University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

Billy the Amazing Broom Balancer
Final Written Report

Kristopher Anderson
December 9, 2003

TAs: Uriel Rodriguez, Louis Brandy
Instructor: Dr. A. Arroyo

1. Table of Contents

2. Abstract

Billy the Amazing Broom Balancer is a physical implementation of a solution to the inverted pendulum problem. Billy is fully autonomous; he requires no connection to a computer to control him and no external power connection. Billy uses a fuzzy controller to balance his pendulum.

3. Executive Summary

Billy the Amazing Broom Balancer is an autonomous robot designed to balance an inverted pendulum. He utilizes an Atmel Mega128 AVR microcontroller to control his movement. The balance algorithm is implemented with fuzzy logic.

The angle of the pendulum is measured with an incremental rotary encoder. This encoder performed well; I have written a special sensor report concerning its performance an implementation.

The design and construction of the platform proved much more difficult than making the software for Billy. In fact, Billy was not able to keep the pendulum balanced due to physical limitations; specifically, the motor is not able to provide enough power to bring the pendulum upright once it has gone beyond a certain angle.

4. Introduction

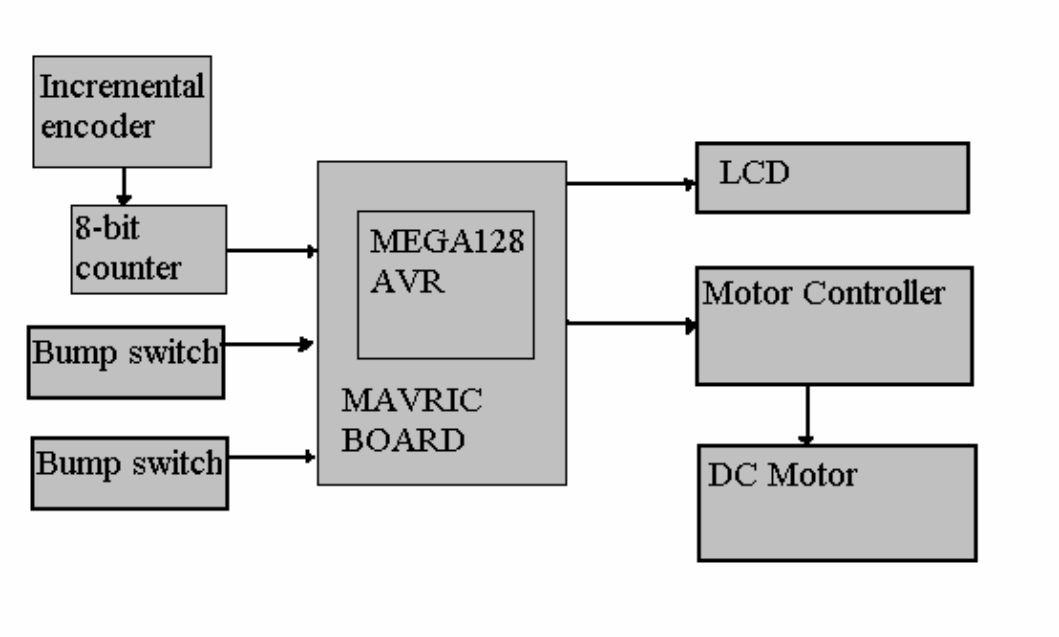
Balancing an inverted pendulum is a common and well studied problem in control theory. The inverted pendulum is attached to a cart, which moves in one dimension to keep the pendulum balanced. The problem is non linear and unstable.

Most physical implementations of this problem and its solution are controlled by an external computer. These implementations are also usually built on tracks with an external motor moving the platform. Billy is totally self-contained; he runs on batteries and is controlled by an on-board processor.

Billy is actually a cart with an inverted pendulum on top. He moves back and forth using a DC motor driven by batteries. Billy's control system consists of an Atmel AVR microcontroller.

This report gives an overview of Billy the Amazing Broom Balancer.

5. Integrated System



The control system has one external input, the angle of the pendulum. The output of the control system is the force Billy's motor must exert to move either forward or backwards. Past input data will be used to determine the pendulum's angular velocity. These two values are used by the control system to determine the output.

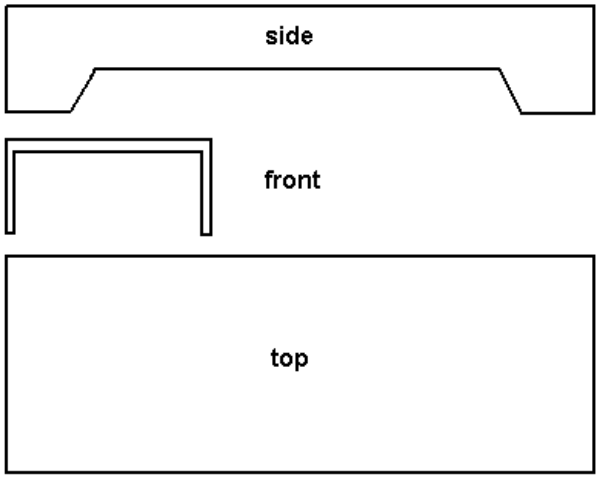
A fuzzy control system is used to balance the pendulum. This system will be explained in more detail later in the report.

Billy uses a BDmicro MAVRIC board with an Atmel Mega128 processor. The Processor has 4K EEPROM and 4K SRAM on the chip. The MAVRIC board has an additional 128K SRAM, but that memory is not used for this project; the memory expansion ports are used for I/O. The Atmel AVR Mega128 can run at up to 16MHz; Billy's processor runs at 14.7 MHz. This processor provides ample memory and speed for this application.

6. Mobile Platform

A platform is required that is strong enough to handle the forces generated by the quick back and forth movement necessary to balance the pendulum. A large motor and heavy batteries are required for Billy to be fast enough to keep the pendulum balanced.

Billy's main platform is made from plywood [A1]. Figure 1 shows a sketch of Billy's main platform.



Designing the mobile platform was kind of a paradox. A stronger frame is required to hold the motor and batteries and to stand up to the large forces; but as the frame gets larger and stronger, more batteries and a larger motor are required to move it. I chose to build the platform out of plywood for its low cost and because it is easy to work with. A strong plastic would most likely provide better performance.

The motor needs to turn one of the axles to move the robot. Billy's motor turns the axle through gears [A9]. The gears are the part of the system that must handle the most stress;

they must also fit together precisely. I would suggest to someone using a similar design to have a few spare sets of gears and a way to easily change them. Acceleration is more important then top speed, so I would also suggest a high gear ratio. Billy's gear ratio is 5:2, but 5:1 might provide better results.

7. Actuation

To balance the pendulum, Billy must be able to move in two directions: forward and backward. Movement in more than one dimension is not desirable, as it adds to the complexity of the problem.

Billy's 12V DC motor is controlled by a forward/reverse-proportional DC motor controller [A4]. This controller has a fairly high output capability, but it proved to be too little for this application. During a test run, two of the MOSFETs in the H-bridge were blown. They were replaced with better ones, and heat sinks were installed on both sides of the MOSFETs.

8. Sensors

Billy will need to measure 2 outside variables

1. The angle of the pendulum
2. If the pendulum has swung too far

The angle of the pendulum is measured by a rotational encoder [A5]. The encoder sends out pulses as it turns. These pulses are converted into clock pulses by a quadrature to up/down counter converter chip [A6] which are then used by two 4-bit up/down counters

[A7] to measure the angle of the pendulum. The encoder has 2048 positions, but the pendulum can only swing in a small arc so not all of the positions are used.

The encoder's index is positioned so that it is active when the pendulum is approximately straight up. The index resets the counters when it is active. This helps to keep the counters accurate, because the index position is passed often during balancing. More information on the encoder can be found in my special sensor report.

There are bump switches on the supports on either side of the pendulum. These switches are pressed when the pendulum leans on them. If the pendulum is leaning on one of these switches, it has traveled too far and Billy needs to start over. The switches create an interrupt when they are pressed.

Some additional sensors would have been useful for this design, but were not implemented. Billy will collide with any objects in his path while he is balancing the pendulum. Infrared or sonar sensors that detect obstacles in Billy's path would solve this problem.

9. Behaviors

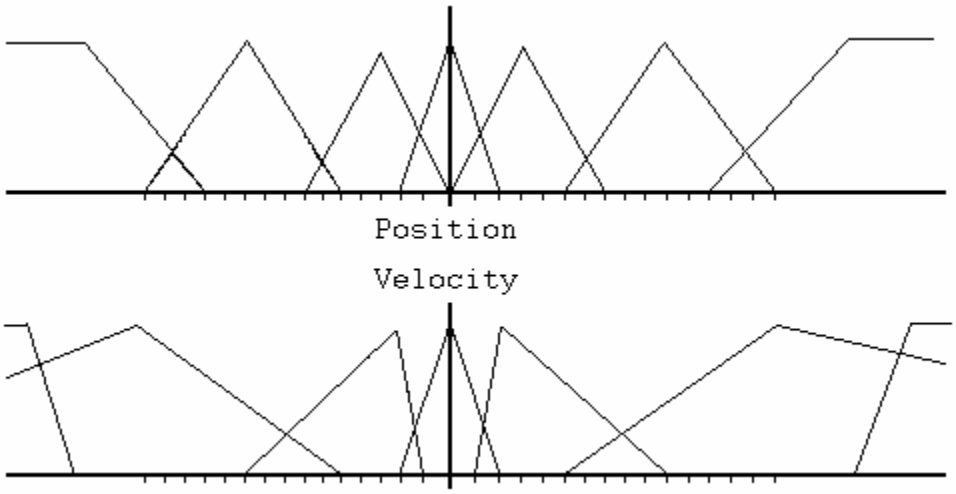
Before he can start balancing the pendulum, Billy needs to know when the pendulum is straight up. The index output of the encoder is useful here. When the pendulum is approximately straight up, the index is active and the counters are reset. This gives Billy a reference angle. When Billy starts up, he asks you to balance the pendulum so that he can calibrate the balance position. As the pendulum is balanced for five seconds, the position of the pendulum (the counter values) are measured and averaged together. This

value is remembered and when it is subtracted from the position value of the pendulum it gives the correct value.

Note that the angles and velocities are not in any specific units, they are just the measurements from the sensors.

Angle measurements are taken approximately every 4ms. The velocity is calculated by differentiating the angle values.

To keep the pendulum balanced, a fuzzy controller is used. The membership functions are shown in the figure below.



The percent power to send to the motors is determined from these two functions. The membership value in each category is then used with the table below to determine the power.

	Velocities						
-100	-100	-100	-100	-100	-100	-100	Positions
-100	-95	-90	-85	-80	-60	-50	
-85	-75	-50	-35	-15	0	10	

-70	-40	-5	0	10	45	75
-5	0	20	40	55	80	90
50	60	85	95	100	100	100
100	100	100	100	100	100	100

This is done as follows:

1. Make two arrays that are as long as the number of memberships in each category. One array is for position and one is for velocity. So each array has a length of seven.
2. Fill each array with the membership values. For example, if the velocity is 7, the velocity array would be [0.0 0.0 0.0 0.0 0.3 0.2].
3. Multiply the matrix above by each array, and then add all of the values in the result. This gives the weighted average.

This algorithm takes about 3 milliseconds to run. The power delivered to the motor is updated every 35 milliseconds. This is about twice the length of the PWM period.

The code in Appendix B will probably explain the algorithm better.

10. Results

Billy can almost balance the pendulum. The motors do not provide enough power to keep the pendulum balanced very long. The voltage regulator circuit and the motor driver start to overheat fairly rapidly. I plan to remove the regulator circuit and see how Billy performs.

11. Conclusion

Before I started work on this project, I expected to spend more time programming the balance algorithms than building the robot. I actually spent much more time building the robot.

12. Advice

Anyone attempting this project in the future should focus on first designing a strong but lightweight platform. This platform should, like Billy's platform, be large; about 18"x10". A tall pendulum, about four or five feet, will make the task of balancing it easier. Be sure to use a lightweight metal, such as aluminum, for the pendulum. Wood is not acceptable as it will bend. A weight at the end of the pendulum is necessary as well. Try to get strong gears; they need to withstand a lot of stress.

The motor should be as powerful as possible. Something greater than 12 volts is recommended. The motor driver is a critical piece of the design. It will need to be actively cooled (with a fan). Don't be surprised if you blow a 60 amp MOSFET. You might even want to replace the MOSFETs in your motor driver with ones designed to handle 100 amps or more.

Something has to provide all of this power. Billy uses four 7.2V NiMH batteries; two parallel pairs are connected in series to provide 14.4V. If possible, you might even want to use six batteries. There's a reason that most physical implementations of the inverted pendulum problem are externally powered. The batteries add weight, but they are necessary. **Be sure to isolate the motor circuit from the other electronics.** The abrupt changes in the motors will create voltage spikes.

Billy has a 12 volt regulator circuit between the batteries and the motor. This is because the voltage should be constant so that the controller will know what PWM value corresponds to what acceleration. It is kind of a waste of battery power though, as it cuts the voltage and it requires a fan to cool it. Maybe there is some way that this circuit

could be omitted. You could just assume fully charged batteries, or you could monitor the voltage somehow. The latter solution would be difficult due to the spikes induced by the motor; you wouldn't want to just connect the battery output to the A/D converter on your processor.

Now to the software; if you planned well and built a ~~good~~ outstanding platform, this should be easy. You could make it hard by trying to model the robot on your computer and then using nonlinear control theory. A fuzzy controller is a much better way to go.

12. Appendices

Appendix A: Billy's parts and prices

	Description	Purchased From	Price
1.	¼" plywood.	Lowes	$10.00
2.	12v DC motor. (Unknown manufacturer)	EBay	$6.00
3.	4 Green Dot Sumo Tires and hubs	www.lynxmotion.com	$30.00
4.	Vantec RET 411	www.lynxmotion.com	$50.00
5.	Rotary Encoder S2-2048-IB	US Digital	$78.00
6.	Quadrature to up/down counter LS7083	US Digital	$3.00
7.	2 4-bit up/down counters	Texas Instruments	Free
8.	2 small lever switches	Radio Shack	$5.00
9.	Plastic Gears	www.jameco.com	$8.00
10.	MAVRIC AVR board	www.bdmicro.com	$109.00
11.	2 5V voltage regulators	Texas Instruments	Free
12.	4 12V (5A) voltage regulators	Texas Instruments	Free
13.	6 TO-220 heat sinks	Radio Shack	$10.00
14.	Various Dowels	Lowes	$10.00
15.	Aluminum rod (pendulum)	Lowes	$6.00
16.	2 Threaded rods (axles)	Lowes	$5.00
17.	16X2 LCD	Home	Free
18.	Ribbon Cable	IMDL Lab	Free
19.	Male and Female Headers	www.jameco.com	$12.00
20.	AA battery holder	Radio Shack	$3.00
21.	8 AA NiMH batteries	Wal-Mart	$15.00
22.	4 3000mAh 7.2V NiMH battery packs	www.batteryspace.com	$40.00
23.	4 6mm bearings	www.towerhobbies.com	$8.00
		Total	
			$408.00

Appendix B: Source code

```
//#DEFINES:
#define        ANGLE_PORT            PINC
#define        MOTOR_MAX_POS         160
#define        MOTOR_MIN_POS         96
#define        MOTOR_STOP            88
#define        MOTOR_MIN_NEG         75
#define        MOTOR_MAX_NEG         1
#define        ANGLE_ARRAY_SIZE      20
#define        NUM_POS_MEM           7
#define        NUM_VEL_MEM           7


/*****************************************
Chip type           : ATmega128
Program type        : Application
Clock frequency     : 14.745600 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 1024
*****************************************/

#include <mega128.h>

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x1B
#endasm
#include <lcd.h>
#include <stdio.h>
#include <delay.h>

/******************* Global variables ***************************************/
char angle_bias;              // This is the bias to account for floor tilt, etc.
char motor_speed = 0;         // This is the speed of the motor [-100 to 100]
char prev_speed = 0;          // This is the previous speed of the motor
char curr_angle = 0;          // This is the current angle of the pendulum
char ang_velocity = 0;        // The angular velocity of the pendulum
char current_angle_index = 0; //
unsigned char angle_port_var; // This is updated with the value of the angle port
bit accepting_bump_int = 0;   // 1=true, 0=false
bit pendulum_rest_side = 0;   // The last side the pendulum was on [0=FAR, 1=NEAR]

char pendulum_angles[ANGLE_ARRAY_SIZE];// These are the 20 most recent angle measurements

char lcd_buffer[33];          // The LCD buffer


int pos_member_funcs[NUM_POS_MEM][3] =       {{-9500,-20,-12},
                                              {-15,-10, -5},
                                              { -7, -3,  0},
                                              { -2,  0,  2},
                                              {  0,  3,  7},
                                              {  5, 10, 15},
                                              { 12, 20, 9500}
                                             };
int vel_member_funcs[NUM_VEL_MEM][3] =               {{-9500,-35,-20},
                                              {-30,-15, -5},
                                              {-10, -2, -1},
                                              { -2,  0,  2},
                                              {  1,  2, 10},
                                              {  5, 15, 30},
                                              { 20, 35, 9500}
                                             };
int speed_matrix[NUM_POS_MEM][NUM_VEL_MEM] =         {
                                              {-100,-100,-100,-100,-100,-100,-100},
                                              {-100, -95, -90, -85, -80, -60, -50},
                                              { -85, -75, -50, -35, -15,   0,  10},
```

```
                                                { -70, -40,  -5,   0,  10,  45,  75},
                                                {  -5,   0,  20,  40,  55,  80,  90},
                                                {  50,  60,  85,  95, 100, 100, 100},
                                                { 100, 100, 100, 100, 100, 100, 100}
                                               };

char angle_conv[256];  // This array is used to convert from the angle sensor
                       // reading to the angle reading.  It is initialized in main.
/*****************************************************************************/

/******************* Function prototypes *************************************/
//UPDATE_ANGLE;
interrupt [TIM1_OVF] void timer1_ovf_isr(void);
interrupt [EXT_INT4] void ext_int4_isr(void);
interrupt [EXT_INT5] void ext_int5_isr(void);
void set_bias(void);
char average(char samples[], unsigned char num_samples);
void update_motor(void);
void swing_pendulum_up(void);
void update_ang_velocity(void);
void calculate_speed(void);
/*****************************************************************************/

/********************** UPDATE_ANGLE *****************************************/

// Function to convert ANGLE_PORT (angle) value into a valid angle

// I made it a macro because it is called often, and it doesn't return anything

#define UPDATE_ANGLE  \
                      angle_port_var = ANGLE_PORT;\
                      curr_angle = angle_conv[angle_port_var] - angle_bias;
/*****************************************************************************/

/************************** TIMER1_OVF_ISR ***********************************/

// Timer 1 overflow interrupt service routine  (Updates angle array)

#pragma savereg-
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
        #asm
                push r26
                push r27
                push r30
                push r31
                in   r30,SREG
                push r30
        #endasm
        UPDATE_ANGLE
        pendulum_angles[current_angle_index] = curr_angle;
        current_angle_index++;
        if(current_angle_index == ANGLE_ARRAY_SIZE)
        {
                current_angle_index = 0;
        }
        #asm
                pop r30
                out SREG,r30
                pop r31
                pop r30
                pop r27
                pop r26
        #endasm
}
#pragma savereg+

/*****************************************************************************/

/************************** EXT_INT4_ISR *************************************/
```

```
// External Interrupt 4 service routine (Far side bump switch)

interrupt [EXT_INT4] void ext_int4_isr(void)
{
        unsigned char i;

        pendulum_rest_side = 0;

        if(accepting_bump_int == 1)
        {
                accepting_bump_int = 0;

                motor_speed = 0;
                update_motor();

                lcd_clear();
                delay_ms(20);
                lcd_putsf("Pendulum hit far");
                delay_ms(2000);

                lcd_clear();
                for(i=5; i>0; i--)
                {
                        sprintf(lcd_buffer, "Trying Again %2d", i);
                        lcd_gotoxy(0,0);
                        lcd_puts(lcd_buffer);
                        delay_ms(1000);
                }

                swing_pendulum_up();
         }
}
/*****************************************************************************/

/************************* EXT_INT5_ISR ***************************************/

// External Interrupt 5 service routine (Near side bump switch)

interrupt [EXT_INT5] void ext_int5_isr(void)
{
        unsigned char i;

        pendulum_rest_side = 1;

        if(accepting_bump_int == 1)
        {
                accepting_bump_int = 0;

                motor_speed = 0;
                update_motor();

                lcd_clear();
                delay_ms(20);
                lcd_putsf("Pendulum hit near");
                delay_ms(2000);

                lcd_clear();
                for(i=5; i>0; i--)
                {
                        sprintf(lcd_buffer, "Trying Again %2d", i);
                        lcd_gotoxy(0,0);
                        lcd_puts(lcd_buffer);
                        delay_ms(1000);
                }

                swing_pendulum_up();
         }
}
/*****************************************************************************/

/******************* SET_BIAS ************************************************/
```

```
// This function is called on reset to determine what is straight up

void set_bias(void)
{
        unsigned char i, j;
        char samples[100];

        delay_ms(20);
        lcd_clear();
        delay_ms(20);
        lcd_putsf("Balance pendulum     please..");

        delay_ms(3000);

        lcd_clear();
        delay_ms(20);

        for(i=5; i>0; i--)
        {
                sprintf(lcd_buffer, "Calibrating %2d", i);
                lcd_gotoxy(0,0);
                lcd_puts(lcd_buffer);
                for(j=0; j<10; j++)
                {
                        UPDATE_ANGLE
                        samples[(10-i)*10+j] = curr_angle;
                        delay_ms(100);
                }
        }

        angle_bias = average(samples, 100);
        lcd_clear();
        sprintf(lcd_buffer, "Bias = %2d", angle_bias);
        lcd_puts(lcd_buffer);
        delay_ms(1000);
        lcd_clear();


}
/*****************************************************************************/

/************************** AVERAGE ******************************************/

// This function returns the average of the array

char average(char samples[], unsigned char num_samples)
{
        unsigned char i;
        int total = 0;

        for(i=0;i<num_samples;i++)
        {
                total += samples[i];
        }

        return ((char)(total / num_samples));
}
/*****************************************************************************/

/************************** UPDATE_MOTOR *************************************/

// This function converts the global variable 'motor_speed' into a PWM value and
// sets it

void update_motor(void)
{
        float m_speed;
        float neg_diff;
        float pos_diff;
```

```
        m_speed = motor_speed + 0.0;
        neg_diff = (MOTOR_MIN_NEG-MOTOR_MAX_NEG) + 0.0;
        pos_diff = (MOTOR_MAX_POS-MOTOR_MIN_POS) + 0.0;

        if((prev_speed > 0 && motor_speed < 0) || (prev_speed < 0 && motor_speed > 0))
        {
                prev_speed = 0;
                OCR0 = MOTOR_STOP;
                return;
        }
        if(motor_speed == 0)
        {
                prev_speed = 0;
                OCR0 = MOTOR_STOP;
                return;
        }
        if(motor_speed > 0)
        {
                prev_speed = motor_speed;
                OCR0 = (unsigned char) MOTOR_MIN_NEG - (unsigned
char)(neg_diff*(m_speed/100.0));
                return;
        }
        if(motor_speed < 0)
        {
                prev_speed = motor_speed;
                OCR0 = (unsigned char) MOTOR_MIN_POS + (unsigned char)(pos_diff*(-
m_speed/100.0));
                return;
        }


//      OCR0 = MOTOR_STOP + curr_angle;


}
/*****************************************************************************/

/************************* SWING_PENDULUM_UP ***********************************/

// This function gets the robot to swing the pendulum from resting on one of
// The bump switches to balancing.  It uses 'pendulum_rest_side' to determine
// which way to go.  NOT IMPLEMENTED

void swing_pendulum_up(void)
{

        lcd_clear();
        lcd_putsf("Ready");


        delay_ms(1000);


        lcd_clear();
        accepting_bump_int = 1;
}
/*****************************************************************************/

/************************ UPDATE_ANG_VELOCITY ***********************************/

//

void update_ang_velocity(void)
{
        char index_1_a;
        char index_1_b;
        char index_2_a;
        // and current_angle_index

        index_1_a = (current_angle_index-1);
```

```
        index_1_b = (current_angle_index+1);
        index_2_a = (current_angle_index-2);

        // Wrap around the array
        if(index_1_a == -1)
        {
                index_1_a = (ANGLE_ARRAY_SIZE-1);
                index_2_a = (ANGLE_ARRAY_SIZE-2);
        }
        else if(index_2_a == -1)
        {
                index_2_a = (ANGLE_ARRAY_SIZE-1);
        }
        else if(index_1_b == ANGLE_ARRAY_SIZE)
        {
                index_1_b = 0;
        }

        ang_velocity = ((pendulum_angles[index_1_a] -
pendulum_angles[index_1_b])+(pendulum_angles[index_2_a] -
pendulum_angles[current_angle_index]))/2;
}
/*****************************************************************************/

/************************* CALCULATE_SPEED ***********************************/

// This is the big function that implements the fuzzy controller

void calculate_speed(void)
{
        char i,j;

        float curr_pos_mem, curr_vel_mem;

        float vel_memberships[NUM_VEL_MEM];
        float pos_memberships[NUM_POS_MEM];
        float speed_m[NUM_POS_MEM][NUM_VEL_MEM];
        float total_speed;
        float total_vel, total_pos;

        // Disable angle measurement interrupts
        TIMSK=0x00;

        UPDATE_ANGLE
        update_ang_velocity();

        // Get memberships in velocity and position
        for(i=0; i< NUM_VEL_MEM; i++)
        {
                if(ang_velocity >= vel_member_funcs[i][0] && ang_velocity <=
vel_member_funcs[i][2])
                {
                        if(ang_velocity < vel_member_funcs[i][1])
                        {
                                vel_memberships[i] = (float)((1000*(ang_velocity-
vel_member_funcs[i][0]))/(vel_member_funcs[i][1]-vel_member_funcs[i][0]));
                        }
                        else
                        {
                                vel_memberships[i] = (float)((1000*(vel_member_funcs[i][2]-
ang_velocity))/(vel_member_funcs[i][2]-vel_member_funcs[i][1]));
                        }
                }
                else
                {
                        vel_memberships[i] = 0.0;
                }
        }
        for(i=0; i< NUM_POS_MEM; i++)
        {
```

```
                if(curr_angle >= pos_member_funcs[i][0] && curr_angle <=
pos_member_funcs[i][2])
                {
                        if(curr_angle < pos_member_funcs[i][1])
                        {
                                pos_memberships[i] = (float)((1000*(curr_angle-
pos_member_funcs[i][0]))/(pos_member_funcs[i][1]-pos_member_funcs[i][0]));
                        }
                        else
                        {
                                pos_memberships[i] = (float)((1000*(pos_member_funcs[i][2]-
curr_angle))/(pos_member_funcs[i][2]-pos_member_funcs[i][1]));
                        }
                }
                else
                {
                        pos_memberships[i] = 0.0;
                }
        }

        // Normalize the memberships
        total_pos = 0.0;
        for(i=0; i<NUM_POS_MEM; i++)
        {
                total_pos += pos_memberships[i];
        }
        total_vel = 0.0;
        for(i=0; i<NUM_VEL_MEM; i++)
        {
                total_vel += vel_memberships[i];
        }
        for(i=0; i<NUM_POS_MEM; i++)
        {
                pos_memberships[i] /= total_pos;
        }
        for(i=0; i<NUM_VEL_MEM; i++)
        {
                vel_memberships[i] /= total_vel;
        }

        // Get the speed from the matrix
        for(i=0; i<NUM_POS_MEM; i++)
        {
                curr_pos_mem = pos_memberships[i];
                for(j=0; j<NUM_VEL_MEM; j++)
                {
                        speed_m[i][j] = speed_matrix[i][j] * curr_pos_mem;
                }
        }
        total_speed = 0.0;
        for(i=0; i<NUM_VEL_MEM; i++)
        {
                curr_vel_mem = vel_memberships[i];
                for(j=0; j<NUM_POS_MEM; j++)
                {
                        total_speed += speed_m[j][i] * curr_vel_mem;
                }
        }

        motor_speed = (short) total_speed;

        // Enable angle measurement interrupts
        TIMSK=0x04;
}
/*****************************************************************************/

void main(void)
{
        // local variables
        unsigned char i;
```

```
        // Input/Output Ports initialization
        // Port A initialization
        // Func0=Out Func1=Out Func2=Out Func3=Out Func4=Out Func5=Out Func6=Out
Func7=Out
        // State0=0 State1=0 State2=0 State3=0 State4=0 State5=0 State6=0 State7=0
        PORTA=0x00;
        DDRA=0xFF;

        // Port B initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=Out Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=0 State5=T State6=T State7=T
        PORTB=0x00;
        DDRB=0x10;

        // Port C initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        PORTC=0x00;
        DDRC=0x00;

        // Port D initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        PORTD=0x00;
        DDRD=0x00;

        // Port E initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        PORTE=0x00;
        DDRE=0x00;

        // Port F initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In Func5=In Func6=In Func7=In
        // State0=T State1=T State2=T State3=T State4=T State5=T State6=T State7=T
        PORTF=0x00;
        DDRF=0x00;

        // Port G initialization
        // Func0=In Func1=In Func2=In Func3=In Func4=In
        // State0=T State1=T State2=T State3=T State4=T
        PORTG=0x00;
        DDRG=0x00;

        // External Interrupt(s) initialization
        // INT0: Off
        // INT1: Off
        // INT2: Off
        // INT3: Off
        // INT4: On
        // INT4 Mode: Low level
        // INT5: On
        // INT5 Mode: Low level
        // INT6: Off
        // INT7: Off
        EICRA=0x00;
        EICRB=0x00;
        EIMSK=0x30;
        EIFR=0x30;


       // Timer/Counter 0 initialization
        // Clock source: TOSC1 pin
        // Clock value: TOSC1/128
        // Mode: Fast PWM top=FFh
        // OC0 output: Non-Inverted PWM
        //ASSR=0x08;
        //TCCR0=0x6D;
        //TCNT0=0x00;
        //OCR0=0x00;
        TCCR0=0x6D;
```

```
OCR0 = MOTOR_STOP;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 14745.600 kHz
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x00;
TCCR1B=0x01;
TCNT1H=0x00;
TCNT1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x04;
ETIMSK=0x00;



// LCD module initialization
lcd_init(16);
lcd_clear();
lcd_putsf("Initialize angle array");

// Initialize the array that holds the angle conversions
for(i=0; i <= 127; i++)
{
       angle_conv[i] = i;
}
angle_bias = -128; // Just used temporarily, it gets set later
for(i=128; i < 255; i++)
{
       angle_conv[i] = angle_bias;
       angle_bias++;
}
angle_conv[255] = angle_bias;

lcd_clear();
lcd_putsf("Hi!");
delay_ms(1000);


// First find the balance point
set_bias();

// Count down to start
lcd_clear();
for(i=3; i>0; i--)
{
       sprintf(lcd_buffer, "Starting in %2d", i);
       lcd_gotoxy(0,0);
       lcd_puts(lcd_buffer);
       delay_ms(1000);
}

// Initialize the angle array
for(i=0; i<ANGLE_ARRAY_SIZE; i++)
{
       UPDATE_ANGLE
       pendulum_angles[i] = curr_angle;
}
```

```
        current_angle_index = 0;

        // Global enable interrupts
        #asm("sei")

        // Now swing the pendulum up
        swing_pendulum_up();

        // The main loop
        while (1)
        {
               sprintf(lcd_buffer,"ang=%3d vel=%4d", curr_angle, ang_velocity);
               lcd_gotoxy(0,0);
               lcd_puts(lcd_buffer);
               i = OCR0;
               sprintf(lcd_buffer,"motr=%3d pwm=%3d      ", motor_speed, i);
               lcd_gotoxy(0,1);
               lcd_puts(lcd_buffer);


               delay_ms(30);

               calculate_speed();


               update_motor();
        }
}
```