Floating Ball Using Fuzzy Logic Controller

Abdullah Alrashedi

Ahmad Alghanim

Iris Tsai

Tareq Alduwailah

Fahad Alsaqer

Mohammad Alkandari

Jasem Alrabeeh

Sponsored by:
Dr. Ruting Jia

***Abstract*— Floating ball controller is about floating a ball in the middle of a plastic tube using a computer cooling fan. The controller measures speed and distance using ultrasonic range sensor. This controller uses fuzzy logic rules to control the fan. This project helps could be applied to industrial applications like car speed cruiser.**

Keywords—Fuzzy Logic, Pulse width modulation, Membership function.

I. Introduction

Fuzzy logic is the operation where we use valued logic instead of the binary logic that uses only 0 and 1. While words are inherently less precise than numbers, their use is closer to what would be considered a human logic. A basic concept of fuzzy logic is using the if-then rule to describe system rules. For example, if the temperature is very hot then the fan should blow very hard. We will be using MATLAB to design a fuzzy logic system that help us to simulate our fuzzy logic design for the floating ball project. To test this, we are building and programming different parts of the project and they all should reflect the design we made using the MATLAB Fuzzy Toolbox. The project consisted of several elements including an ultrasonic sensor, a plastic tube with ball and an Arduino. A 3D printed adaptor shall be used to connect the fan with the tube. This floating ball project could be used as a source for other projects like car speed cruiser.

II. Description of the Project

A. Background Reason and Problem Statement

Using the fuzzy logic in desiging a floating ball system helps to make the system more accurate and precise. It also gives more flixability by using multiple inputs and rules to get the desired output. This floating ball project is a simple case of how fuzzy logic could be used in solving a lot of problems such as: car speed cruiser which has the same idea as the floating ball project. Using fuzzy logic to design this controller helps to choose different membership functions for the input and the output ,and we can control the interval of the inputs and the output.

B. High Level Requirments

This floating ball controller must be built using a fuzzy logic approach with fuzzy membership functions. It also must be with at least two inputs and one output controller. We used Arduino as the controller, a computer cooling fan as the blower, and an ultrasonic sensor for speed and distance detection, as shown in figure 1.1. Figure 1.1 shows the block diagram of the project regarding the levitation of a ping-pong ball. It consists of a hollow tube and the ball will be residing inside of it. At the top of the tube, there is ultrasonic sensor that will send ultrasonic sound to sense the distance of the ball from the top. At the bottom of the tube, there is a four-pin computer fan that will provide the wind necessary to flow through the tube and blow ping pong ball upward. The challenge is to get and keep the ping pong ball at the middle of the tube. There is an Arduino UNO board equipped with an AVR microcontroller to control the whole system. The control operation includes the distance measurement and speed of the fan to keep the ping pong ball levitated.

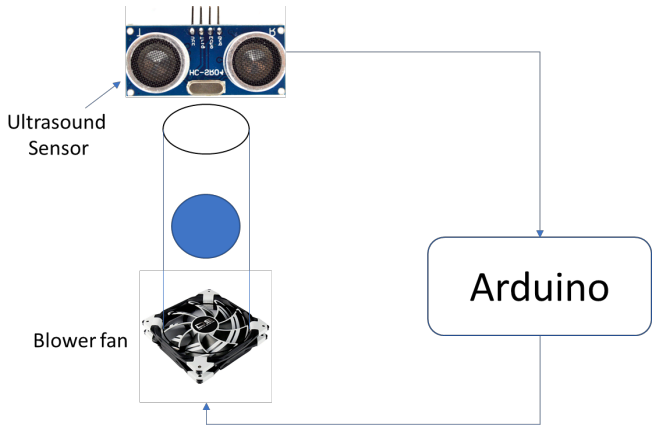

Figure 1.1: Block Diagram for the project

III. Design Approach

A. MATLAB Fuzzy Logic Toolbox:

First, we used the MATLAB fuzzy logic toolbox as a tool to create our design. We used distance (position), and speed of the ball as inputs for the controller, as shown in figure 1.2. Then we defined the membership functions, or the function that defines the membership value of the input as seen in figures 1.3 and 1.4. After this we created the rules for fuzzy logic which would decide what the out would be depending on the inputs. Thesee rules were made using Boolean logic, as seen in figure 1.5. The output is the pulse width modulation (PWM) that controls the fan, as shown in figure

1.6.

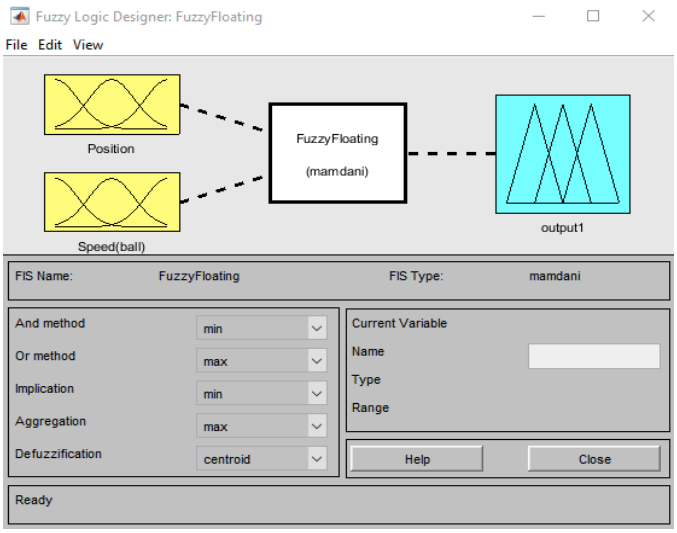

Figure 1. 2 Block Diagram

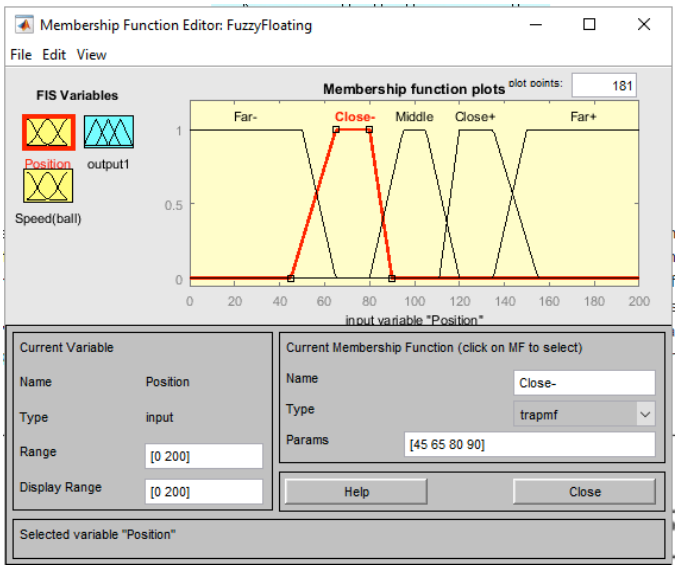

Figure 1.3 Position inputs for the system.

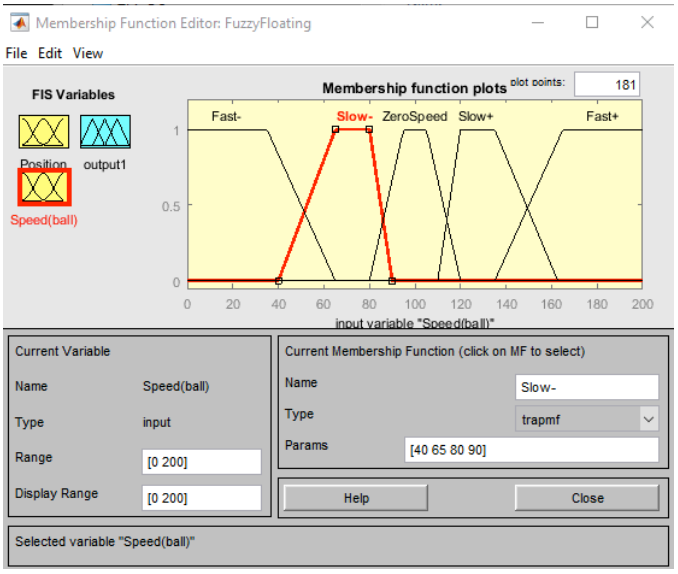

Figure 1. 4 Speed inputs for the system.

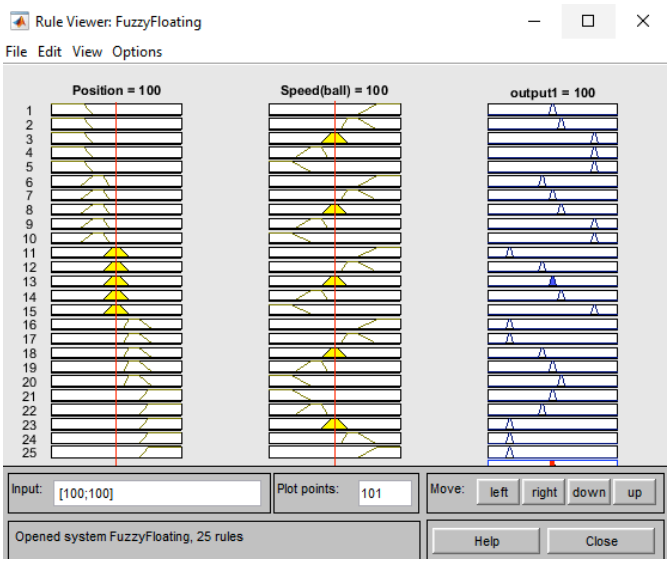

Figure 1. 5

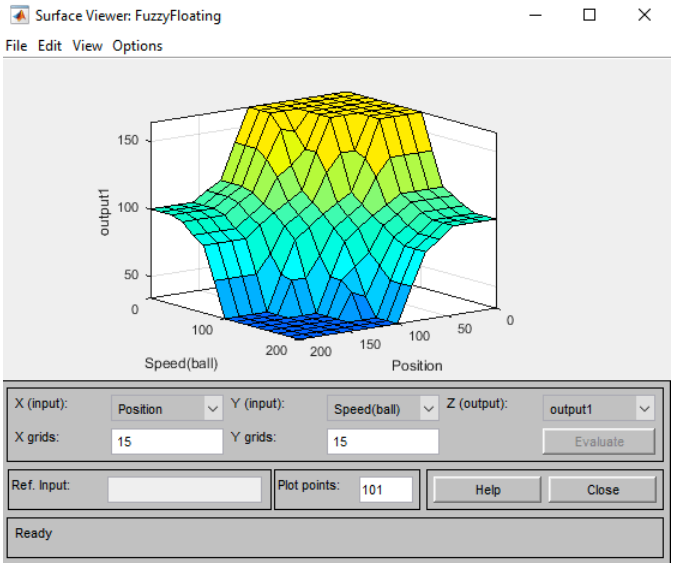

Figure 1. 6 Simulation results using MATLAB Fuzzy Interference.

	far-	close-	middle	close+	far+
fast+	zero	low-	high-	high-	high-
slow+	low+	zero	low-	high-	high-
zerospeed	high+	low+	zero	low-	high-
slow-	high+	high+	low+	zero	low-
fast-	high+	high+	high+	low+	zero

Table 1.1: Fuzzy Logic Table for the floating ball project.

B. *Project Parts:*

- Ultrasound Sensor (MCSR04):

Ultrasonic sensor emits an ultrasound at 40 000 Hz and then it travels through the air and if there is an obstacle on its path, the ultrasound will return back to the module. Based on the travelling time and the distance of the ball, we can calculate the speed.

Figure 1.7: Ultrasonic sensor sends ultrasound that reflects to sense the distance.

Here we used a HC-SR04 Ultrasonic Module which has 4 pins, Ground, VCC, Trig and Echo. The GND and the VCC pins of the sensor needs to be connected to the Ground and the 5 Volts pins on the Arduino Board respectively and the trig and echo pins to any Digital I/O pin on the Arduino Board.

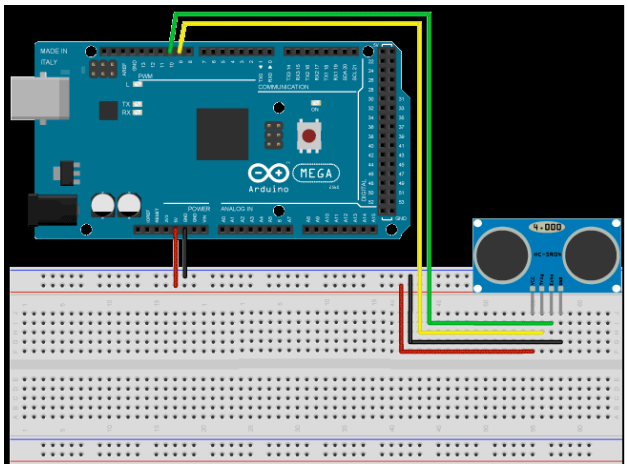

Figure 1.8: Pin Connection between ultrasonic sensor and Arduino

To generate the ultrasound, the Trig pin is set on a High State for 10 µs. That will send out an 8 cycles sonic burst which will travel at the speed sound and it will be received in the Echo pin.

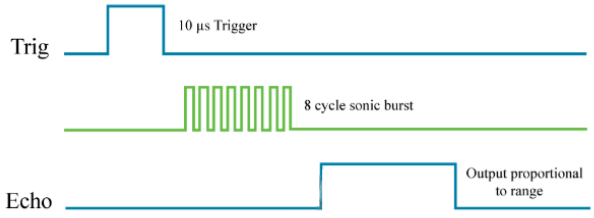

Figure 1.9: Operation of sonar sensor (pulse diagram)

If an example is considered, when an object is 10 cm away from the sensor, and the speed of the sound is 340 m/s or 0.034 cm/µs the sound wave will need to travel about 294 u seconds. However, the Echo pin will be double that number because the sound wave needs to travel forward and bounce backward. Therefore, in order to get the distance in cm, the received travel time value from the echo pin is multiplied by 0.034 and divide it by 2. The speed is found by using the equation Velocity=Distance/Tine and taking the difference of distance taken at different times and dividing it with the time between the taking of the differences.

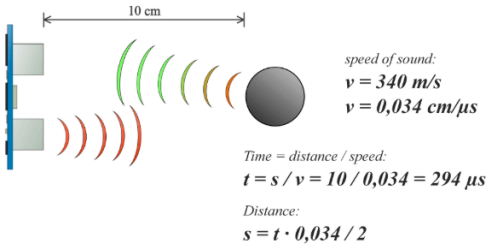

Figure 1.10: An example of sonar sensor.

A sample code for the sonar operation is given below:

```
// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;

// defines variables
long duration;
int distance;

void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}

void loop() {
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);

// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

// Reads the echoPin, returns the sound wave travel time in microsec
duration = pulseIn(echoPin, HIGH);

// Calculating the distance
distance= duration*0.034/2;

// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
}
```

Figure 1.11: Sample code for sonar sensor operation and control

Cooling Fan:

In this project, A 9cm 4-pin computer fan is used and controlled using the pulse width modulation.
The pin number and the pin functionality are shown in figures 4.1:

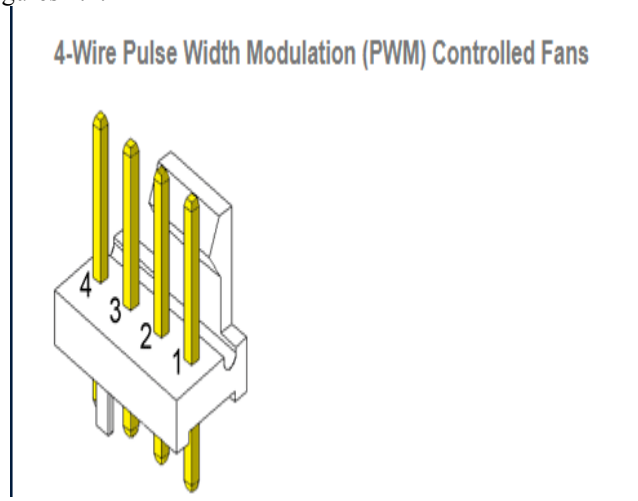

Figure 1.12 pin numbering

Pin	Name	Color
1	GND	black
2	+12VDC	red
3	Sense	yellow
4	Control	blue

Table 1.2: 4-pin fan pin functions

For the fan, we were originally planning to use a three-pin fan, which has pins for positive voltage, negative voltage and one to find the speed of the fan. We were going to lower and raise the voltage to control the speed of the fan using DC voltage control.
We found a better way to go about it using pulse width modulation with four pin fan. This fan in addition to the previous three pins has an additional pin that controls the fan using pulse width modulation. Pulse width modulation is a type of digital signal that acts like a switch that turns on and off, regulating whatever it is controlling. Depending on the duty cycle, which is the percentage of the period that is the "switch" is on, these impulses of power get faster or slower. As the impulses get faster and the duty cycle is higher the fan spins faster as it is "on" more often. When the impulses get slow and the duty cycle is low the fan spins slower as it is "off" more often. If there is no signal the fan would instead work on full power since the voltage powering the fan is separate from the controlling signal. This separate power source allows us to control the fan using the Arduinos 5 volts.

Pulse width modulation is perfect in controlling our fan because it allows us to have a steadier change in speed. The fan can be seamlessly run in lower speeds without stalling or having a sudden sharp drop in speed. This was a problem that we saw in voltage control when on the lower voltages the fan would completely stop.
Thus, the output of our fuzzy logic is the duty cycle of the pulse width modulator. Although by using a pulse width modulator for a computer fan the frequency of the square wave must be sufficiently high enough to control the fan so the frequency was changed to 25khz. During the beginning of the program we have a delay before the pulse width modulation signal so that a big burst of air will start up the process. After which using the inputs given by the sensor they would go through the fuzzy logic and output the duty cycle to the pulse width modulator. As the ball changes position the inputs would also change and the duty cycle would change in accordance, changing the speed of the fan. This would allow the fan to move the ball into the designated position.

```
word VentPin = 11;
int dutycycle=0;

void setup() {
  pinMode(VentPin, OUTPUT);
  pwm25kHzBegin();
}
void loop() {
  fuzzy()
  dutycycle=fuzzyout;
digitalWrite(VentPin, HIGH);
  delayMicroseconds(dutycycle); // Approximately 10% duty cycle @
  digitalWrite(VentPin, LOW);
  delayMicroseconds(79 - dutycycle);
}
void pwm25kHzBegin() {
  TCCR2A = 0;                                 // TC2 Control Registe
  TCCR2B = 0;                                 // TC2 Control Registe
  TIMSK2 = 0;                                 // TC2 Interrupt Mask
  TIFR2 = 0;                                  // TC2 Interrupt Flag
  TCCR2A |= (1 << COM2B1) | (1 << WGM21) | (1 << WGM20);
  // OC2B cleared/set on match when up/down counting, fast PWM
  TCCR2B |= (1 << WGM22) | (1 << CS21);      // prescaler 8
  OCR2A = 79;                                 // TOP overflow value
  OCR2B = 0;
}
}
```

Figure 1.13: Sample code for fan control

In the above code, an initial fan duty cycle is set to 0. Note that, the digitalwrite() function of the Arduino generates the PWM signals using duty cycle.
We put the parts together and connected it to a 120cm tall clear plastic tube, and our goal is to keep the ball flying in constant height in the middle of the tube.

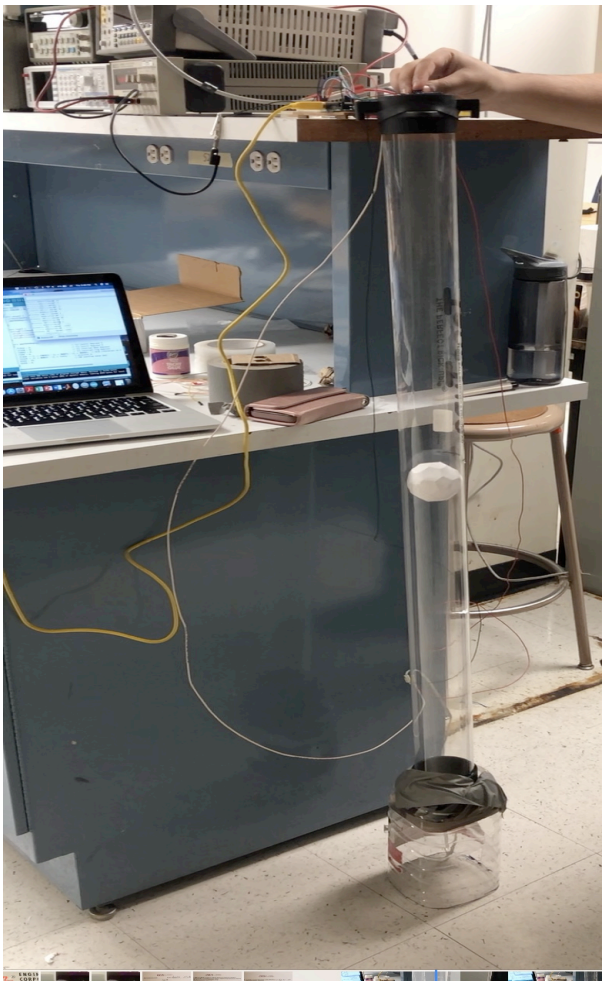

Figure 1.14 Project equipment.

C. Coding for different parts:

This project reqiures a lot of coding in order to define the readings that come from the ultrasound sensor. These readings are used to find the distance from the echo of the sonic signal and the speed of the ball. Tthese variables are transferred to the fuzzy logic from the MATLAB fuzzy toolbox, and based on the rules the PWM is the output of the system that controls the fan that adjusts the ball height.

The Appendix in this report shows the distance sensor code, the fuzzy rules code, and the PWM code.

D. Cost:

Element	Price	Store	Type
Tube	18$	Lowe's home improvement	120cm Tube
CPU Cooling Fan	11.99$	Fry's Electronics	
Arduino	45$	Fry's Electronics	UNO
Ultrasonic Sensor	14.95	Fry's	MCSR04
Total Cost	90$		

Table 1.3: Cost of the project.

IV. Testing

We divided the testing into three stages as shown in table 1.4.

Stage No.	**Goal**	**Results**
1	Test each part seperately	Sensor works, Fan works.
2	Test each part with our project code	The sensor code works, the fan responds to a PWM.
3	Apply all the parts with the fuzzy logic code.	In Progress, the fuzzy logic works but the PWM code generation does not.

Table 1.4: Testing Stages so far.

For the first testing stage, we made sure that the Ultrasonic sensor and the fans could work for the project by seeing if the sensor could sense the bottom of the tube and if the fans were powerful enough to blow the ball in the tube. We went through 2 fans before deciding on the fans that we currently use.

For the second testing stage, we created a program that creates a pulse in order to test the pulse width modulation of the fan, in addition to finding parameters to use for the fuzzy logic. By using the code, we were able to find the duty cycle that kept the ball floating in the middle of the tube. We also tested the code that would be able to find the correct distance and speed to use in the inputs, and changed the values to a reasonable cm/s. We performed an initial testing for the floating ball and the results are shown in figure 1.15.

```
distance from center:17  speed:50        dutycycle: 45
distance from center:13  speed:40        dutycycle: 45
distance from center:-4  speed:170       dutycycle: 45
distance from center:-27         speed:230       dutycycle: 65
distance from center:-52         speed:250       dutycycle: 65
distance from center:-51         speed:10        dutycycle: 65
distance from center:-51         speed:0         dutycycle: 65
distance from center:-51         speed:0         dutycycle: 65
distance from center:-51         speed:0         dutycycle: 65
```

Figure1.15: testing results for the floating ball

From figure 5.1, the distance from center first was 17cm, which means the ball is close from the desired distance that is 0 cm. Then, the ball falled below 0 cm to -51 cm which required bigger duty cycle and it was 65, this is shown in figure 1.16.

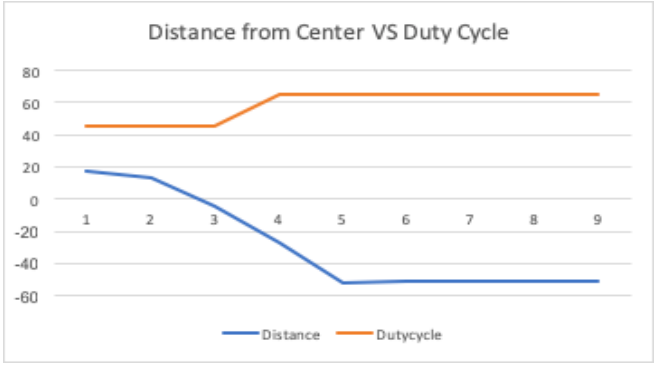

Figure 1.16: Distance of ball VS duty cycle

For the third testing stage we connected the sensor, fan and Arduino together and put the inputs from the sensor into the fuzzy logic to get a duty cycle for the fan. Although the fuzzy code worked and the duty cycle was gained from the inputs, the pulse width code does not work.

We are now planning three steps to finish this project. First, we have print the 3d model for the tube base which will help the air flow into the fan, as well as provide a steadier base. Second, we will connect the fuzzy logic output to a PWM generator. Third, we will try different fuzzy rules and membership functions in order to maximize speed, accuracy and precision of the ball.

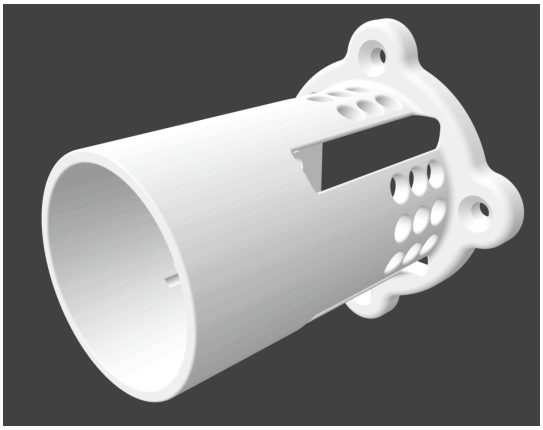

Figure 1.17: 3d printed model for the tube base

CONCLUSION:

This project is a great way to show how fuzzy logic is useful for different applications. By using a cheap Sensor and fan we can control a ball in a seamless way. This is opposed by using digital logic and calculating every possible result from every possible input to achieve a similar smoothness.

By using fuzzy logic we can program the fan to blow depending on the inputs, and create more complex reactions to the inputs then if we used digital logic. The fuzzy logics ability to have a more "grey" logic results in a smoother and more accurate result the digital logics more "black and white" logic.

We are still doing testing on the coding part because it is the hardest and most challenging part. The hardware part is going to be done as long as we get the 3d printed parts done in one week. In the future, we want to try to use an accelerometer to find the speed of the ball as this would get a more accurate result then the ultrasonic sensor. The current method of finding the speed has the problem of multiplying the error if the ultrasonic sensor gets the wrong value. This messes up the result. If we were to use an accelerometer this error would not propagate and the result would be more accurate.

REFERENCES

[1]http://www.intechopen.com/source/html/39428/media/image1.jpeg.
[2] http://www.ti.com/lit/ds/symlink/lm35.pdf

Appendix:

```
#include "fis_header.h"

// Number of inputs to the fuzzy inference system
const int fis_gcI = 2;
// Number of outputs to the fuzzy inference system
const int fis_gcO = 1;
// Number of rules to the fuzzy inference system
const int fis_gcR = 0;

FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];

// Setup routine runs once when you press reset:
void setup()
{
   // initialize the Analog pins for input.
   // Pin mode for Input: Position
   pinMode(0 , INPUT);
   // Pin mode for Input: Speed(ball)
   pinMode(1 , INPUT);


   // initialize the Analog pins for output.
   // Pin mode for Output: output1
   pinMode(2 , OUTPUT);

}

// Loop routine runs over and over again forever:
void loop()
{
   // Read Input: Position
   g_fisInput[0] = analogRead(0);
   // Read Input: Speed(ball)
   g_fisInput[1] = analogRead(1);

   g_fisOutput[0] = 0;

   fis_evaluate();

   // Set output vlaue: output1
   analogWrite(2 , g_fisOutput[0]);

}

//***********************************************************************
// Support functions for Fuzzy Inference System
//***********************************************************************
// Trapezoidal Member Function
FIS_TYPE fis_trapmf(FIS_TYPE x, FIS_TYPE* p)
{
   FIS_TYPE a = p[0], b = p[1], c = p[2], d = p[3];
   FIS_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d - x) / (d - c)) : 0)));
   FIS_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x - a) / (b - a)) : 0)));
   return (FIS_TYPE) min(t1, t2);
}

// Gaussian Member Function
FIS_TYPE fis_gauss2mf(FIS_TYPE x, FIS_TYPE* p)
{
```

```
    FIS_TYPE c1 = p[1], c2 = p[3];
    FIS_TYPE t1 = ((x >= c1) ? 1.0 : fis_gaussmf(x, p));
    FIS_TYPE t2 = ((x <= c2) ? 1.0 : fis_gaussmf(x, p + 2));
    return (t1*t2);
}

// Gaussian Member Function
FIS_TYPE fis_gaussmf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE s = p[0], c = p[1];
    FIS_TYPE t = (x - c) / s;
    return exp(-(t * t) / 2);
}

FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}

FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}

FIS_TYPE fis_array_operation(FIS_TYPE *array, int size, _FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }

    return ret;
}


//***************************************************************************
// Data for Fuzzy Inference System
//***************************************************************************
// Pointers to the implementations of member functions
_FIS_MF fis_gMF[] =
{
    fis_trapmf, fis_gauss2mf, fis_gaussmf
};

// Count of member function for each Input
int fis_gIMFCount[] = { 5, 5 };

// Count of member function for each Output
int fis_gOMFCount[] = { 5 };

// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFI0Coeff1[] = { 45, 65, 80, 90 };
FIS_TYPE fis_gMFI0Coeff2[] = { 80, 95, 105, 120 };
FIS_TYPE fis_gMFI0Coeff3[] = { 0, 0, 50, 65 };
FIS_TYPE fis_gMFI0Coeff4[] = { 111, 120, 135, 155 };
```

```
FIS_TYPE fis_gMFI0Coeff5[] = { 135, 150, 200, 200 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2, fis_gMFI0Coeff3, fis_gMFI0Coeff4,
fis_gMFI0Coeff5 };
FIS_TYPE fis_gMFI1Coeff1[] = { 40, 65, 80, 90 };
FIS_TYPE fis_gMFI1Coeff2[] = { 0, 0, 35, 65 };
FIS_TYPE fis_gMFI1Coeff3[] = { 80, 95, 105, 120 };
FIS_TYPE fis_gMFI1Coeff4[] = { 110, 120, 135, 162.698412698413 };
FIS_TYPE fis_gMFI1Coeff5[] = { 135, 165, 200, 200 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2, fis_gMFI1Coeff3, fis_gMFI1Coeff4,
fis_gMFI1Coeff5 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff };

// Coefficients for the Output Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { 2, 33, 2, 35 };
FIS_TYPE fis_gMFO0Coeff2[] = { 2, 83, 2, 85 };
FIS_TYPE fis_gMFO0Coeff3[] = { 2, 99, 2, 101 };
FIS_TYPE fis_gMFO0Coeff4[] = { 2, 112, 2, 114 };
FIS_TYPE fis_gMFO0Coeff5[] = { 2, 163, 2, 165 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2, fis_gMFO0Coeff3, fis_gMFO0Coeff4,
fis_gMFO0Coeff5 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff };

// Input membership function set
int fis_gMFI0[] = { 0, 0, 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1};

// Output membership function set
int fis_gMFO0[] = { 1, 1, 1, 1, 1 };
int* fis_gMFO[] = { fis_gMFO0};

// Rule Weights
FIS_TYPE fis_gRWeight[] = { };

// Rule Type
int fis_gRType[] = { };

// Rule Inputs

int* fis_gRI[] = { };

// Rule Outputs

int* fis_gRO[] = { };

// Input range Min
FIS_TYPE fis_gIMin[] = { 0, 0 };

// Input range Max
FIS_TYPE fis_gIMax[] = { 200, 200 };

// Output range Min
FIS_TYPE fis_gOMin[] = { 0 };

// Output range Max
FIS_TYPE fis_gOMax[] = { 200 };

//*********************************************************************
// Data dependent support functions for Fuzzy Inference System
//*********************************************************************
FIS_TYPE fis_MF_out(FIS_TYPE** fuzzyRuleSet, FIS_TYPE x, int o)
{
```

```
    FIS_TYPE mfOut;
    int r;

    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
            mfOut = (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut = 1 - (fis_gMF[fis_gMFO[o][index]])(x, fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }

        fuzzyRuleSet[0][r] = fis_min(mfOut, fuzzyRuleSet[1][r]);
    }
    return fis_array_operation(fuzzyRuleSet[0], fis_gcR, fis_max);
}

FIS_TYPE fis_defuzz_centroid(FIS_TYPE** fuzzyRuleSet, int o)
{
    FIS_TYPE step = (fis_gOMax[o] - fis_gOMin[o]) / (FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;

    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(fuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }

    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) : (momentum / area));
}

//***************************************************************************
// Fuzzy Inference System
//***************************************************************************
void fis_evaluate()
{
    FIS_TYPE fuzzyInput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE fuzzyInput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyInput[fis_gcI] = { fuzzyInput0, fuzzyInput1, };
    FIS_TYPE fuzzyOutput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* fuzzyOutput[fis_gcO] = { fuzzyOutput0, };
    FIS_TYPE fuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE fuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* fuzzyRuleSet[] = { fuzzyRules, fuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to fuzzy Input
    int i, j, r, o;
```

```
    for (i = 0; i < fis_gcI; ++i)
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            fuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i], fis_gMFICoeff[i][j]);
        }
    }

    int index = 0;
    for (r = 0; r < fis_gcR; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            fuzzyFires[r] = FIS_MAX;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_min(fuzzyFires[r], 1);
            }
        }
        else
        {
            fuzzyFires[r] = FIS_MIN;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], fuzzyInput[i][index - 1]);
                else if (index < 0)
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 1 - fuzzyInput[i][-index - 1]);
                else
                    fuzzyFires[r] = fis_max(fuzzyFires[r], 0);
            }
        }

        fuzzyFires[r] = fis_gRWeight[r] * fuzzyFires[r];
        sW += fuzzyFires[r];
    }

    if (sW == 0)
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = fis_defuzz_centroid(fuzzyRuleSet, o);
        }
    }
}
```