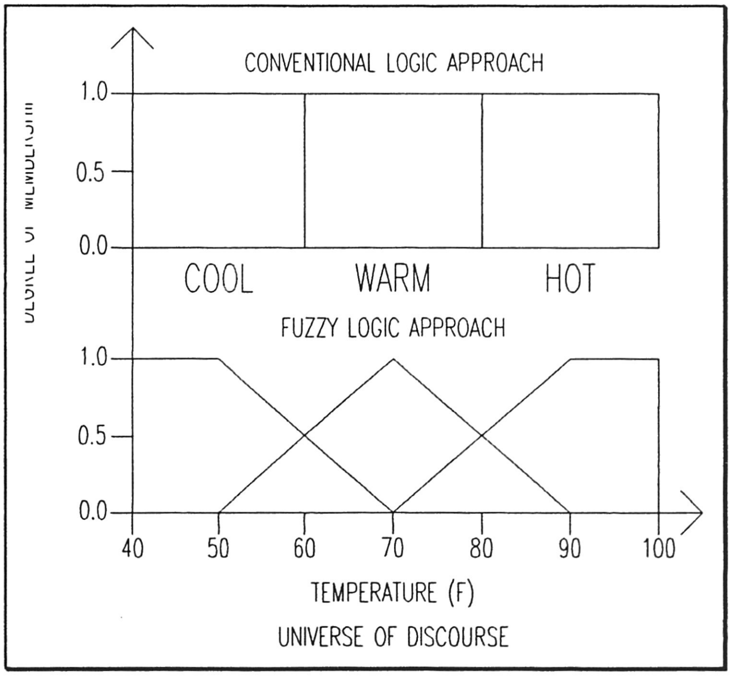


Figure 2: Conventional vs. Fuzzy Set Definition

A membership function for a given label specifies the degree of membership for each possible input value. For example, from the above figure it can be seen that a temperature of 80 °F has a degree of membership of 0.5 in the fuzzy label **WARM**, and a degree of membership of 0.5 in the fuzzy label **HOT**.

The second step in fuzzy logic processing is rule evaluation. In this step, fuzzy logic rules are evaluated to determine what control action should be taken for a given set of inputs. Just as in Boolean logic, fuzzy decisions are made by evaluating sets of if-then statements. However, instead of each statement being absolutely true or false, it may be true or false to varying degrees. The syntax for fuzzy logic rules is:

IF antecedent 1 AND antecedent 2... THEN consequent 1 AND consequent 2...

where the antecedents and consequents compare a variable name with a fuzzy label as in:

antecedent 1 = Temperature is HOT

The numerical value associated with each antecedent is defined as the degree of membership of the input variable in the fuzzy label specified in the antecedent statement. Since multiple rules may be true at the same time, the fuzzy logic processor must determine the strength of each rule by looking at he value of the antecedents. If there is an "AND" statement connecting two antecedents, then the fuzzy processor will use the minimum value of the two antecedents as the rule strength. Once a value for rule strength has been assigned to each rule, the next step is to determine the fuzzy output. This is accomplished by comparing the rule strength of all rules that specify the same output action. Intuitively, if multiple rules apply, the one that is the most true will dominate the output. The multiple rules are combined to determine a single fuzzy output for each fuzzy label. This process is repeated for every membership function for output variables specified in the antecedents of fuzzy rules. For example, if Temperature were an output variable for the control system, there would be a single output from the rule evaluation step for the fuzzy labels COOL, WARM, and HOT.

In the final step of fuzzy logic processing, known as defuzzification, all of the values obtained from rule evaluation are transformed into a single value. This process can be thought of as transforming the fuzzy outputs to a single "crisp" output for the system. The degree of membership value for a fuzzy label is mapped into its membership function for the variable. This process is repeated for all fuzzy labels, and a weighted average technique is used to determine the "crisp" output of the system.

III. Self Paced Fuzzy Tracking Controller

The objective of the Self Paced Fuzzy Tracking Controller is to maintain contouring accuracy while minimizing the time required to complete a given path. The Self Paced Fuzzy Tracking Controller maintains accuracy by reducing the tracking speed around sharp corners. This action has been compared to the thought process of a person driving a car down a winding road. When the road begins to turn sharply, the driver reduces the speed to maintain control of the vehicle, but as the road begins to straighten out, the driver once again increases the speed of the automobile. Just like a human operator, the Self Paced Fuzzy Tracking Controller has the ability to optimize tracking speed based on the current conditions of the path it is following.

Input variables to the fuzzy tracking controller include the current tracking speed of the motors (v), the current curvature of the path (c), and the change in curvature (cc), where:

$$cc = (\textit{previewed curvature - current curvature})$$

The output of each rule is in terms of a relative change in velocity, and the velocity for the next step is determined by:

$$v(n+1) = v(n) + cv(n)$$

where $v(n+1)$ is the new velocity of the motors, $v(n)$ is the previous velocity, and $cv(n)$ is the change in velocity as determined by the Self Paced Fuzzy Tracking Controller. The smaller the current curvature of the path, the faster the tracking speed may be set. Thus, as the curvature becomes smaller, the preview distance for calculating previewed curvature should be increased to allow sufficient time to plan the tracking velocity.

The control rules for the Self Paced Fuzzy Tracking Controller are derived from a human's driving behavior. For example:

if ***v*** is ***S*** and ***c*** is ***M*** and ***cc*** is ***NS*** then ***cv*** is ***PS***

This control law states that if the path is slowly straightening out (*cc* is *NS*) then slightly increase the speed (*cv* is *PS*). Twelve fuzzy logic rules are developed for the Self Paced Fuzzy Tracking Controller which cover the entire range of the system's operation. For computational efficiency, the rules are divided into two sub-sets based on whether the curvature of the path is increasing or decreasing. From this division, only six of the twelve rules need to be evaluated for a given situation.

IV. Existing Hardware

From the beginning of the project, two pieces of hardware are specified to be used in the final design and implementation of the system: the Aerotech X-Y table, and the Motorola 6811 central processing unit.

A. Positioning Table

The X-Y positioning table consists of two stages with the top stage mounted on top of and perpendicular to the bottom stage. A schematic diagram of the two-axis positioning system is shown in Figure 3.

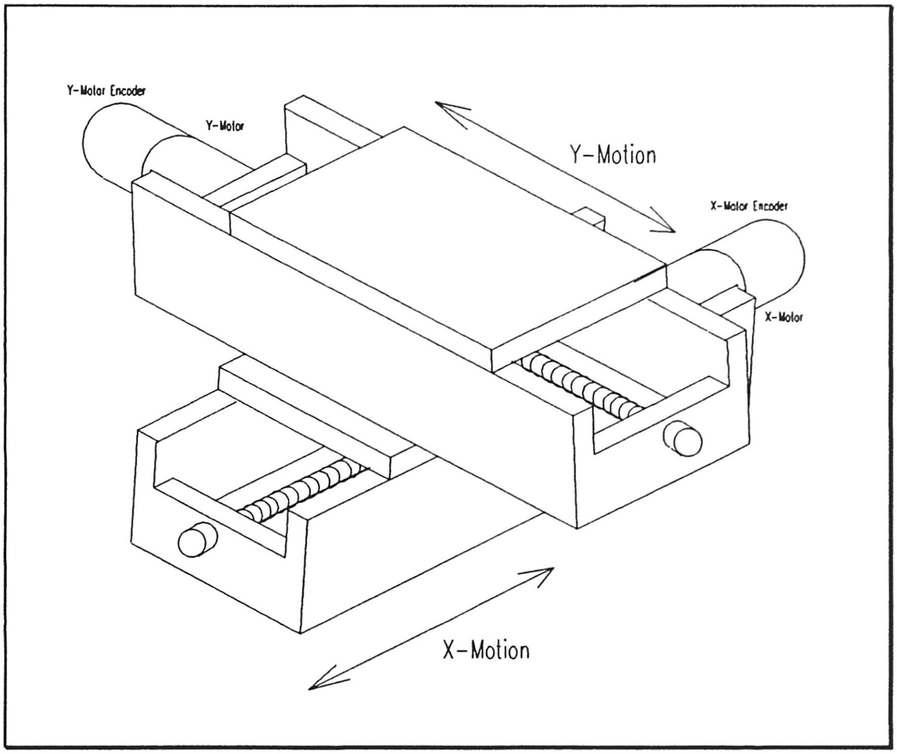


Figure 3: Two-Axis Positioning System Configuration

Each stage has a length of 24 inches, allowing a 24 inch by 24 inch usable workspace. Mounted at the ends of each stage are optical limit switches, used to sense when the positioning stage has reached the limit of travel. This signal can be used to terminate power to the motors once the switch is tripped.

1) Drive System

The motors used for the X-Y table are Aerotech model 1075 DC permanent magnet servo motors. This particular model is capable of a maximum rotational speed of 5000 rpm, and is capable of a continuous power output of 140 watts. Attached to the motor by a flexible coupling is the shaft which the table rides on. Moving the table from one end of the shaft of the other requires 158 revolutions of the shaft, corresponding to a shaft pitch of 0.158 in/rev.

2) Position Encoder System

Also connected to the motor shaft is the rotational encoder system. The encoder translates the rotational motion of the motors into an electrical signal to be interpreted by the controller. The Datametric Sinewave Encoder, which is the model used on the Aerotech motors, has a resolution of 200 pulses per revolution, which corresponds to 0.00079 inches per pulse for a single encoder line. The encoder has three output signals: SIN, COS, and MARKER. The SIN and COS signals are sinusoidal waveforms with an amplitude of 0.5 volts, and a DC offset of +2.5 volts. The MARKER signal is an absolute position signal which pulses to a logic low state once per revolution.

B. Motorola 6811 Microcontroller

The Motorola 6811 is a single-chip programmable device, known as a microcontroller. This particular chip has a bus speed of 2 MHz, and an on-chip memory bank of 8 Kbytes of ROM, 256 Bytes of RAM, and 512 Bytes of Electrically-Erasable-

Programmable-Read-Only-Memory (EEPROM). Five input/output ports are provided on the Motorola 6811, however this project is only concerned with Ports B, an output only port, and Port C, an eight bit input/output port. For development and testing of programs for the microcontroller, Motorola provides the Evaluation Board, or EVB. The EVB essentially the Motorola 6811 microcontroller with a collection of support chips to provide increased flexibility to develop and test application programs. This is the product that the control software for this project was developed and tested on.

V. CONTROL SYSTEM NEED ANALYSIS

The development of the control system was based on the following need statement: **to develop an accurate control system for a two-dimensional positioning system, using the Motorola 6811 microcontroller, a software implementation of the Self Paced Fuzzy Tracking Controller, and the Aerotech X-Y table.** In its barest form, this project involves development of an interface between the path data, stored in the Motorola 6811, and the Aerotech X-Y table. The control system required for the project involves dual levels of control. On one level, the system must have a position control system to allow it to accurately move from point A to point B. On the other level, the system must include a software implementation of Huang and Tomizuka's Self Paced Fuzzy Tracking Controller to set the appropriate motor speed for given path conditions. A schematic diagram of the dual level controller is shown in Figure 4.

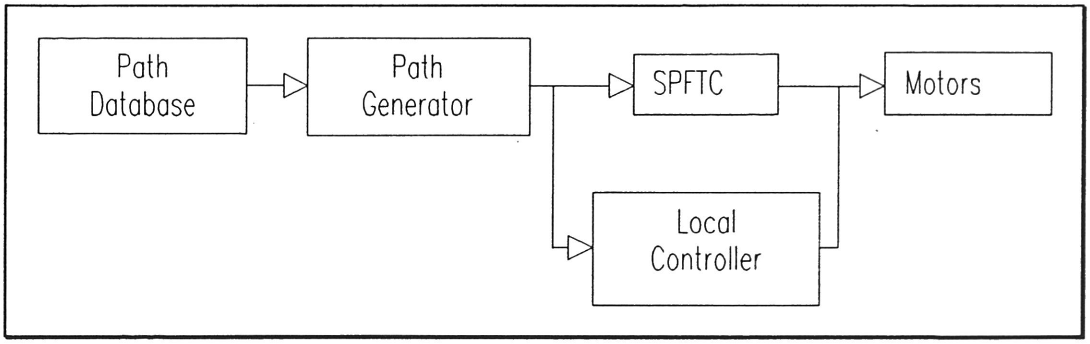


Figure 4: Schematic Diagram of Dual Level Controller

In order to fulfill the need statement, the system can be broken down into five main sub-systems: Path Data Storage, Current Motor Status, Local Position Control, Fuzzy Logic Speed Control, and Actuator Signal Conversion.

A. Path Storage and Generation System

The Path Data Storage system is a means of storing arbitrary path data. This function may be implemented in one of several ways, due to the trade-offs involved given the memory and speed limitations of the Motorola 6811. One possibility would be to store an arbitrary, complex path as a collection of geometric shapes, and subroutines could be stored in the memory of the EVB to produce those shapes when called on. For example, one routine could move the X-Y table in a line given two endpoints, and another could create an arc given a radius, center point and beginning and ending angles. This novel idea would definitely be an ideal way to store a very complex path in a minimal amount of space, however decoding the path in real-time would consume an excessive amount of processor time.

Another solution, which would not require quite as much processing time would be to discretize the path into a list of (x,y) points. This way the controller would only have to move the x-y table from one point to the next, but there exists one primary difficulty that could arise from such a tactic, the memory limitations of the EVB. Discretizing a complex path into a list of points would certainly occupy a large amount of memory, and memory would certainly become the limiting factor to the complexity of the path. There are advantages to both techniques. While one would provide a compressed path storage medium and as much accuracy as the user requires, the other would most certainly reduce the amount of processing required at each step, which would in turn speed up the motion of the system.

As a compromise between the two extremes, the path could also be stored as an analytic approximation of the path, such as spline based or some other functional based approximation. For instance, $\sum_i a_i f_i(x)$ would be used in which case, only coefficients a_i would be stored. From here, one generic subroutine could be implemented to evaluate $f_i(x)$ given the values of i and x. This approach would consume less memory than the tabular approach, and use less processing time than the subroutine approach, but it would only be an approximation of the true path. Regardless of the method used to store the data, there must be a method to derive a set of desired (x,y) coordinates lying on that path separated by a constant arc length. All of this data must be readily available, since it is frequently accessed by several of the other sub-systems.

B. Current Motor Status System

As the name implies, the function of the Current Motor Status system is continuously monitor the status of both the x- and y-motors. In addition to keeping track of the position of each motor, the Current Motor Status system must know the velocity of each motor as well as other vital parameters such as limit switch activation and locked rotor conditions. Accomplishing this task involves an interface with the shaft encoders and limit switches of both motors, a pair of accumulators to count the number of pulses received, and a precise timing device. The timing device will allow the Current Motor Status system to compute the time rate of change of motor position, or velocity, of each motor. A locked rotor condition could be detected by sensing when the position of the motor is constant, or the time rate of change of motor position is zero.

C. Local Position Control System

The Local Position Control system is primarily concerned with moving the Aerotech X-Y table from point A to point B, and correcting for any error incurred along the way. In order to accomplish this task, an interface must be in place to facilitate easy access to the information generated by the Path Data Storage system. This information will be used to assign the desired x- and y-coordinates of the table at a particular point in time. Additionally, the position control system must know its own location in real-time. This involves an interface with the Current Motor Status system to receive up to date information about the position of each motor at any instant of time. Finally, the Local Position Control system will involve a control algorithm will be used to compensate for any error between actual and desired positions of the x- and y-motors by way of the Actuator Signal Conversion system.

D. Fuzzy Logic Speed Control System

The next component of the control system is the Fuzzy Logic Speed Control system. Based on the Self-Paced Fuzzy Logic Controller for Two-Dimensional Motion Control, this system selects a maximum speed for the motors, based on current path conditions, in order to minimize tracking error. Again, the fuzzy controller would require an interface with the Path Data System in order to determine current path conditions, such as: current path curvature and upcoming change in path curvature. Additionally, the fuzzy controller would have to interface with the Current Motor Status system to obtain information about the current speeds of the motor. Based on the fuzzy logic rules, the controller would select an appropriate incremental change in speed, which would be output to the motors via the Actuator Signal Conversion system.

E. Actuator Signal Conversion System

Finally, the Actuator Signal Conversion system is required to convert the digital output signals from the processor into analog signals which can be used by the motors. These signals must also be amplified significantly in order to drive the large motors of the Aerotech X-Y positioning system.

F. Conceptual Designs:

Two possible configurations arise when considering fulfilling the requirements of the design. The first is a total software implementation of the system, and the second is a combination hardware/software system. Before an appropriate structure for the system can be determined, the project need statement must be reviewed. The system must be able to correct for any error encountered while tracking, it must have the ability to follow an arbitrary two-dimensional path, and it must use the Self Paced Fuzzy Tracking Controller algorithm for determining motor speed. Next, the limitations of the Motorola 6811 must be assessed. The 6811 has a limited amount of processing power, therefore the fewer calculations required the more smoothly the system will run. Keep in mind that the 6811 is responsible for controlling both the x- and y-motors, and in order for their motion to appear simultaneous the control program must be very streamlined. Another limitation of the 6811 is the small amount of memory available on which to store programs and data. Minimizing the number of routines in the control code will both streamline the operation of the program and consume less of the available memory. In light of the limitations of the central processing unit selected for this project, it was decided to move as many operations as possible off of the processor and onto integrated circuit hardware components. The processes which were able to be moved form the central processing unit to integrated circuit components, as well as the selection of the IC components will be discussed in the following section.

VI. Hardware Development

This section of the report contains the final design of the hardware to interface the Motorola 6811 with the motors, component selection and component functions, and the final circuit schematic diagram. As stated previously, the decision to develop a combination hardware/software control system resulted in moving several of the intended functions of the system off of the central processing unit into sub-processors. The functions which were able to be achieved using integrated circuit components were: current motor status monitoring system, local position control system, and the actuator signal conversion system. A complete circuit diagram is shown in Appendix A.

A. LM628 Precision Motion Controller

Recently National Semiconductor Corporation released a new integrated circuit designated the LM628 Precision Motion Controller. The LM628 is a dedicated motion-control processor designed for use with a single DC servo motor which provides an incremental position feedback signal, such as the Aerotech motors used in the X-Y table. Note that the LM628 is only capable of controlling a single motor; in the final design two LM628 chips had to be used. In effect, this single chip accomplishes two of the three previously mentioned functions which were incorporated into hardware, the current motor status system and the local position control system. While this component simplified several aspects of the design, it also created some complications by requiring new interfaces between the Motorola 6811 and the shaft encoders from the motors. A block diagram of a typical LM628 application is shown in Figure 5.

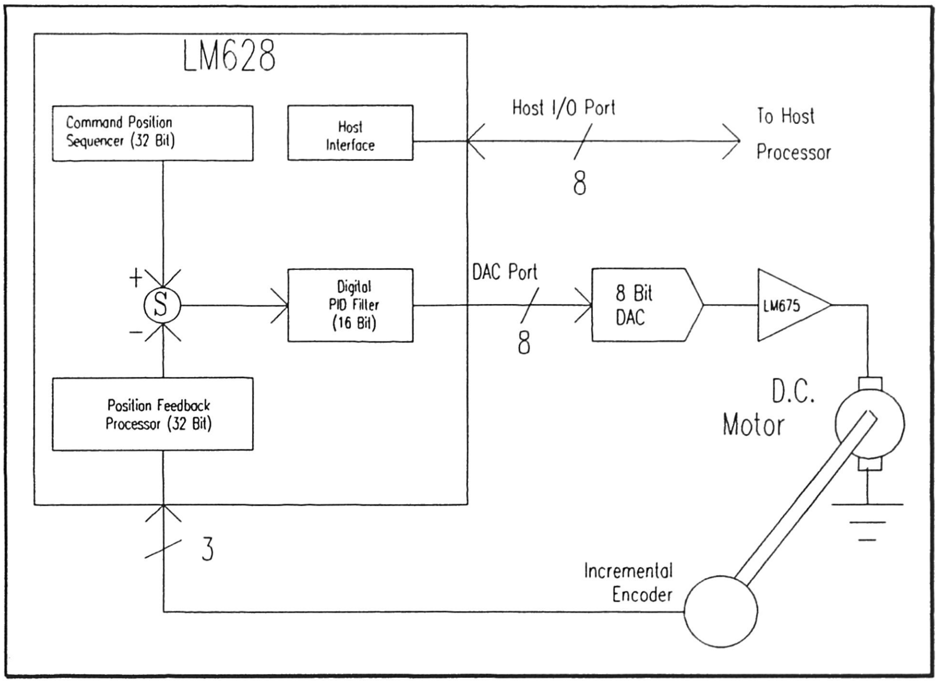


Figure 5:Typical System Block Diagram

The host processor communicates with the LM628 through an input/output (I/O) port to facilitate programming the desired trajectory and the local position controller. The LM628 includes an 8-bit output port to interface with an external digital-to-analog converter to produce the signal that is amplified and applied to the motor. An incremental encoder provides feedback for closing the position servo loop. In operation, the LM628 subtracts the actual position (feedback position) from the desired position, and the resulting position error is processed by the local position controller to drive the motor to the desired position.

1) Position Feedback Interface

The LM628 interfaces to a single motor via an incremental encoder, or in the case of the Aerotech X-Y table, the shaft mounted encoders. The chip is equipped with three inputs: two quadrature signal inputs, and an index pulse input. The quadrature signals are used to keep track of the absolute position of the motor. Each time a logic transition occurs at one of the quadrature inputs, the LM628 internal position register is incremented or decremented accordingly. This provides four times the resolution over a single encoder input line. A diagram of the quadrature input to the LM628 Precision Motion Controller is shown in Figure 6.

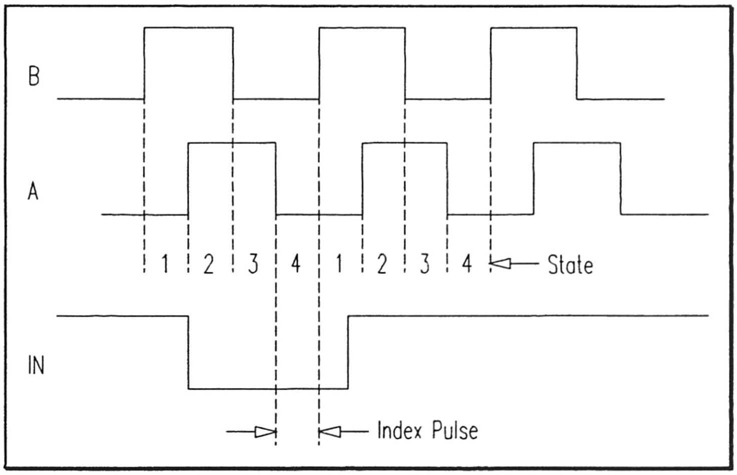


Figure 6: Quadrature Encoder Input Signals

The direction that the motor is traveling in is determined by observing the order that the state transition between the two quadrature inputs occurs. Table 1 displays the logic used by the Precision Motion Controller to determine the direction of motion.

Table 1: Quadrature Encoder Direction Logic

States	B	A
1	1	0
2	1	1
3	0	1
4	0	0
1	1	0
2	1	1
3	0	1

When the logic states appear in the order 4, 3, 2, 1, the A signal leads the B signal and the motor is turning in the clockwise direction (defined as positive by the LM628). On the other hand, when the states appear as 4, 1, 2, 3, then the B signal is leading the A signal and the motor is turning in the negative direction.

The interface with the motor shaft encoders is one of the areas where additional circuitry had to be implemented. The signals from the encoders are sinusoidal waveforms with a DC offset of 2.5 volts and an amplitude of 0.5 volts. The LM628 defines a logic high state as a voltage greater than or equal to 2.4 volts. Unfortunately, a logic low state can only be guaranteed below 0.4 volts, with the area in between as an undefined region. In order to obtain a reliable interface between the LM628 and the shaft encoders, an additional integrated circuit had to be installed.

To allow a compatible interface between the shaft encoders and the LM628, the sinusoidal encoder output must transformed into a quadrature signal which varies between 0 volts and +5 volts. All four encoder channels were able to be changed using National Semiconductor Corporation LM339 Low Power Low Offset Voltage Quad Comparator chip. Applying the encoder output signal to the positive input of one of the four comparators, and applying +2.5 volts to the negative input, the comparator has an output of +5 volts when the positive input is greater than the negative input, and the chip has an output of 0 volts when the positive input is less than the negative input.

2) Trajectory Generation

The trajectory generator computes the desired position of the motor versus time. In the case of this project, the host processor (the Motorola 6811) specifies acceleration, maximum velocity (from the Fuzzy Logic Speed Control system), and final position (from the Path Data Generation system). The LM628 uses this information to affect the move by accelerating as specified until the maximum velocity is reached, or until deceleration must begin to stop at the specified final position. At any time during the move, the maximum velocity and/or the target position may be changed, and the motor will accelerate or decelerate accordingly.

3) PID Compensation Filter

To achieve the requirements of the Local Position Control system, the LM628 uses a digital implementation of the classical three term controller to compensate the control loop. The motor is held at the desired position by applying a voltage to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation illustrates the control performed by the LM628:

$$u(n) = kp \cdot e(n) + ki \cdot \sum_{N=0}^{n} e(n) + kd \cdot [e(n') - e(n' - 1)]$$

where *u(n)* is the motor control signal output at sample time *n*, *e(n)* is the position error at sample time *n*, *n'* indicates sampling at the derivative sampling rate, and *kp*, *ki*, and *kd* are the gains for the proportional, integral and derivative terms respectively.

4) Host Processor Interface

The host processor interface is the link between the Motorola 6811 and the LM628. The 8-bit asynchronous parallel input/output is achieved through a technique known as strobed I/O. This entails signaling pins on the Precision Motion Controller to let it know that data is available, or that data is being requested. A command is sent to the LM628 in the following manner: first the command code is sent from the host computer to the LM628, then a number (from one to seven) of two-byte data words are sent from the host computer to the LM628, or vice versa depending on the nature of the command. There are four main functions performed by the host processor interface: writing commands to the LM628, reading the status bit of the LM628, and writing and reading data from the LM628.

The host processor writes commands to the LM628 via the host I/O port when the Port Select (PS) input pin is logic low. The desired command code is applied to the parallel port line of the host computer (OUT), and the Write (WR) input pin is strobed. When writing command bytes, it is necessary to first read the status byte, and check the state of a flag called the "busy bit". When the busy bit is logic high, no command write may take place. A timing diagram of a typical command write is shown in Figure 7.

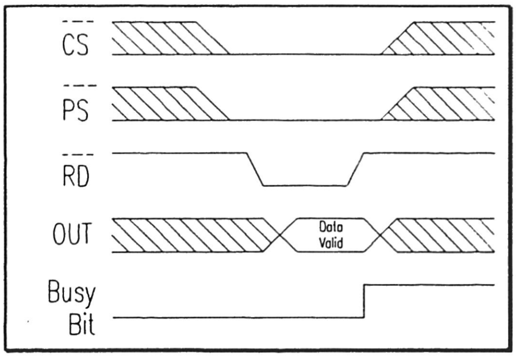


Figure 7: Command Byte Write Timing

The host processor reads the status byte in a similar manner. When the PS input pin is logic low, the Read (RD) pin is strobed, and status information remains valid as long as RD is low. Shown in Figure 8 is the timing diagram for a typical status byte read operation.

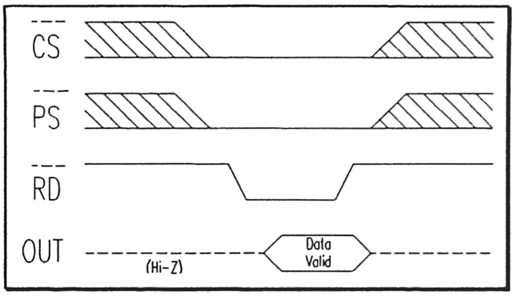


Figure 8: Status Byte Read Timing

Writing and reading data to and from the LM628 are done with the PS input pin logic high. These writes and reads are always a number (from one to seven) of two-byte words, with the first byte of each word being the more significant. Each byte requires a write (WR) or a read (RD) strobe. When transferring data words, it is necessary to first read the status byte and check the state of the busy bit. Timing diagrams for typical read and write data commands are shown in Figure 9 and Figure 10 respectively.

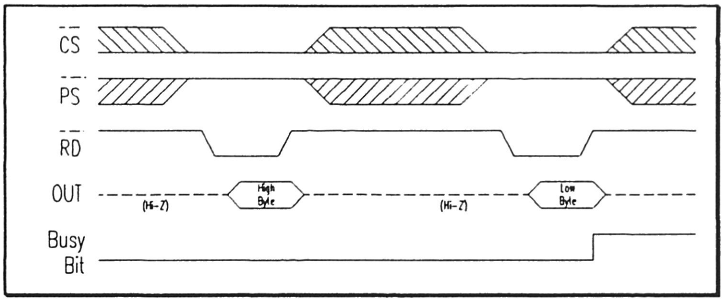


Figure 9: Data Word Read Timing

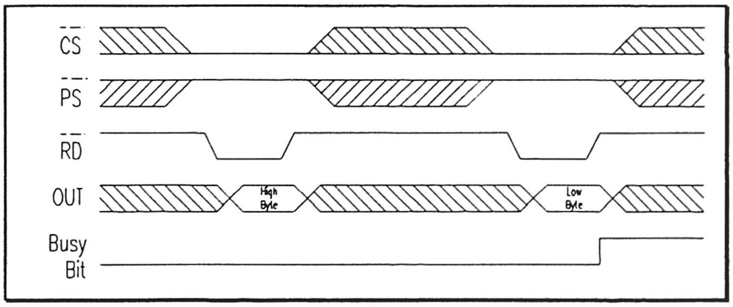


Figure 10: Data Word Write Timing

Again, the host processor interface is the source of numerous complications due to incompatibilities between the Motorola 6811 and the LM628. First of all, the Motorola 6811 has only one 8-bit bi-directional data port. Since the 6811 will be responsible for controlling two LM628 chips, a complex switching scheme was developed to use the single I/O port on the Motorola 6811 to interface with both of the LM628 chips. In order to physically implement this interface, five bits of an eight bit output only port on the Motorola 6811 were dedicated to control signals. These five control bits are: X/Y Select, Read/Write, Port Select, Read Strobe, and Write Strobe. The X/Y Select bit is used to select between communication with the x- or y-motor. As the name implies, the Read/Write bit determines the direction of the communication between the two processors. The Port Select pin is used exclusively by the LM628 to determine what type of signals are being requested (commands/data), and the Read and Write Strobe pins are also exclusively used by the LM628 and their function will be discussed in the following paragraph.

Another complication arose when trying to interface the strobing capabilities of the Motorola 6811 with the LM628 chips. In this case, the Motorola 6811 had one output strobe and one input strobe, while the LM628 require two independent output strobe pins in order to operate the read and write pins. To overcome this difficulty, National Semiconductor Corporation's 74LS123 integrated circuit was installed. The 74LS123 is capable of generating dual independent output pulses from a few nano-seconds to extremely long duration. The Read and Write Strobe command bits were connected to this device to generate the read and write strobe signals. When the output bit goes through a transition from logic low to logic high, the chip outputs a pulse which lasts for a pre-determined length of time. The pulse duration is set through the selection of a resistor and capacitor to use in the 74LS123 circuit. For this project a 100 kΩ resistor and a 100 pF capacitor were used to generate a pulse which lasts 5.5×10^3 ns.

B. Interfacing the LM628 to the Motorola 6811

As stated before, interfacing the two LM628 chips to the Motorola 6811 with a single eight bit input/output port created many problems. In order have the capability to communicate with both motors simultaneously, a complex switching scheme was developed. The plan which was implemented involves two National Semiconductor Corporation 74LS245 Octal Bus Transceiver integrated circuits for communication from the Motorola 6811 to the LM628 chips. Also, two National Semiconductor Corporation

74LS257 2-Data Selector/Multiplexer integrated circuits were used to allow communication from the LM628 chips to the Motorola 6811.

1) Communication from the Motorola 6811 to the LM628

The 74LS245 Octal Bus Transceiver is designed for asynchronous two-way communication between data busses. This particular integrated circuit was selected because of its isolation capability. Depending on the logic state of the Enable pin (pin 19), the device either allows communication across the bus, or isolates the bus as if it were removed from the circuit. The following table summarizes the operation of the Octal Bus Transceiver:

Table 2: Function Table for Octal Bus Transceiver

Enable (G)	Direction Control	Operation
L	L	B data to A bus
H	X	Isolation

H = high level, L = low level, X = irrelevant

The direction control pin of each transceiver was permanently grounded to achieve a logic low state, while the Enable pin was connected to the Read/Write output pin on the Motorola 6811. When the Read/Write pin is logic low, the Motorola 6811 is in Write mode, and the Octal Bus Transceiver allows communication from the Motorola 6811 to the LM628. On the other hand, when the Read/Write pin is logic high, the transceiver goes into isolation effectively removing the components from the circuit.

Notice that the hardware does not have the capability to turn on one transceiver and not the other. Communication from both LM628 chips simultaneously is avoided through the use of the Chip Select pin on the LM628. The Precision Motion Controller chip only permits communication with the host computer when the Chip Select pin is at a logic low state. To allow only one controller to communicate at a time, the X/Y Select pin on the Motorola 6811 is connected directly to the LM628 for the x-motor, while its inverse is connected to the LM628 for the y-motor. Thus, when X/Y Select is logic low, the Chip Select pin is logic low at the x-motor and logic high at the y-motor allowing communication with the x-motor and not the y-motor. When the X/Y Select pin is logic high, the reverse is true.

2) Communication from the LM628 to the Motorola 6811

The 74LS257 2-Data Selector/Multiplexer is designed to select between two inputs for a single output signal. Each 74LS257 chip is capable of accepting four sets of input bits and producing four outputs. Because of this, two chips were used, with the top four bits of each LM628 host I/O port is connected to one data selector, and the bottom four bits are connected to the other. The following table summarizes the operation of the 2-Data Selector/Multiplexer integrated circuit:

Table 3: Function Table for 2-Data Selector

Output Control	Select	Input A	Input B	Output Y
H	X	X	X	Z
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

H = high level, L = low level, X = irrelevant, Z = isolation

First of all, the select pin (pin 1) was connected directly to the X/Y Select pin on the Motorola 6811. Thus, when the X/Y Select pin is logic low, the 2-Data selector uses input A, which is connected to the x-motor. On the other hand, when the X/Y Select pin is logic high, the chip uses input B, which is connected to the y-motor. The Read/Write pin on the Motorola 6811 is first put through an inverter and connected to the Output Control of the 2-Data Selector. When Read/Write is logic low, the Motorola 6811 is in write mode, and the inverse signal, logic high, puts the selector in a high impedance state, isolating it from the circuit. On the other hand, when Read/Write is logic high, the Motorola 6811 is in read mode, and the inverse signal, logic low, allows the chip to select the A or B input channel. This allows the LM628 to send data to the Motorola 6811. Notice that when the Octal Bus Transceiver is in a high impedance state, the 2-Data Selector is active, and vice versa. This ensures that the hardware may only perform one operation (read or write) at a time. The following table illustrates the function of each control bit by showing he state of each affected device at both logic states of the bits.

Table 4: Summary of Control Bit Functions

Pin	Logic Low	Logic High
X/Y Select	X-Motor State CS Low on LM628-X CS High on LM628-Y Input A (LM628-X) Selected	Y-Motor State CS High on LM628-X CS Low on LM628-Y Input B (LM628-Y) Selected
Read/Write	Write Mode (6811 to LM628) Transceiver in Active State Selector in Isolation State	Read Mode (LM628 to 6811) Transceiver in Isolation St. Selector in Active State
Port Select	Write Commands to LM628 Read Status Bit	Read Data from LM628 Write Data to LM628

C. Digital to analog conversion and signal amplification

The final component of the interface between the Motorola 6811 and the Aerotech X-Y table is the connection between the LM628 chips and the motors. The Precision Motion Controller chip is equipped with an eight bit output port for connection with a digital to analog converter. The digital output of the LM628 varies from $00 to $FF in hexadecimal, with $00 being full speed counter-clockwise, $80 being zero speed, and $FF as full speed clockwise. The digital to analog converter changes this digital signal to an analog signal to drive the motors. The output from the digital-to-analog converter is a low-current signal, and to drive the large motors of the Aerotech X-Y table, the analog signal must be power amplified. This is accomplished using a National Semiconductor Corporation LM675 Power Operational Amplifier. Using such a device not only provides reliable signal amplification, but also includes several safety features built into the device. The LM675 has short circuit protection as well as thermal protection which could prevent costly damage to the table motors and the LM675 device.

VII. Control Software Development

Before going into the development of the control software, it is necessary to review the function structure of the software system. As shown in Figure 11, the system is responsible for storing path data, and generating an array of (x,y) points from this data.

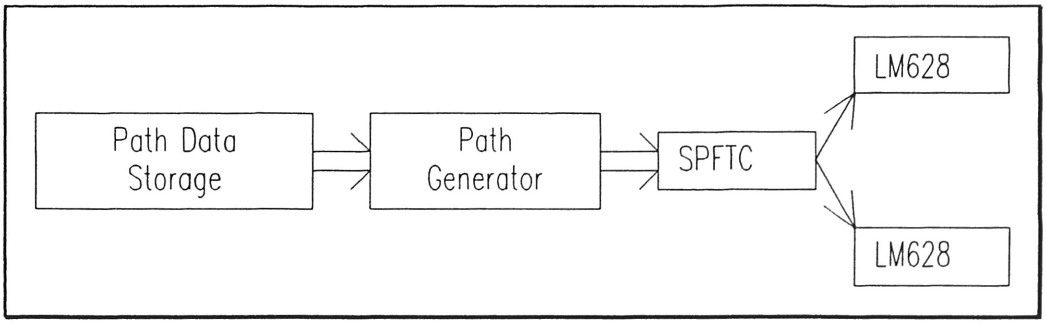


Figure 11: Control Software Schematic

The code must also compute path parameters such as current path curvature and change in path curvature. Velocity of the motors must be specified using the Self Paced Fuzzy Tracking Controller algorithm, and finally the control software must interface the Motorola 6811 with both of the LM628 Precision Motion Controller chips to transmit data. From this list of system functions and the limitations of the Motorola 6811 microcontroller, two driving requirements emerge for the control code. First, the code must be streamlined; the fewer calculations and operations the code must perform, the better. Second, the control software must be compact such that it will fit an a minimum amount of memory.

A. Design Selection

1) Programming Language

Based on the functions of the system, and the design requirements, several decisions were made in the development of the software to better fulfill the requirements. First, assembly language was chosen for the control software language. Using this language allows the programmer to fully optimize the code, since he/she has absolute control over the processor. The program must be developed in a logical step by step process, where nothing is assumed. This approach lends itself to the problem at hand, because the higher degree of programmer control allows for better optimization of system resources.

2) Path Data Storage and Path Generation

In order to minimize the number of calculations required at each iteration of the control loop, the path generation function was removed. To replace this sub-system, the path is stored in the Motorola 6811 memory as an array, as illustrated in Table 5:

Table 5: Path Data Storage Format

Increment in X-Position	Increment in Y-position	Current Path Curvature	Change in Path Curvature	X-Velocity Scaling Factor	Y-Velocity Scaling Factor
ΔX_1	ΔY_1	C_1	CC_1	V_{x1}	V_{y1}
ΔX_2	ΔY_2	C_2	CC_2	V_{x2}	V_{y2}
...	...	...	...	...	...
ΔX_n	ΔY_n	C_n	CC_n	V_{xn}	V_{yn}

The parameters which are stored in the path data array are the increment in the x- and y-position to move from the current position to the next position, the current path curvature, the change in path curvature based on the preview distance, and the x- and y- velocity scaling factors. The position is stored as an incremental position for two reasons: 1) instead of storing a full 32 bit position, the position increment may be stored as an eight bit number reducing the number of memory bytes required to store the path, and 2) the incremental position aids in decomposition of the overall velocity, as determined by the Self Paced Fuzzy Tracking Controller, into individual velocities for the x- and y-motors.

3) Self Paced Fuzzy Tracking Controller Implementation

The third and final change implemented to achieve the requirements of the software design was to reduce the Self Paced Fuzzy Tracking Controller algorithm to a two-input, single-output system. Although the Self Paced Fuzzy Tracking Controller was reduced to a more simplified version, the basic theory behind the controller is still in place. Instead of three input variables, the modified fuzzy controller only has two, curvature and change in curvature. Also, instead of the output variable being a relative change in velocity, it is an absolute velocity. These changes minimize the number of steps involved in the fuzzy controller routine to decrease code size and increase speed of code operation. The rule base for the modified Self Paced Fuzzy Tracking Controller is shown in the following table:

Table 6:Rule Base for Modified SPFTC

			C	
		S	M	L
CC	S	L	M	S
	M	M	M	S
	L	S	S	S

Fuzzy rules are derived from the above table as illustrated by the following example:

if c is S and cc is S then v is L

which states that if the path curvature is small and is not changing much in the preview distance then the velocity is large.

B. Path Parameter Calculations

In the operation of the control software system, multiple real-time calculations must be performed to determine the conditions of the path at a given instant of time. The first of which is the current curvature of the path. For a path where the y-coordinate may be represented as a function of the x coordinate, the path curvature is calculated by:

$$c(x) = \sqrt{1+[f'(x)]^2}$$

Another calculation which must be performed at each iteration of the control code is the decomposition of the velocity specified by the Self Paced Fuzzy Tracking Controller algorithm into appropriate velocities for the x- and y-motors. Using the incremental changes in the x- and y-position and a few trigonometric relationships, the decomposed velocity is found by:

$$ {}_x = V \sin\left(\tan^{-1}\frac{V_x}{V_y}\right)$$

$$ {}_y = V \cos\left(\tan^{-1}\frac{V_x}{V_y}\right)$$

C. Control Software Flowchart

The operation of the control system software may be divided into two sections: 1) the hardware initialization section, and 2) the control loop.

1) Initialization Section

The hardware initialization section is responsible for initializing both the Motorola 6811 microcontroller and the LM628 Precision Motion Controller chips, and a flowchart for the initialization code is shown in Figure 12.

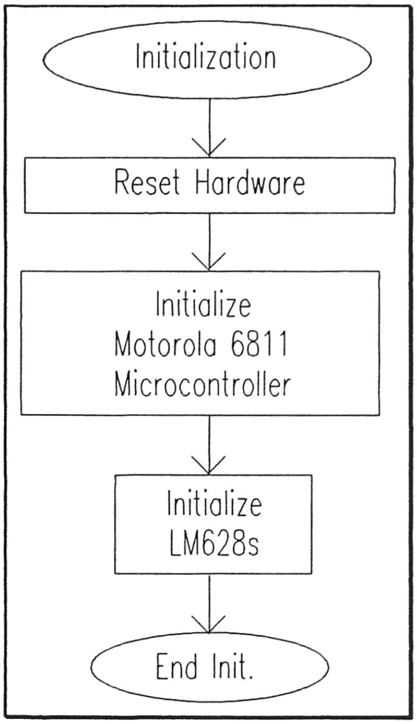


Figure 12: Flowchart for Initialization Section of Control Code

The first set of operations in the control code must define the areas of the Motorola 6811 memory that the control code will use. This includes definition of system variables, memory locations to store program code and path data, memory location of the stack, and initialization of system pointers such as the path data pointer. Following the Motorola 6811 hardware initialization, the LM628 chips must be initialized. The first command to be sent to both chips is the **Reset LM628** command. This command results in setting the PID filter coefficients, the trajectory parameters, the motor position counters, and the motor control output to zero. After this command, the **Load Filter Parameters** command must be executed on both motors to set the PID filter coefficients for the Precision Motion Controllers. Once the filter parameters have been loaded into the LM628 buffers, the **Update Filter** command must be sent to make the new filter parameters valid. This command completes the initialization section of the code.

2) Control Loop

Following hardware initialization, the software system falls into a control loop which is executed repeatedly until the desired path is complete. A flowchart for the control loop software is shown in Figure 13.

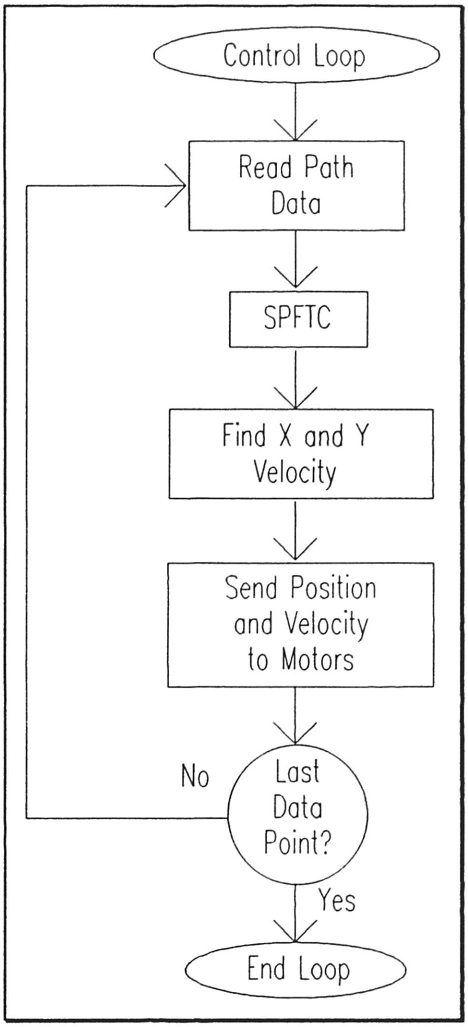


Figure 13: Flowchart for Control Loop Operation

The first operation in the control loop is to reference the stored path data to obtain parameters for the position increment in the x- and y-direction, the current curvature at that point, the previewed change in curvature, and the scaling factors to decompose the velocity set by the Self Paced Fuzzy Tracking Controller. Once these parameters have been read, the Self Paced Fuzzy Tracking Controller subroutine is called to determine the set speed for the motors. The assembly code for the Self Paced Fuzzy Tracking Controller was developed using a computer package called FIDE. This allows the programmer to input membership functions in graphical form and fuzzy logic rules in natural language terms, and the package will convert the rules and membership functions into an assembly code routine for the Motorola 6811. Once the Self Paced Fuzzy Tracking Controller subroutine outputs a velocity, the scaling factors are used to change it into a velocity for the x- and y-motors. Finally, all of this data is assembled, and the **Load Trajectory Parameters** command is sent to the LM628 chips followed by the path data for that iteration of the control code. Once this command and all data has been sent to the buffers of both motion control chips, the **Start Motion** command is sent to better synchronize the motion of the system. At this stage, the control code tests to determine whether the end of the path has been reached, and either stops or jumps back to the beginning of the control loop.

The control software is written with a modular, top-down approach which utilizes as many subroutines as possible. This reduces the size of the control code by using portions of the code again and again. For example, the code sequence to send a command to the motors is implemented in a subroutine. Whenever the control software needs to write a command to the LM628, it pushes the code for the command onto the stack and calls the write command subroutine.

VIII. Experimental Results

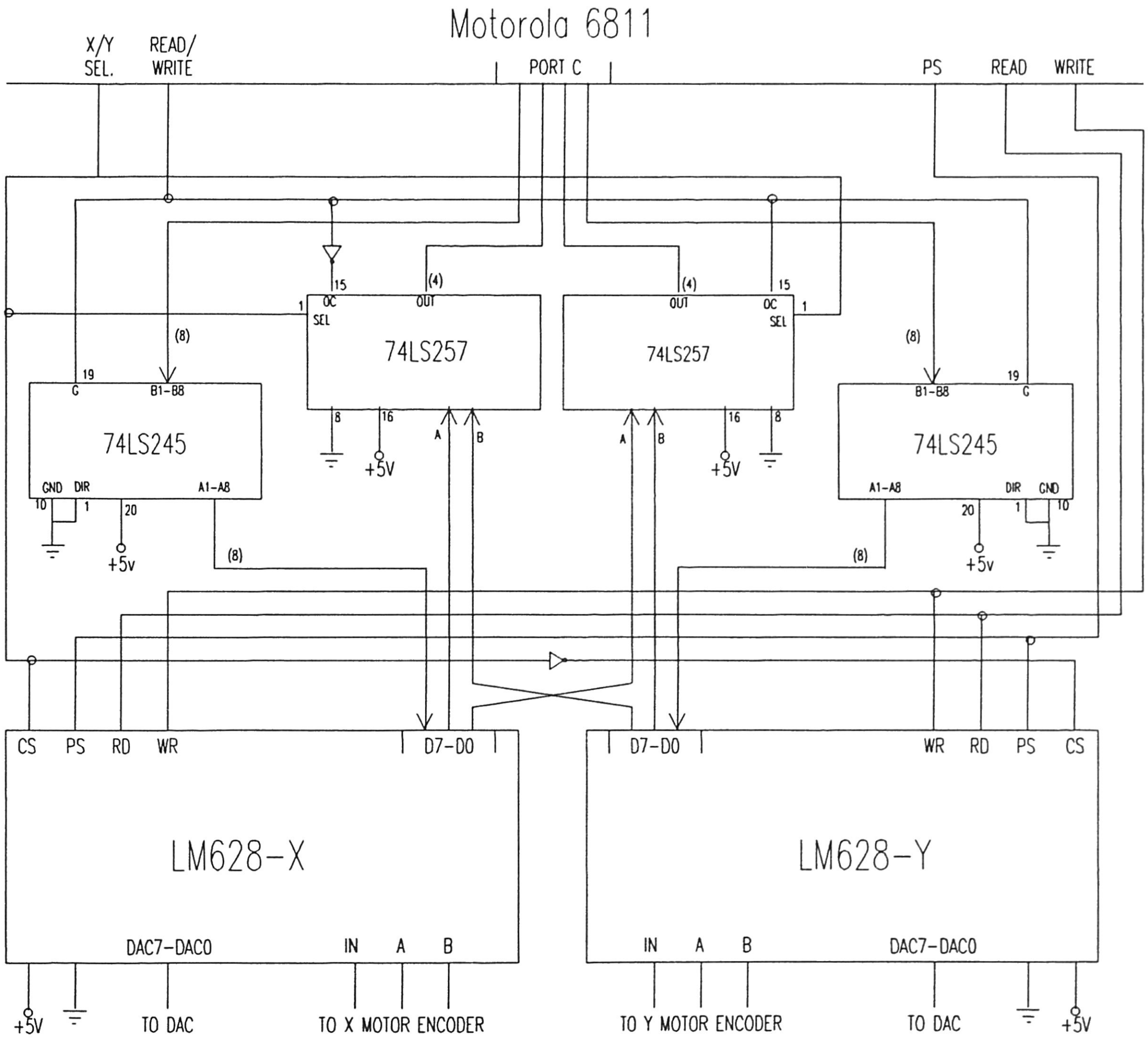
In the end, the control software was developed in three separate stages, each incorporating new commands and greater levels of complexity. Stage one, which is listed in Appendix B, is a simple program which would command the Precision Motion Controller chips to reset, and then continuously read the current motor position from the device. The main function of this program was to test the operation of the system hardware, ensuring that data could both be read from and written to the LM628. The next program, denoted Stage II and shown in Appendix C, is written to tune the system operation. This program performs a reset on one of the LM628 chips, loads a proportional gain into the controller, and move a set distance of ten inches. This technique, known as tuning the PID controller through observing the response of the system to a step input, is commonly used to obtain the best response from a controller. Once the best gains for the PID control algorithm have been selected, the third stage of the program, listed in Appendix D, could be implemented. This program is the full version of the control program, which is able to track an arbitrary path of (x,y) points and includes the fuzzy control routine to set the desired motor speed based on the current path conditions.

Unfortunately, the system hardware did not even pass the first stage of software development. The code itself was valid, and the microcontroller was producing the appropriate signals, however, the LM628 Precision Motion Controller chips were determined to be damaged. Upon performing a hardware reset on the chips, they are designed to display a particular code at the host communications port, which indicates that a successful chip reset has been performed. Also, the motor output port which connects to the digital-to-analog converter is supposed to automatically reset to the "zero" value of $80 in hexadecimal. Neither the successful reset code not the "zero" motor output were observed in the hardware. Based on this, it was concluded that the hardware was defective, and unfortunately could not be replaced within the time frame of this project.

Despite the failure of the Precision Motion Controller chips, it was possible to observe the successful operation of the remaining system hardware. Using an oscilloscope, the levels of the Read/Write, Port Select, and Read and Write strobe control bits were observed. From this observation, all of the software functions were working properly, and well within the specifications of the Precision Motion Controller device.

This project is intended to be the background material for a master's thesis project. Once the control system is up and running, it will provide a performance benchmark in terms of speed and accuracy of the two-axis positioning system. From there, different system configurations using microcontroller of various speed and accuracy will be used to determine the optimum motion control configuration. This optimum configuration would be that which provides the greatest level of speed and accuracy while at the same time being cost efficient.

Appendix A. Experimental Circuit Diagram



Appendix B. Stage I Control Code Listing

```
** Equates - Registers will be addressed with Ind,X mode
*
*REGBAS  EQU     $1000   ;Starting address for register block
PORTB   EQU     $1004   ;Output port B
PORTC   EQU     $1003   ;I/O port C
DDRC    EQU     $1007   ;Port C direction Control
POSIT   EQU     $C000   ;POSITION DATA (C000-C003)
COMM    EQU     $C004   ;COMMAND DATA

*** Start of program

        ORG     $C005   ;Prog starts in EVB RAM at $C005
        LDS     #$0047  ;Top of User's stack area on EVB
        LDY     #$1000  ;
        LDAA    #$00    ;RESET MOTOR
        STAA    COMM    ;PUSH ONTO STACK
        JSR     WRCOM   ;WRITE COMMAND
LOOP    LDAA    #$0A    ;READ POSITION Command
        STAA    COMM    ;STORE COMMAND
        JSR     WRCOM   ;WRITE COMMAND
        LDX     #$C000  ;RESET DATA STORAGE POINTER
        JSR     RDATA   ;READ FIRST 2 BYTES
        JSR     RDATA   ;READ LAST 2 BYTES
        BRA     LOOP    ;INFINITE LOOP TO READ POSITION

*Subroutine to test busy bit (X-only)
BUSY    LDAA    #$00            ;All port C lines to input
        STAA    DDRC            ;SET PORT C TO INPUT
        LDAA    #$08            ;X/Y=L, R/W=H, PS=L, RD=L, WR=L
        STAA    PORTB           ;STORE PORTB CONFIG
TEST    BSET    $04,Y,$02       ;STROBE READ PIN
        BRCLR   $03,Y,$01,GO    ;IF BUSY BIT CLEAR, EXIT LOOP
        BCLR    $04,Y,$02       ;CLEAR READ STROBE
        BRA     TEST            ;LOOP BACK AND TEST AGAIN
GO      BCLR    $04,Y,$02       ;CLEAR READ STROBE
        RTS

*Subroutine to Write Commands to LM628 (X-Only)
WRCOM   LDAB    COMM            ;RETREIVE COMMAND
        LDAA    #$FF            ;
        STAA    DDRC            ;SET PORT C TO OUTPUT
        LDAA    #$00            ;X/Y=L, R/W=L, PS=L, RD=L, WR=L
        STAA    PORTB           ;STORE PORTB CONFIG
        STAB    PORTC           ;PUT COMMAND ON PORT C
        BSET    $04,Y,$01       ;STROBE WRITE PIN
        BCLR    $04,Y,$01       ;CLEAR WIRTE STROBE PIN
        JSR     BUSY            ;TEST BUSY BIT
        RTS                     ;

*** Subroutine to Read Data from LM628 (X-only)
RDATA   LDAA    #$00            ;
        STAA    DDRC            ;SET PORT C TO RECEIVE
        LDAA    #$0C            ;X/Y=L, R/W=H, PS=H, RD=L, WR=L
        STAA    PORTB           ;SET COMMAND BITS
        BSET    $04,Y,$02       ;STROBE READ PIN
        LDAA    PORTC           ;LOAD RESULT FROM LM628
        BCLR    $04,Y,$02       ;CLEAR READ STROBE
        STAA    $00,X           ;STORE
        INX                     ;INCREMENT ADDRESS POINTER
        BSET    $04,Y,$02       ;STROBE READ PIN
        LDAA    PORTC           ;LOAD RESULT FROM LM628
        BCLR    $04,Y,$02       ;CLEAR READ STROBE
        STAA    $00,X           ;STORE IN ADDRESS
        JSR     BUSY            ;TEST BUSY BIT
        INX                     ;INCREMENT ADDRESS POINTER
        RTS                     ;
```

APPENDIX C. STAGE II OF CONTROL CODE LISTING

```
** Equates - Registers will be addressed with Ind,X mode
*
*REGBAS  EQU     $1000   ;Starting address for register block
PORTB   EQU     $1004   ;Output port B
PORTC   EQU     $1003   ;I/O port C
DDRC    EQU     $1007   ;Port C direction Control
COMM    EQU     $C000   ;LOCATION FOR COMMAND STORAGE
FCW     EQU     $C001   ;LOC FOR FILTER CONTROL WORD
GAIN    EQU     $C003   ;LOC FOR PROPORATIONAL GAIN
PCW     EQU     $C005   ;POSITION CONTROL WORD
ACCEL   EQU     $C007   ;ACCELERATION
VELOC   EQU     $C00B   ;VELOCITY
POSIT   EQU     $C00F   ;POSITION
*** RAM Variable Assignments
*** Start of program

        ORG     $C013   ;Prog starts in EVB RAM at $C000
        LDS     #$0047  ;Top of User's stack area on EVB
        LDX     #$C001
        LDY     #$1000
        LDAA    #$00    ;RESET MOTOR
        STAA    COMM    ;PUSH ONTO STACK
        JSR     WRCOM   ;WRITE COMMAND
*** DEFINE BITS IN MEMORY
        LDD     #$0008  ;FILTER CONTROL WORD
        STD     FCW
        LDD     #$00FF  ;PROPORTIONAL GAIN
        STD     GAIN
        LDD     #$002A  ;POSITION CONTROL WORD
        STD     PCW
        LDD     #$0000  ;TOP BYTES OF ACCELERATION
        STD     ACCEL
        LDD     #$0025  ;BOTTOM BYTES OF ACCELERATION
        STD     ACCEL+02
        LDD     #$0002  ;TOP BYTES OF VELOCITY
        STD     VELOC
        LDD     #$BB0D  ;BOTTOM BYTES OF VELOCITY
        STD     VELOC+02
        LDD     #$0000  ;TOP BYTES OF POSITION
        STD     POSIT
        LDD     #$CE40  ;BOTTOM BYTES OF POSITION
        STD     POSIT+02
*** END PARAMETERS
        LDAA    #$1E    ;LOAD FILTER PARAMETERS Command
        STAA    COMM
        JSR     WRCOM   ;WRITE COMMAND
        JSR     WRDATA  ;WRITE FILTER DATA
        JSR     WRDATA  ;WRITE DATA
        LDAA    #$04    ;UPDATE FILTER COMMAND
        STAA    COMM
        JSR     WRCOM   ;WRITE COMMAND
*** END INITIALIZATION SECTION
        LDAA    #$1F    ;LOAD TRAJECTORY COMMAND
        STAA    COMM
        JSR     WRCOM   ;WRITE COMMAND
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        JSR     WRDATA  ;WRITE DATA
        LDAA    #$01    ;START MOTION COMMAND
        STAA    COMM
        JSR     WRCOM   ;WRITE COMMAND
END     BRA     END     ;ENDLESS LOOP

*** SUBROUTINES
*STROBE READ PIN
RDSTR   BSET    $04,Y,$02       ;STROBE READ PIN
        BCLR    $04,Y,$02       ;CLEAR READ STROBE BIT
        RTS

*STROBE WRITE PIN
WRSTR   BSET    $04,Y,$01       ;STROBE WRITE PIN
        BCLR    $04,Y,$01       ;CLEAR WRITE STROBE BIT
        RTS

*TEST BUSY BIT OF LM628-X
BUSY    LDAA    #$00              ;All port C lines to input
        STAA    DDRC              ;SET PORT C TO INPUT
        LDAA    #$00              ;SETUP PORTB
        STAA    PORTB
TEST    JSR     RDSTR             ;STROBE READ PIN
        BRSET   $03,Y,$01,TEST    ;TEST AGAIN IF BUSY BIT HIGH
        RTS

*WRITE DATA TO THE LM628-X
WRDATA
*** NOTE THAT BYTES TO SEND MUST BE AN EVEN NUMBER!!!
        LDAA    #$FF
        STAA    DDRC              ;SET PORT C TO OUTPUT
        LDAA    #$0C
        STAA    PORTB             ;SET COMMAND BITS
        LDAA    $00,X             ;LOAD DATA
        INX                       ;INCREASE COUNTER
        STAA    PORTC             ;PUT DATA ON PORTC
        JSR     WRSTR             ;STROBE WRITE PIN
        LDAA    $00,X             ;RETREIVE DATA
        INX                       ;INCREASE COUNTER
        STAA    PORTC             ;PUT DATA ON PORTC
        JSR     WRSTR             ;STROBE WRITE PIN
        JSR     BUSY              ;TEST BUSY BIT
        RTS

*WRITE A COMMAND TO THE LM628-X
WRCOM   LDAA    #$FF
        STAA    DDRC              ;SET PORT C TO OUTPUT
        LDAA    #$08              ;SET UP CONTROL BITS
        STAA    PORTB
        LDAB    COMM              ;RETREIVE COMMAND
        STAB    PORTC             ;PUT COMMAND ON PORT C
        JSR     WRSTR             ;STROBE WRITE PIN
        JSR     BUSY              ;TEST BUSY BIT
        RTS
```

Appendix D. Stage III Control Code Listing

```
** Equates - Registers will be addressed with Ind,X mode
*
REGBAS  EQU     $1000   Starting address for register block
PORTB   EQU     $04     Output port B
PORTC   EQU   $03     Input/Ouptut port C
DDIRC   EQU   $07     Data Direction port C

COUNTX  EQU     $DFF0
COUNTY  EQU     $DFF7

*** RAM Variable Assignments

          ORG      $D000   Start variables in EVB RAM (upper
                    half)

HDLY     RMB      2       Half-cycle delay (in 0.5μS
                    increments)
PWMP1P   RMB      1       1% of PWM period (1 to 256 cycles) Ex 10-7
PWMDC1   RMB      1       Duty cycle for PWM signal at OC2 pin
PWMDC2   RMB      1       Duty cycle for PWM signal at OC3 pin
IC1DUN   RMB      1       flag: 0-not done,1-pulse measured
IC1MOD   RMB      1       s/w mode flag: FF-off,0-1st,1-last
                    edge
OVCNT1   RMB      1       Overflow count (upper 8-bits of
                    result)
RES1     RMB      2       Pulse Width in cycles (16-bits)
HTEMP    RMB      3       Temp for H6TOD8 (3 bytes)
FRSTE    RMB      2       Time of first edge (16-bits)
PERC     RMB      2       Period in cycles (16-bits)
TEMP1    RMB      2       Temp for conversion (16-bits)
FREQH    RMB      2       Freq in Hex (16-bits)
HPW      RMB      2       Pulse Width (16-bits hex)
DBUFR    RMB      8       Decimal result buffer (8 bytes
                    ASCII)
* Some routines use only first 5 bytes of DBUFR
PWMPER   RMB      2       Period of PWM signals in (cycles)
OFFHI    RMB      2       OC2 high offset (calculated)
OFFLO    RMB      2       OC2 low offset (calculated)

      ORG      $C000   Prog starts in EVB RAM at $C000
      SEI
*End initialization section
      LDAA  #$00          Initialize motor command
      PSHA
      LDAA  #$00          Initialize motor command
      PSHA
      JSR   WRXY          Write command to x&y motors
      JSR   LFIL          Load filter parameters x&y
      LDX   #$0000              Reset path data pointer
LOOP  LDAA  $B7FF,X             Load path curvature data
      STAA  $INPUTS+00    Set up SPFTC invar c
      INX                 Increment data pointer
      LDAA  $B7FF,X             Load change in curvature data
      STAA  $INPUTS+01    Set up SPFTC invar cc
      INX                 Increment data pointer
      JSR   SPFTC         Fuzzy speed setting routine
      LDAA  $B7FF,X             Load Vx scaling factor
      INX                 Increment data pointer
      PSHY                Store contents of IY register
      LDY   #$0000              Reset IY register
```

```
       JSR   DECOMP            Find Vx component of velocity
       LDAA  $B7FF,X           Load Vy scaling factor
       INX               Increment data pointer
       JSR   DECOMP            Find Vy component of velocity
       PULY              Reset original contents of IY
       JSR   OUTPUT            Output desired motion
       LDAA  #$01        Start motion command
       PSHA
       LDAA  #$01        Start motion command
       PSHA
       JSR   WRXY        Write command to x&y motors
       BRA   LOOP


DECOMP       Subroutine to decompose Velocity into Vx and Vy
       LDAB  $OUTPUT+00  Put velocity in acc B
       MUL               Multiply and keep top 8 bits
       LDAB  #$F1        Velocity scaling factor
       MUL
       STD   $VELOC,Y    Store in memory
       INY
       INY
       RTS

SPFTC
; For AVMAC11 Macro Assembler,
; the following 3 lines should be included without comment symbol:
;       PUBLIC INPUTS, OUTPUTS, FIU
;       DEFSEG variables, ABSOLUTE
;       SEG variables
        ORG $00
; c is stored in the INPUTS+00
; cc is stored in the INPUTS+01
INPUTS     RMB  $02
; v is stored in the OUTPUTS+00
OUTPUTS    RMB  $01
TVLST    RMB $09
IN_BUF     RMB  1
MF1_REG    RMB  1
MF2_REG    RMB  1
AND_REG    RMB  1
OR_REG     RMB  1
TUH_REG    RMB  1
TUL_REG    RMB  1
SUH_REG    RMB  1
SUL_REG    RMB  1
OUT_REG    RMB  1
BUF_REG    RMB  1
        ORG $D000
MF_OFF     EQU $D000
IMFPTR
        FDB MF1
        FDB MF2
        FDB MF3
        FDB MF4
        FDB MF5
        FDB MF6
        FDB MF7
        FDB MF8
MF1
        FCB  $01
        FCB  $03
        FCB  $00
        FCB  $20
```

```
                FCB   $FF
                FCB   $80
                FCB   $00
                FCB   $00
                FCB   $FF
MF2
                FCB   $02
                FCB   $03
                FCB   $00
                FCB   $20
                FCB   $00
                FCB   $80
                FCB   $20
                FCB   $FF
                FCB   $FF
MF3
                FCB   $00
                FCB   $03
                FCB   $00
                FCB   $00
                FCB   $00
                FCB   $80
                FCB   $20
                FCB   $00
                FCB   $FF
MF4
                FCB   $01
                FCB   $03
                FCB   $00
                FCB   $20
                FCB   $FF
                FCB   $80
                FCB   $00
                FCB   $00
                FCB   $FF
MF5
                FCB   $02
                FCB   $03
                FCB   $00
                FCB   $20
                FCB   $00
                FCB   $80
                FCB   $20
                FCB   $FF
                FCB   $FF
MF6
                FCB   $00
                FCB   $03
                FCB   $00
                FCB   $00
                FCB   $00
                FCB   $80
                FCB   $20
                FCB   $00
                FCB   $FF
MF7
                FCB   $00
                FCB   $01
                FCB   $00
                FCB   $00
                FCB   $FF
                FCB   $FF
MF8
                FCB   $00
```

```
        FCB  $01
        FCB  $80
        FCB  $00
        FCB  $FF
        FCB  $FF
FIU
        JSR INIT
        LDAA INPUTS+$00
        STAA IN_BUF
        LDAB #$01
        JSR MF_SUB
        STAA TVLST+$00
        LDAB #$02
        JSR MF_SUB
        STAA TVLST+$01
        LDAB #$03
        JSR MF_SUB
        STAA TVLST+$02
        LDAA INPUTS+$01
        STAA IN_BUF
        LDAB #$04
        JSR MF_SUB
        STAA TVLST+$03
        LDAB #$05
        JSR MF_SUB
        STAA TVLST+$04
        LDAB #$06
        JSR MF_SUB
        STAA TVLST+$05
        LDAA TVLST+$02
        STAA AND_REG
        LDAA AND_REG
        STAA OR_REG
        LDAA TVLST+$05
        STAA AND_REG
        LDAA TVLST+$01
        JSR MIN1_OP
        JSR MAX2_OP
        LDAA TVLST+$05
        STAA AND_REG
        LDAA TVLST+$00
        JSR MIN1_OP
        JSR MAX2_OP
        LDAB #$00
        JSR DFZ_C
        LDAA TVLST+$03
        STAA AND_REG
        LDAA TVLST+$01
        JSR MIN1_OP
        LDAA AND_REG
        STAA OR_REG
        LDAA TVLST+$04
        STAA AND_REG
        LDAA TVLST+$01
        JSR MIN1_OP
        JSR MAX2_OP
        LDAA TVLST+$04
        STAA AND_REG
        LDAA TVLST+$00
        JSR MIN1_OP
        JSR MAX2_OP
        LDAB #$80
        JSR DFZ_C
        LDAA TVLST+$03
        STAA AND_REG
        LDAA TVLST+$00
        JSR MIN1_OP
        LDAA AND_REG
        STAA OR_REG
        LDAB #$FF
        JSR DFZ_C
        JSR DIV_SUB
        LDX #$00
        LDAA OUT_REG
        STAA OUTPUTS,X
        RTS
MIN1_OP   CMPA AND_REG
        BCC MIN1_E
        STAA AND_REG
MIN1_E    RTS
MAX2_OP   LDAA AND_REG
          CMPA OR_REG
          BCS MAX2_E
          STAA OR_REG
MAX2_E    RTS
DFZ_C     LDAA OR_REG
          MUL
          TAB
          CLRA
          ADDD TUH_REG
          STD TUH_REG
          CLRA
          LDAB OR_REG
          ADDD SUH_REG
          STD SUH_REG
          RTS
DIV_SUB   LDD  TUH_REG
          LDX  SUH_REG
          FDIV
          BCS DIV_1
          XGDX
          STAA OUT_REG
          RTS
DIV_1     CLR OUT_REG
          RTS
INIT      CLR TUL_REG
        CLR TUH_REG
        CLR SUL_REG
        CLR SUH_REG
        RTS
MF_SUB    ASLB
        SUBB #$2
        CLRA
        ADDD #MF_OFF
        XGDX
        LDX $0,X
        LDAA $0,X
        STAA MF1_REG
        INX
        LDAA $0,X
        STAA MF2_REG
        LSL MF1_REG
        LSL MF2_REG
        INX
MF_1      XGDX
        ADDD #$3
        XGDX
        LDAA $0,X
        LSR MF1_REG
        LSR MF2_REG
        CMPA IN_BUF
        BLO MF_1
        LSR MF1_REG
        BCS MF_2
        JSR MF_S2
        ADDB MF1_REG
        BCC MF_4
        LDAB #$FF
MF_4      TBA
        RTS
MF_S2     DEX
        LDAA $0,X
        STAA MF1_REG
        LSR MF2_REG
        BCC MF_3
        JSR MF_S1
        LSRD
        LSRD
        LSRD
        LSRD
        TSTA
        BEQ MF_7
        LDAB #$FF
MF_7      RTS
MF_3      JSR MF_S1
        TSTA
        BEQ MF_5
        LDAB #$FF
MF_5      RTS
MF_S1     DEX
        LDAA $0,X
        STAA MF2_REG
          LDAA IN_BUF
          DEX
          SUBA $0,X
          LDAB MF2_REG
        MUL
          RTS
MF_2      JSR MF_S2
          STAB MF2_REG
          LDAA MF1_REG
          SUBA MF2_REG
          BCC MF_6
          LDAA #$0
MF_6      RTS
      RTS


*Subroutine OUTPUT - Used to send desired path values to the *motors
OUTPUT      LDAA  $B7FF,X           Load Delta Y value
      INX               Increment position counter
      PSHA              Put Delta Y on stack
      LDD   #$0000            #2 and #3 bit of Delta y
      PSHD
      LDAA  #$00        #1 bit of Delta y
      PSHA
      LDD   #$0000            #3 and #4 bits of velocity
      PSHD
      LDAA  $VELOC+03   #2 bit of velocity
      PSHA
      LDAA  $VELOC+02   #1 bit of velocity
      PSHA
      LDD   #$0009            Motor control word
      PSHD
      LDAA  #$0A        # of bits to be sent
      PSHA
      LDAA  #$1F        Load Trajectory Command
      PSHA
      LDAA  $B7FF,X           Load Delta X value
      INX               Increment position counter
      PSHA              Put Delta X on stack
      LDD   #$0000            #2 and #3 bit of Delta x
      PSHD
      LDAA  #$00        #1 bit of Delta x
      PSHA
      LDD   #$0000            #3 and #4 bits of velocity
      PSHD
      LDAA  $VELOC+01   #2 bit of velocity
      PSHA
      LDAA  $VELOC+00   #1 bit of velocity
      PSHA
      LDD   #$0009            Motor control word
      PSHD
      LDAA  #$0A        # of data bits to be sent
      PSHA
      LDAA  #$1F        Load Trajectory Command
      PSHA
      JSR   WRX         Write Load Traj. Comm to X
      JSR   WRDX        Write Traj. Data to X
      JSR   WRY         Write Load Traj. Comm to Y
      JSR   WRDY        Write Traj. Data to Y
      RTS

*Subroutine WRX - Used to write command to the x motor
WRX
      LDAA  #$xx        Set up LM628 configuration
      STAA  PORTB       Put it on port B
      JSR   WRCOM       Write the command
      RTS

*Subroutine WRY - Used to write command to the y motor
WRY
      LDAA  #$xx        Set up LM628 configuration
      STAA  PORTB       Put it on port B
      JSR   WRCOM       Write the command

      RTS
*Subroutine WRXY - Used to write a command to both x and y motors
WRXY
      JSR   WRX         Write command to x motor
      JSR   WRY         Write command to y motor
      RTS

*Subroutine WRDX - Used to write data to the x motor
WRDX
      LDAA  #$XX
      STAA  PORTB
      JSR   WRDATA
      RTS

*Subroutine WRDY - Used to write data to the y motor
WRDY
      LDAA  #$XX
      STAA  PORTB
      JSR   WRDATA
      RTS
```

```
*Subroutine WRSTR - Used to strobe the Write pin
WRSTR   BSET    $04;#$08        STROBE WRITE PIN
        BCLR    $04;#$10        CLEAR WRITE STROBE BIT
        RTS

*Subroutine BUSY - Used to test the busy bit
BUSY    LDAA    PORTB
        PSHA                    SAVE LM628 CONFIG
        LDAA    DDIRC
        PSHA                    SAVE PORT C CONFIG
        LDAA    #$00
        STAA    DDIRC           SET PORT C TO INPUT
        BCLR    PORTB;#$20      SET PS PIN TO LOW
TEST    JSR     RDSTR           STROBE READ PIN
        BRSET   PORTC;#$01,TEST TEST AGAIN IF BUSY BIT HIGH
        PULA
        STAA    DDIRC           RESTORE PORT C CONFIG
        PULA
        STAA    PORTB           RESTORE LM628 CONFIG
        RTS

* Subroutine WRCOM - Used to write a command to a LM628
WRCOM   PULB                    RETREIVE COMMAND
        STAB    PORTC           PUT COMMAND ON PORT C
        JSR     WRSTR           STROBE WRITE PIN
        JSR     BUSY            TEST BUSY BIT
        RTS
```

Appendix E. List of References

Hamid R. Berenji, "Fuzzy Logic Controllers," Sterling Federal Systems, Artificial Intelligence Research Branch, NASA Ames Research Center, January 24, 1991.

David Dale, "Application Note AN-706: LM628/LM629 User Guide," *Linear Applications Handbook*, National Semiconductor Corporation, 1990.

Gene F. Franklin, J. David Powell and Abbas Emami-Naeini, *Feedback Control of Dynamic Systems*, Third Edition, Addison-Wesley Publishing Company: New York, 1994.

"HC11: M68HC11 Reference Manual", Motorola, Incorporated, 1991.

Liang-Jong Huang and Masayoshi Tomizuka. "A Self-Paced Fuzzy Tracking Controller for Two-Dimensional Motion Control," *IEEE Transactions on Systems, Man, and Cybernetics*, Volume 20, Number 5, pp. 1115-1123, September/October 1990.

Steven Hunt, "Application Note AN-693: LM628 Programming Guide," *Linear Applications Handbook*, National Semiconductor Corporation, 1990.

Dr. Robert N. Lea and Yashvant Jani, Ph.D., "Fuzzy Logic Applications to Expert Systems and Control," Software Technology Branch/PT4, NASA Lyndon B. Johnson Space Center.

"LM628/LM629 Precision Motion Controller Data Sheet," *Power IC's. Data Book*, National Semiconductor Corporation, 1991.

Daniel G. Schwartz and George J. Klir, "Fuzzy Logic Flowers in Japan", *Technology Edge*, pp. 22-24, October 1992.

James M. Sibigtroth, “Implementing Fuzzy Expert Rules in Hardware,” *AI Expert*, April 1992.

Peter Spasov, Microcontroller Technology: The 68HC11, Regents/Prentice Hall: Englewood Cliffs, New Jersey, 1993.