CENG386 FUZZY LOGIC DC MOTOR RPM CONTROL

YAHYA EKIN 260206054

PROGRESS REPORT

Index:

1) Purpose:

The RPM control of DC motor is fundamental concept used in the field of where electric motors are used, as an obvious example, electric cars. But how we can sure about that when we press the accelerator pedal in our car, the rpm of the motor is approaching our desired RPM value. To be sure about that, Mamdani type-1 fuzzy logic control system can be used. To control the motor, we need two linguistic variables as RPM Error and Change of RPM Error. The definitions of both are:

RPM Error = Desired RPM – Measured Motor RPM.

Change of RPM Error = RPM Error – Previous RPM Error.

The output linguistic variable is Fuzzy Output PWM, which is defined as:

Fuzzy Output PWM = Output PWM – Previous Output PWM.

To perform the motor control simple Arduino setup is designed. This setup consists of IR Receiver, which is used to measure the motor RPM, 12V DC motor, Potentiometer to define desired RPM value, DC motor driver and Arduino Nano Board.

2) Fuzzy Logic Toolbox

a. Membership Functions

The membership functions are influenced from (Freitas, Rameli, & EAK, 2017), but the range of the functions are adapted to our DC motor. The DC motor that used can deliver 1000 RPM at 12V according to its datasheet. Since we are using L298N Motor Driver, which has 2V voltage drop, our maximum output voltage is limited to 10V. The expected motor RPM at 10V is near 800 RPM. To verify that, motor RPM is measured against the rising PWM values which is used to drive the motor. The graph of motor curve is represented below.

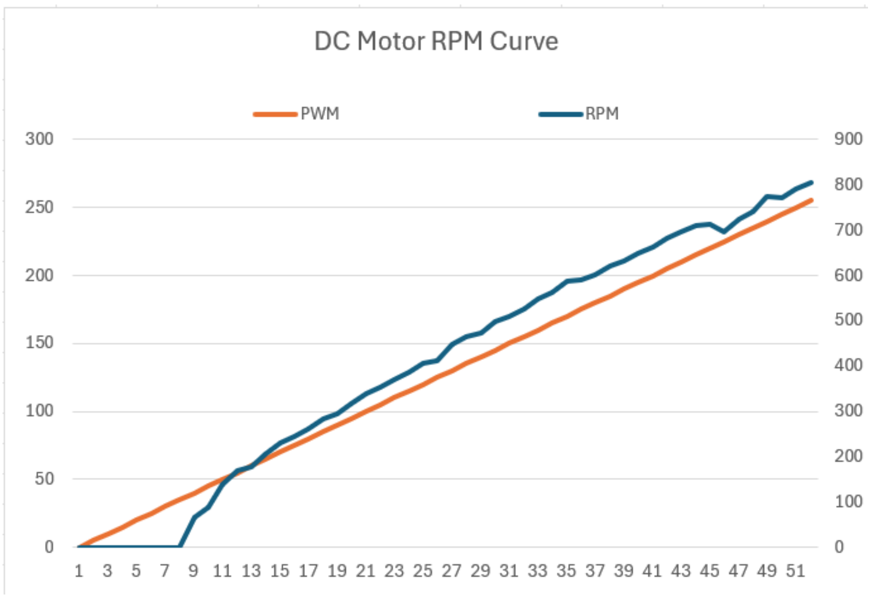
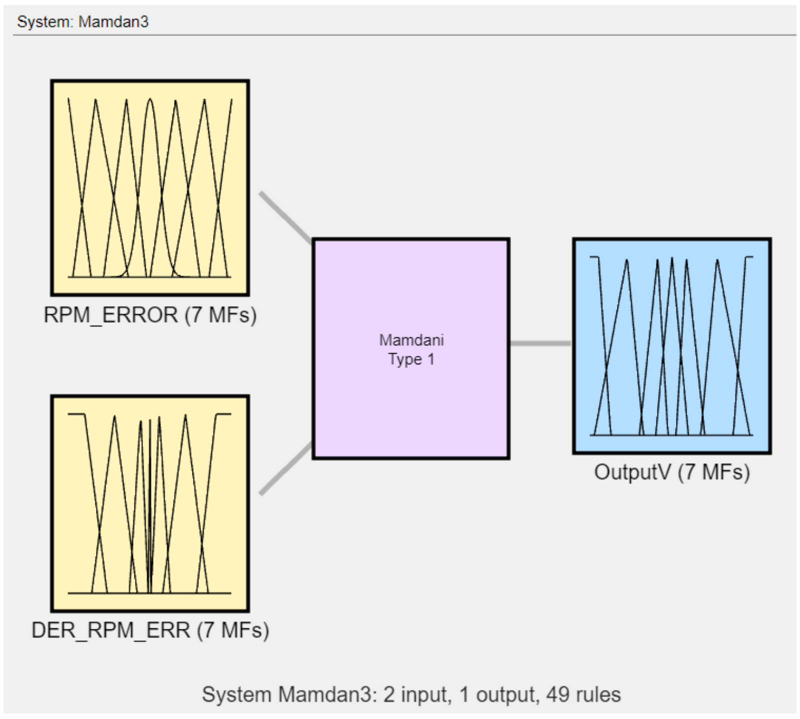


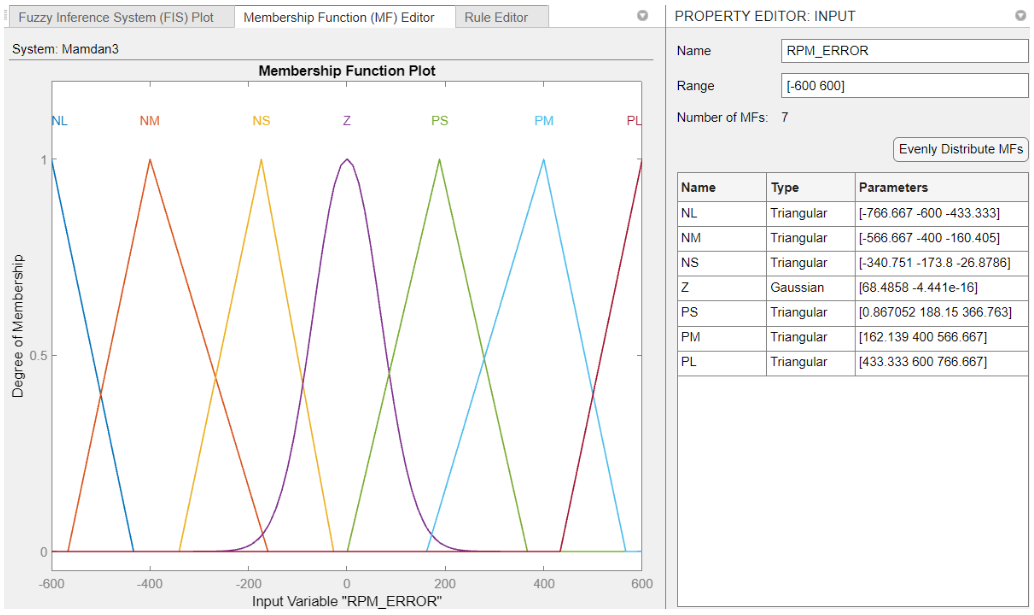
Figure 1. DC Motor RPM Curve.

To stay in safer limits, the motor RPM is limited to 600 RPM in membership functions. The input membership functions are represented below.



NL (Negative Low), NS (Negative Small), Z (Zero), PS (Positive Small), PL (Positive Large)

ANL (Add Negative Large number), APS (Add Positive Small number), ..etc.

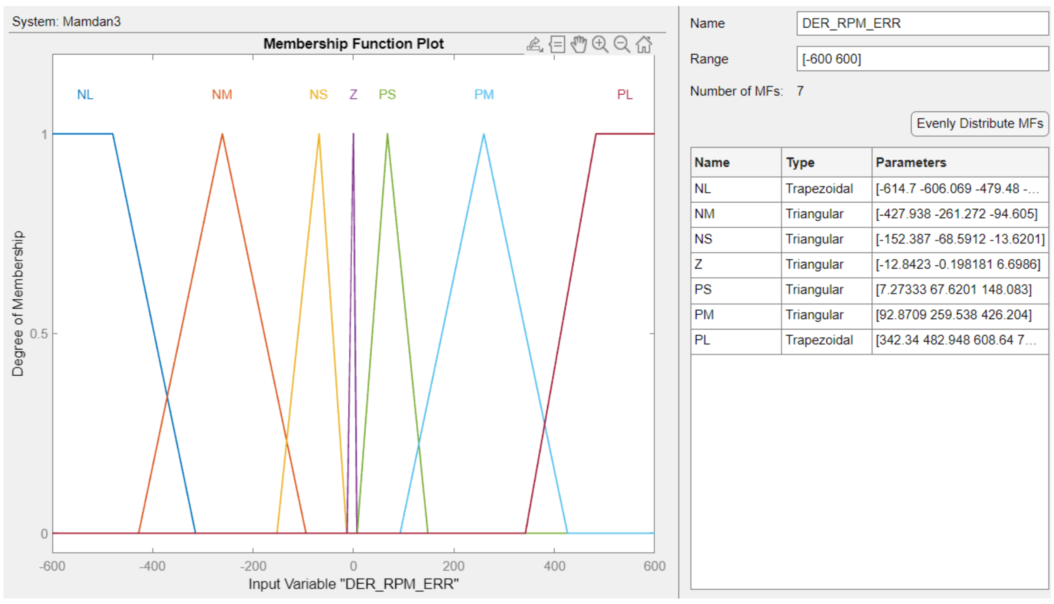


System: Mamdan3

PROPERTY EDITOR: INPUT

Name	RPM_ERROR
Range	[-600 600]
Number of MFs:	7

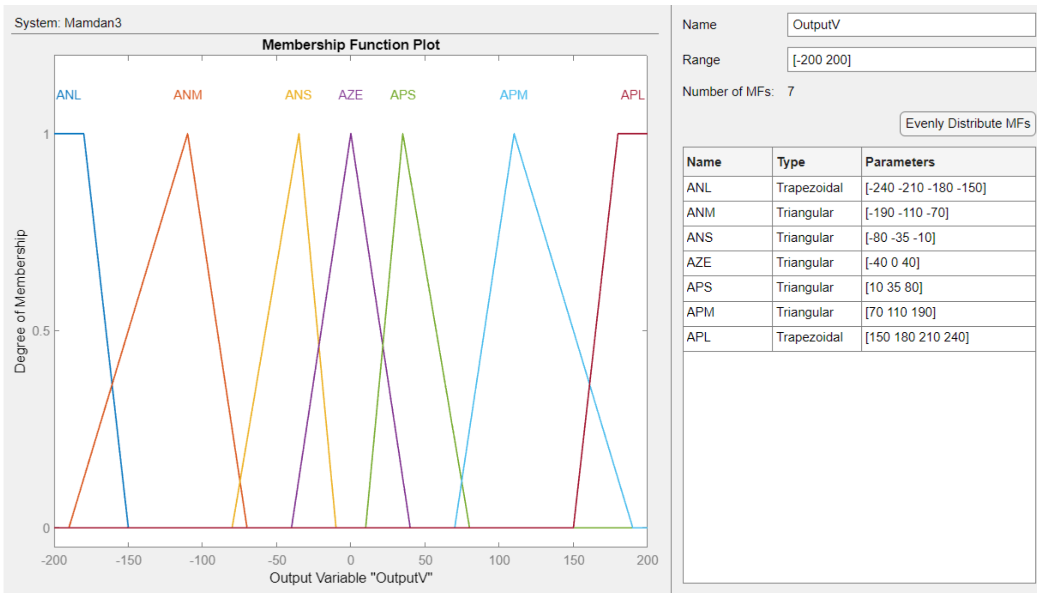
Name	Type	Parameters
NL	Triangular	[-766.667 -600 -433.333]
NM	Triangular	[-566.667 -400 -160.405]
NS	Triangular	[-340.751 -173.8 -26.8786]
Z	Gaussian	[68.4858 -4.441e-16]
PS	Triangular	[0.867052 188.15 366.763]
PM	Triangular	[162.139 400 566.667]
PL	Triangular	[433.333 600 766.667]



System: Mamdan3

Name	DER_RPM_ERR
Range	[-600 600]
Number of MFs:	7

Name	Type	Parameters
NL	Trapezoidal	[-614.7 -606.069 -479.48 -…
NM	Triangular	[-427.938 -261.272 -94.605]
NS	Triangular	[-152.387 -68.5912 -13.6201]
Z	Triangular	[-12.8423 -0.198181 6.6986]
PS	Triangular	[7.27333 67.6201 148.083]
PM	Triangular	[92.8709 259.538 426.204]
PL	Trapezoidal	[342.34 482.948 608.64 7…

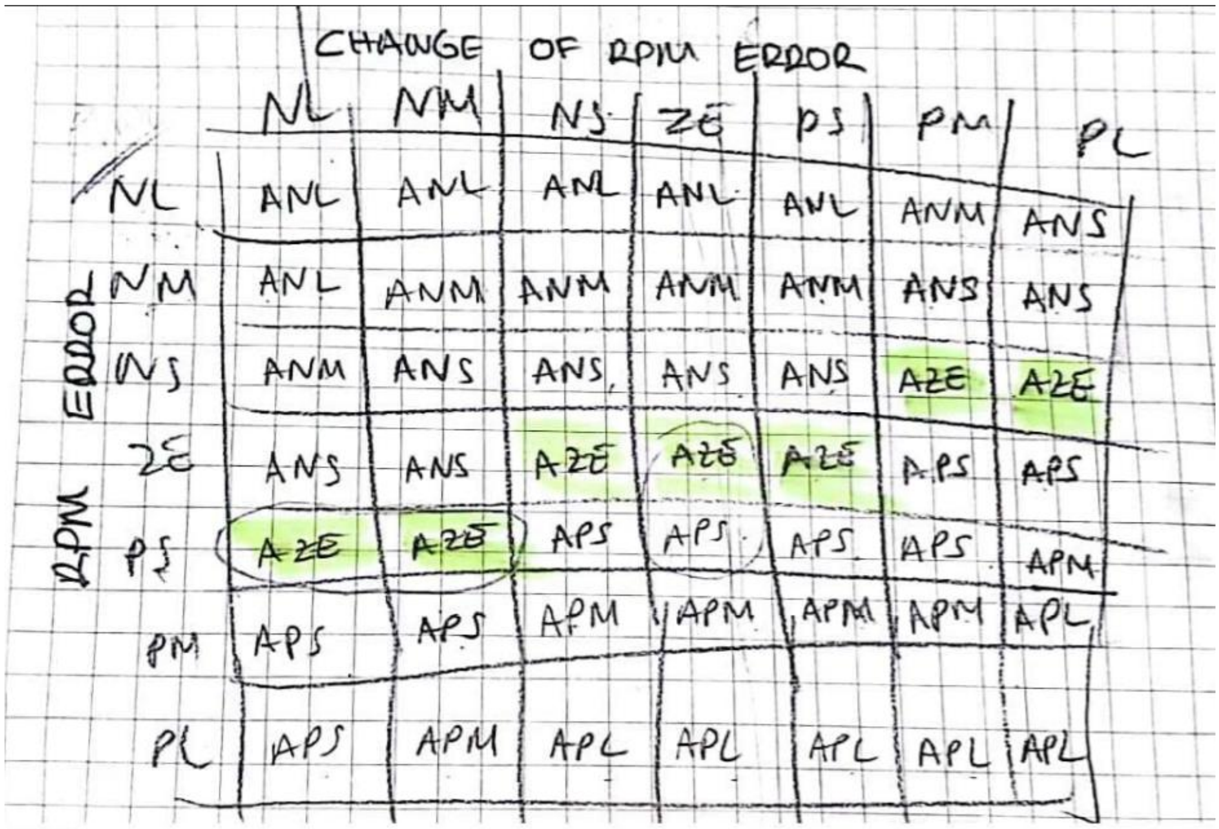


System: Mamdan3

Name	OutputV
Range	[-200 200]
Number of MFs:	7

Name	Type	Parameters
ANL	Trapezoidal	[-240 -210 -180 -150]
ANM	Triangular	[-190 -110 -70]
ANS	Triangular	[-80 -35 -10]
AZE	Triangular	[-40 0 40]
APS	Triangular	[10 35 80]
APM	Triangular	[70 110 190]
APL	Trapezoidal	[150 180 210 240]

b. Rule Base



RPM ERROR \ CHANGE OF RPM ERROR	NL	NM	NS	ZE	PS	PM	PL
NL	ANL	ANL	ANL	ANL	ANL	ANM	ANS
NM	ANL	ANM	ANM	ANM	ANM	ANS	ANS
NS	ANM	ANS	ANS	ANS	ANS	AZE	AZE
ZE	ANS	ANS	AZE	AZE	AZE	APS	APS
PS	AZE	AZE	APS	APS	APS	APS	APM
PM	APS	APS	APM	APM	APM	APM	APL
PL	APS	APM	APL	APL	APL	APL	APL

The rule base table will be replaced by excel version immediately. This rule base is als influenced from (Freitas, Rameli, & EAK, 2017), but the output rules are replaced with the inverse of the rules in order to fit our system.

3) Generating Lookup Table

To implement fuzzy logic control to Arduino platform, various libraries might be used to generate and evaluate fuzzy logic in the Arduino. However, in my approach I chose to use MATLAB's Fuzzy Logic Toolbox as a fuzzy designer and use output of the fuzzy system as a lookup table in the Arduino. Within these 2 steps, anyone can export the output of any fuzzy system as a lookup table.

1- Generating Lookup Table in MATLAB

To generate the lookup table, the evalfis (fis, input1, input2) function in MATLAB can be used to evaluate fuzzy inference system with given inputs as below. Anyone might wonder why we define arrays CE and E in a way that increments 20 value per step. The reason is the limited storage size of Arduino Nano. If we decide to use whole table which consists of 1200x1200 double array, the size will not fit the Arduino's available space.

```matlab
clear all;
fis = readfis('Mamdan3');

% Define input variables
E = -600:20:600; %RPM ERROR
CE = -600:20:600; % CHANGE OF RPM ERROR
N = length(E);
LookUpTableData = zeros(N);

% Compute output values for each combination of input samples
for i = 1:N
    for j = 1:N
        LookUpTableData(i,j) = evalfis(fis,[E(i) CE(j)]);
    end
end

%LookUpTableData = LookUpTableData;
LookUpTableData = round(LookUpTableData);
```

Another important note is that the resulting array should be stored in Arduino’s program memory, not in the SRAM as a global variable. The global variable space in Arduino is 2kB, but the program space is 30kB. The size of the lookup table should be arranged by considering remaining program space in the memory.

2- Convering MATLAB array to Arduino array.

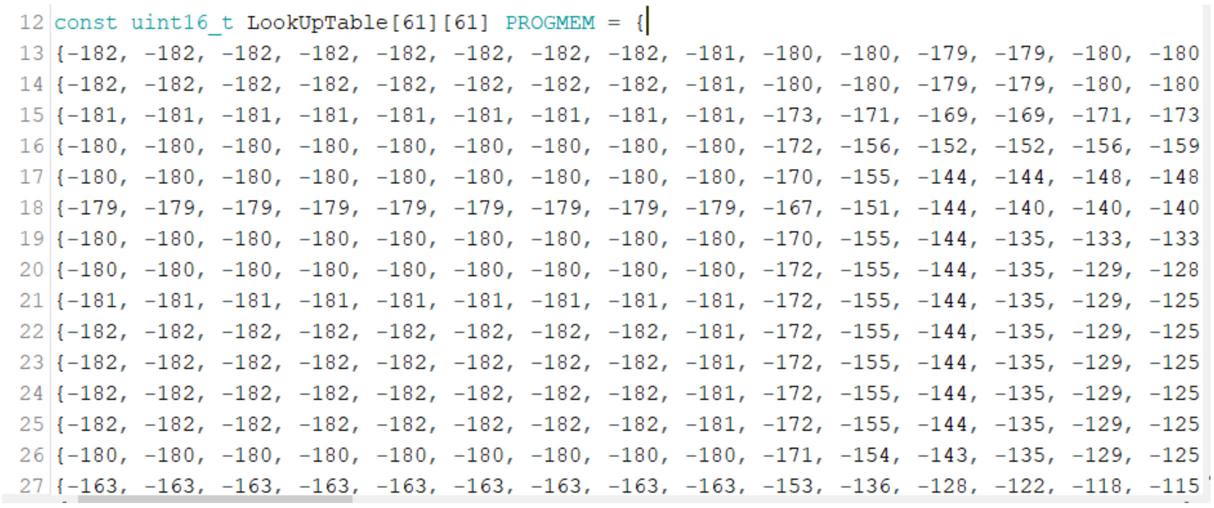
```matlab
% Matlab'da örnek bir 61x61'lik array oluşturma
matlabArray = randi([0, 255], 61, 61); % 0 ile 255 arasında rastgele tam sayılar

% Matlab'daki array'i Arduino'da kullanılacak formata dönüştürme
% 2 boyutlu array'i tek boyuta dönüştürme
arduinoArray = reshape(LookUpTableData.', 1, []);

% Dönüştürülmüş array'i Arduino'ya aktarmak için uygun formatta yazdırma
disp('const uint16_t LookUpTable[61][61] PROGMEM = {');
for i = 1:size(matlabArray, 1)
    fprintf('{');
    for j = 1:size(matlabArray, 2)
        fprintf('%d', arduinoArray((i-1)*size(matlabArray, 2) + j));
        if j < size(matlabArray, 2)
            fprintf(', ');
        end
    end
    fprintf('}');
    if i < size(matlabArray, 1)
        fprintf(',\n');
    else
        fprintf('\n};');
    end
end
```

In this way, we generate Arduino compatible lookup table within minutes.

4) Implementing Lookup Table in Arduino



```
const uint16_t LookUpTable[61][61] PROGMEM = {
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -180, -180, -179, -179, -180, -180
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -180, -180, -179, -179, -180, -180
{-181, -181, -181, -181, -181, -181, -181, -181, -181, -173, -171, -169, -169, -171, -173
{-180, -180, -180, -180, -180, -180, -180, -180, -180, -172, -156, -152, -152, -156, -159
{-180, -180, -180, -180, -180, -180, -180, -180, -180, -170, -155, -144, -144, -148, -148
{-179, -179, -179, -179, -179, -179, -179, -179, -179, -167, -151, -144, -140, -140, -140
{-180, -180, -180, -180, -180, -180, -180, -180, -180, -170, -155, -144, -135, -133, -133
{-180, -180, -180, -180, -180, -180, -180, -180, -180, -172, -155, -144, -135, -129, -128
{-181, -181, -181, -181, -181, -181, -181, -181, -181, -172, -155, -144, -135, -129, -125
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -172, -155, -144, -135, -129, -125
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -172, -155, -144, -135, -129, -125
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -172, -155, -144, -135, -129, -125
{-182, -182, -182, -182, -182, -182, -182, -182, -181, -172, -155, -144, -135, -129, -125
{-180, -180, -180, -180, -180, -180, -180, -180, -180, -171, -154, -143, -135, -129, -125
{-163, -163, -163, -163, -163, -163, -163, -163, -163, -153, -136, -128, -122, -118, -115
```

Here's the portion of the lookup table in the Arduino. The important point is to add "PROGMEM" word after the array definition which tells the compiler to save this array into program memory. However, the program memory is only readable. To read a value from our stored array, this method can be used:

Out_PWM = pgm_read_word(&LookUpTable[selected_y][selected_x]);

The variables selected_y and selected_x defines the index values to reach the data in lookup table. Here's how we define them:

```
rounded_x_input  = round(rpm_error / 20.0) * 20;
rounded_y_input  = round(change_of_rpm_error / 20.0) * 20;
int selected_x = map(rounded_x_input, -600, 600, 0, 61);
int selected_y = map(rounded_y_input, -600, 600, 0, 61);

Out_PWM = pgm_read_word(&LookUpTable[selected_y][selected_x]);
```

Why we use rounded inputs is that: Since we resized our lookup table array, shortened it, we can not directly access to it using ordinary values. Because our inputs, rpm_error and change_of_rpm_error is changing continuously, not by incremented by 20. So, we need to use interpolation to reach our closest desired value in the lookup table. For instance, if our rpm_error is 585,

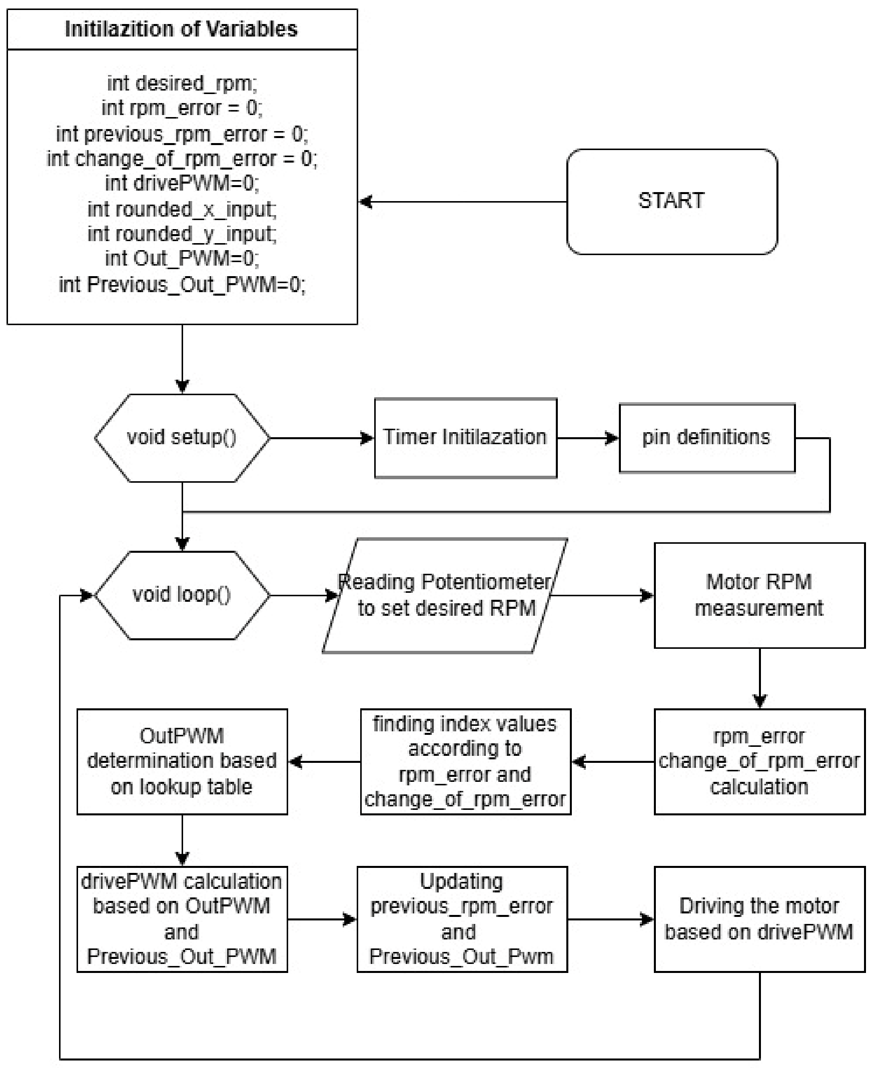
$$585/20 = 29.25$$
$$\text{Round}(29.25) = 29$$
$$29*20 = 580$$

To find which index value is corresponds to 580, the map() function of the Arduino is used. Let's say 580 is the x'th input in our shortened lookup table, so in order to access the output data according to 580, we use the index value x.

5) Code Explanation

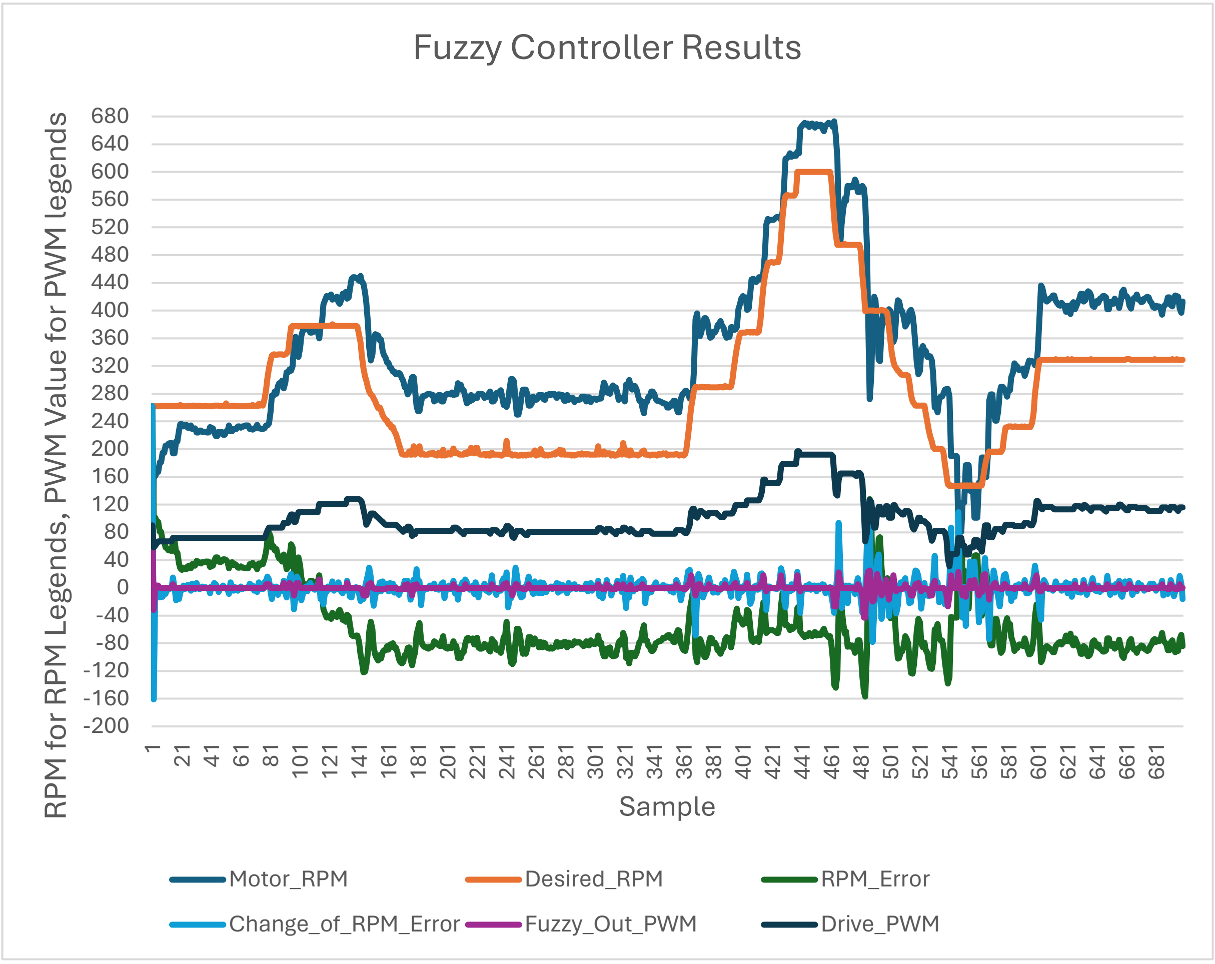
Here's the flowchart of the program.The program gets user input as desired RPM and fuzzy controller is used to reach the desired RPM and keep the motor on desired RPM. Full code is available on ino file. The main logic is taking the rpm data from the motor, calculate rpm error and rpm error change to use input for fuzzy inference system, obtain the response of the fuzzy system via lookup table, use the output control pwm to drive the motor and so on. There is a point shoul I note that the output PWM is not the output cames from the lookup table. Previously, I used that to drive the motor but motor stalls since most of the time the adjustments made from the fuzzy system is relatively small to drive the motor. After couple of hours of error checking and researching, I noticed that the output PWM value from the fuzzy inference represents the change in the output, not the output to drive the motor. So, I adjusted the code in that way, this is why we have separate variable to drive the motor as drivePWM and Out_PWM. Out_PWM is the PWM value comes from fuzzy inference. DrivePWM is the PWM value to drive motor. DrivePWM can be calculated as:

```
drivePWM = Previous_Out_PWM + (Out_PWM);
Previous_Out_PWM = drivePWM;
```


Initilazition of Variables
int desired_rpm;
int rpm_error = 0;
int previous_rpm_error = 0;
int change_of_rpm_error = 0;
int drivePWM=0;
int rounded_x_input;
int rounded_y_input;
int Out_PWM=0;
int Previous_Out_PWM=0;
START
void setup()
Timer Initilazation
pin definitions
void loop()
Reading Potentiometer to set desired RPM
Motor RPM measurement
rpm_error change_of_rpm_error calculation
finding index values according to rpm_error and change_of_rpm_error
OutPWM determination based on lookup table
drivePWM calculation based on OutPWM and Previous_Out_PWM
Updating previous_rpm_error and Previous_Out_Pwm
Driving the motor based on drivePWM

6) Results

The results are taken from the Arduino serial monitor and transferred to excel to process the data. The RPM and PWM graph of fuzzy controller is attached below.



From the graph above, the motor RPM and desired RPM can be tracked, also the error variables of our fuzzy system RPM error and change of RPM error can be observed. From this output, one can say that the fuzzy inference system's fuzzy output PWM matches with change of rpm error. Also, the motor tracks the desired RPM with significant error margin. Although desired RPM and motor RPM aligns, there is an error gap occurred. The error is around -80 RPM. This error might be lowered by adjusting the rule base or input membership functions, or adding some rescaling operations to evaluate drive PWM, which is used to drive the motor.