T.C.

IZMIR KATIP CELEBI UNIVERSITY FACULTY OF ENGINEERING AND ARCHITECTURE DEPARTMENT OF MECHATRONICS ENGINEERING

MEE427 PID Control Project Report

200412050 – Murat Han UYSAL

190412029 – Aytekin DÖNMEZ

Instructor: Levent Çetin

Izmir – 11 December 2024

Contents

Introduction

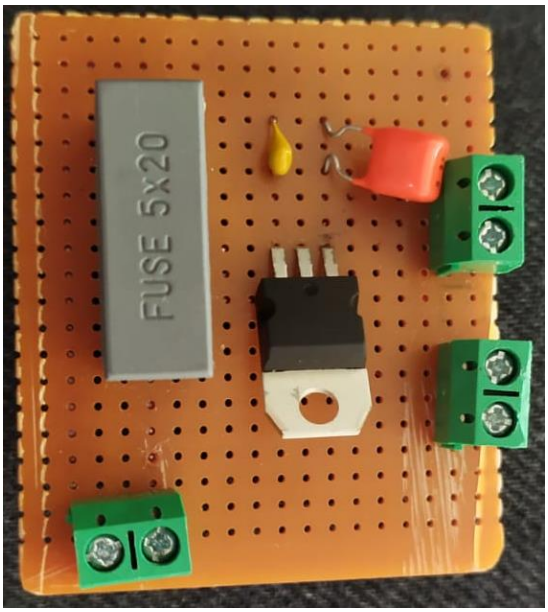
This project combines mechatronics and control systems to achieve precise position control of a DC motor with Proportional-Integral-Derivative (PID) control strategy implemented on a Microcontroller (PIC16F877A). During the experimental phase, different duty cycles are used to modify the motor velocities, and data from an incremental encoder is acquired using a LabVIEW software. The obtained data is then processed and analyzed using MATLAB, which produces essential velocity values for the encoder under various duty cycles.

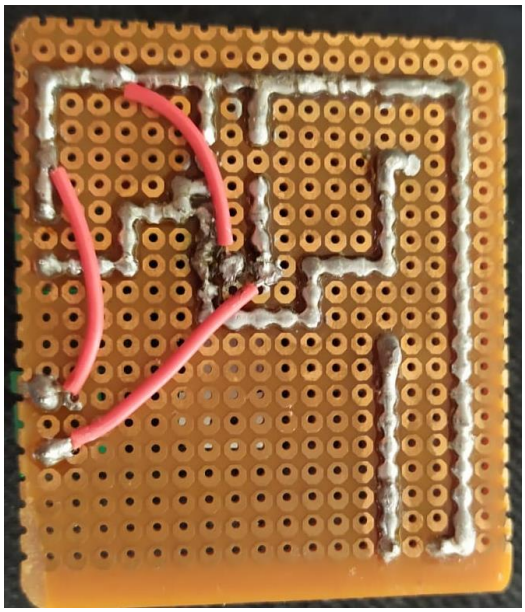
In order to achieve exact position control through potentiometer calibration and to understand motor extraction speed, a second project that focuses on the dynamic analysis and control of motor behavior is being undertaken. The study examines data from an Excel file using a PID control method and thorough data analysis using MATLAB, offering an in-depth understanding of motor dynamics.

The emphasis on PID control for accurate position regulation that unites these two examines is that they both aim to close the gap that exists between theoretical modeling and practical DC motor control applications. With implications for applications ranging from robotics to industrial automation, the integration of experimental setups, system identification, transfer function estimate, and Simulink block diagrams highlights a methodical and efficient approach to achieving accurate position control.

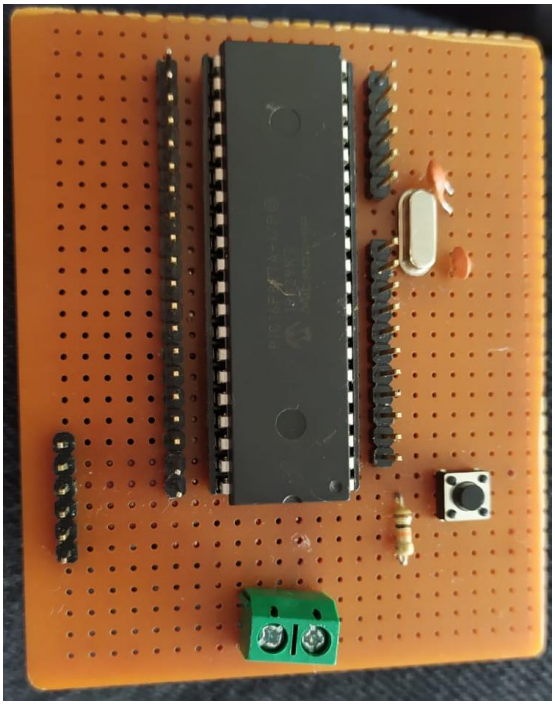
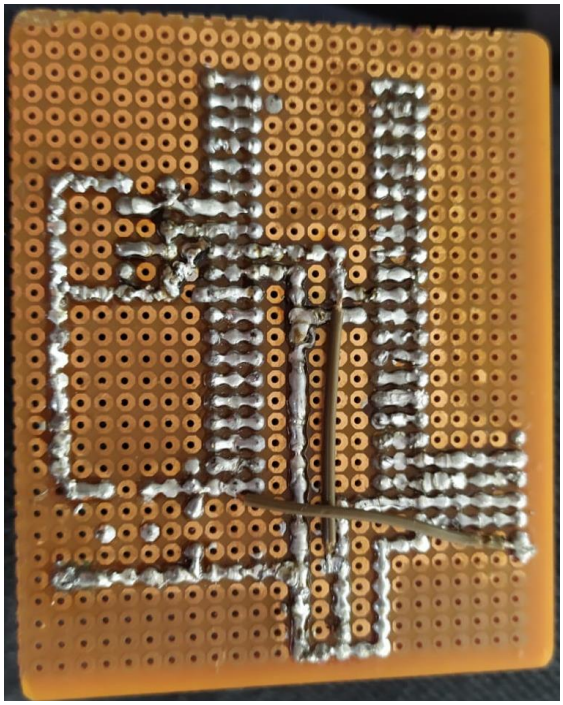
Hardware

In the next assembly step, we solder the PIC16F877A microcontroller onto its card, which contains extra header pins for PIC Kit programming and other components. Concurrently, the independent power card, which has a more straightforward design, has a 12-volt input and produces separate 5-volt and 12-volt outputs. Two safety measures are a 7805 5V Regulator to regulate voltage and a fuse socket with a 2-ampere fuse for motor driver protection. It's important to emphasize that the power cards and microcontroller both work independently and have distinct roles to play within the system as a whole.





(Fig B.1 and B.2: Front and Back of the Power Card)

(Fig B.3 and B.4: Front and back of the Microcontroller Card)

Velocity Extraction

We used DAQ Assistant in combination with LabVIEW. We took two samples out of every waveform using five different duty cycles (20%, 40%, 60%, 80%, and 99%), for a total of 10 samples. A signal generator was used to alter the duty cycles, and an oscilloscope helped to create a 5-volt square wave, guaranteeing a voltage range of almost zero to five volts.

With MatLab code given below:

```
clc
clear all
close all

% Load data from Excel file
data = readmatrix('D99_1.xltx'); % replace 'your_file.xlsx' with the actual file
name

% Extract data excluding the header row
time = data(1:end, 2); % the time column is in the second column
amplitudeVoltage = round(data(1:end, 3)); % the amplitude voltage column is in the
third column

% Initialize the previous time
prevTime = time(1);

prevIndex = 0;

flag = 0;

% Initialize angular velocities array
angularVelocities_99 = zeros(size(time));

% Iterate through the data to find instances where the difference is -5 or 5
for i = 2:length(amplitudeVoltage)
    if ((amplitudeVoltage(i) == 0 && amplitudeVoltage(i-1) == 5))
        % First detection == should be used only once
        if (flag == 0)
            prevIndex = i;
            prevTime = time(i);
            flag = 1;
            continue
        end

        % Calculate angular velocity
        angularVelocityRadPerSec = ((360/16)*(pi/180)) / (time(i) - prevTime);


        % Store the calculated or substituted angular velocity
        angularVelocities_99(prevIndex+1:i) = angularVelocityRadPerSec;

        % Update the previous time for the next iteration
        prevIndex = i;
        prevTime = time(i);
    end
end


% Plot the results
figure;
plot(time, angularVelocities_99);
xlabel('Time');
ylabel('Angular Velocity (rad/s)');
title('Angular Velocities vs Time');
grid on;
hold on;

for i = 1: length(angularVelocities_99)-1
    if(angularVelocities_99(i+1) - angularVelocities_99(i) < -80)
```

The pulse-width modulation (PWM) frequency was determined by the formula:

$$Tpwm = 4 \times [(PR2) + 1] \times [TMR2\ Prescale\ Value] \times Tosc$$

$$fosc = fcrystal/4 = 16000000/4 = 4000000$$

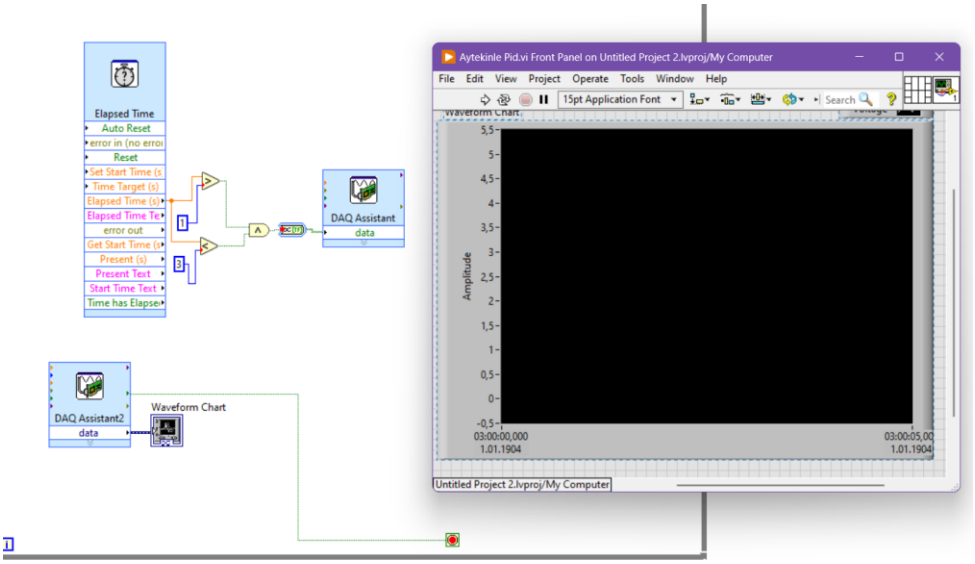
$$Tpwm = 4x256x4x(1/4000000)$$
$$= 0.001024\ so\ the\ frequency\ of\ the\ PWM\ is\ 1/0.001024\ =\ 977\ Hertz$$

977 Hertz was determined to be the PWM frequency using particular parameter values. A order to read five thousand samples was given to DAQ Assistant. Taking into account the encoder's two channels, which undergo 32 modifications every revolution, its maximum rotational speed of 10,000 rpm yielded a frequency of roughly 5,000 Hertz. The ultimate frequency was determined to be 50,000 Hertz after Nyquist criteria adjustments.

The motor driver's start-stop pin was regulated by a simple logic circuit, which added a 1-second delay to the power generation process in order to guarantee correct data collecting. Data was gathered in a span of one to three seconds. Analog encoder channel readings were graphically represented in LabVIEW, and time, voltage, and amplitude data were exported into Excel tables for each duty cycle (20%, 40%, 60%, 80%, and 99%). Information for

Duty Cycle 20	2,33V
Duty Cycle 40	4,67V
Duty Cycle 60	6,97V
Duty Cycle 80	9,89V
Duty Cycle 99	11,47V

	A	B	C
52537	30:37,3	1,05070	0,0881886
52538	30:37,3	1,05072	0,087862
52539	30:37,3	1,05074	0,087862
52540	30:37,3	1,05076	0,0881886
52541	30:37,3	1,05078	0,0872089
52542	30:37,3	1,05080	0,0868824
52543	30:37,3	1,05082	0,0875355
52544	30:37,3	1,05084	0,0872089
52545	30:37,3	1,05086	0,0872089
52546	30:37,3	1,05088	0,087862
52547	30:37,3	1,05090	0,0845965
52548	30:37,3	1,05092	0,0842699
52549	30:37,3	1,05094	0,0836168
52550	30:37,3	1,05096	0,0845965
52551	30:37,3	1,05098	0,0839434
52552	30:37,3	1,05100	0,0845965
52553	30:37,3	1,05102	0,0845965
52554	30:37,3	1,05104	0,0855762
52555	30:37,3	1,05106	0,0845965
52556	30:37,3	1,05108	0,0852496
52557	30:37,3	1,05110	0,0855762
52558	30:37,3	1,05112	0,0852496
52559	30:37,3	1,05114	0,0862293
52560	30:37,3	1,05116	0,0862293
52561	30:37,3	1,05118	0,0868824
52562	30:37,3	1,05120	0,0859027
52563	30:37,3	1,05122	0,0862293
52564	30:37,3	1,05124	0,0865558
52565	30:37,3	1,05126	0,0865558
52566	30:37,3	1,05128	0,0872089
52567	30:37,3	1,05130	0,0881886
52568	30:37,3	1,05132	0,0901479
52569	30:37,3	1,05134	0,087862
52570	30:37,3	1,05136	5,06946
52571	30:37,3	1,05138	5,07632
52572	30:37,3	1,05140	5,07893
52573	30:37,3	1,05142	5,08122
52574	30:37,3	1,05144	5,07991
52575	30:37,3	1,05146	5,07861
52576	30:37,3	1,05148	5,08318
52577	30:37,3	1,05150	5,08089
52578	30:37,3	1,05152	5,0822
52579	30:37,3	1,05154	5,08089
52580	30:37,3	1,05156	5,07991
52581	30:37,3	1,05158	5,07763
52582	30:37,3	1,05160	5,0822
52583	30:37,3	1,05162	5,07959
52584	30:37,3	1,05164	5,08024
52585	30:37,3	1,05166	5,08057
52586	30:37,3	1,05168	5,0773
52587	30:37,3	1,05170	5,07763
52588	30:37,3	1,05172	5,07828
52589	30:37,3	1,05174	5,0773

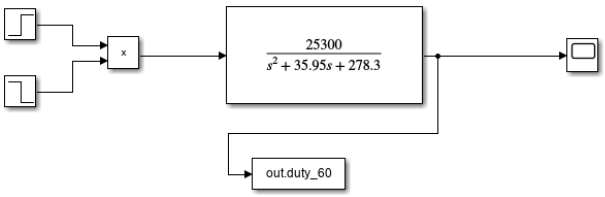


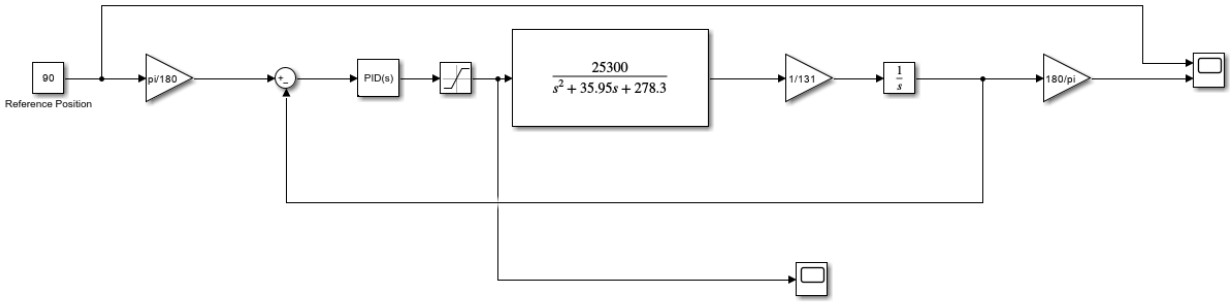
System Identification

The process of determining the mathematical connection between inputs and outputs is known as system identification. This usually entails fitting the data to a mathematical model, which is frequently represented as a state-space representation or transfer function. We combined input voltages at different duty cycles (20%, 40%, 80%, and 99%) with matching output velocities in our system identification method. By combining these information, we were able to build a transfer function that captures the link between output velocity and input voltage.

```
tf1 =
  From input "u1" to output "y1":
         2.53e04
  ---------------------
  s^2 + 35.95 s + 278.3
```

The model that is found results in the estimation of a transfer function that connects the output's Laplace transform to the input's and captures the dynamic behavior of the system. We created a Simulink graph for a 60% duty cycle using this transfer function that was obtained from system identification.





With the use of a proportional-integral-derivative (PID) controller, exact position control is the desired outcome. In order to reduce the error between the reference and feedback locations, we set a reference position, convert it to radians, and then apply a PID controller. We introduce saturation limitations (+12 to -12) for real-world applications. An integrator aids in achieving the intended position while a gear ratio modifies the output shaft's velocity. PID values are adjusted to optimize parameters such as rising time, peak time, overshoot, and settling time by examining graphs of the reference and output positions.

Caliberation of the Potentiometer

The potentiometer is calibrated within a predetermined angle range by measuring resistance values and translating them to voltage values between 0 and 180 degrees in order to guarantee accurate functionality. The technique encompasses initial and maximum position measurements, computing voltage values, establishing a voltage-bit ratio, plotting a voltage-bit graph, and incorporating the resulting values into the project's code for exact potentiometer readings and effective position control.

Our potentiometer was 930Ω resistor. So we need to make ratio proportion within 930Ω, 5V passes.

Potentiometer was 201Ω when our pointer is at 0 degrees and the potentiometer was 827Ω when our pointer is at 180 degrees so:

$$201 \times \frac{5V}{930} = 1.08V \qquad 827 \times \frac{5V}{930} = 4.45V$$

To convert them to bits

$$1.08V \times \frac{1023}{5} = 220bit \qquad 4{,}45V \times \frac{1023}{5V} = 910bit$$

To find slope in y = mx + n form

$$\frac{180°}{910 - 220} = 0.26 = m$$

$$220 \times 0.26 = 57.2$$

$$\boldsymbol{y = 0.26x + 57.2}$$

Implementing of the Results

With the code:

```
#include <16F877A.h>
#device ADC=10
#include <stdlib.h>
#FUSES NOWDT  NOBROWNOUT NOLVP XT,NOPROTECT,NOPUT, NODEBUG, NOCPD
#use delay(crystal=16000000)
#use rs232 (baud=9600,xmit=PIN_C6, rcv=PIN_C7, parity=N, stop=1) // configuretion
for serial communication
#define DIRECTION PIN_B1
float refAngle = 90.0f;
float revAngle = 0.0f;
unsigned long int result_1;
signed long controlOut=0;
unsigned long int pwmOut;
float error = 0.0f;
float prevError = 0.0f;
float integral = 0.0f;
float derivative = 0.0f;
float dt = 0.02f;
unsigned long int Z;
unsigned long int T;
float kp = 0.22f;
float ki = 0.13f;
float kd = 0.018f;

int16 elapsedTime;
int16 remainingTime;

#int_timer0
void tmr_int()    {
 set_timer0(236);
 Z++;
 T++;
 /*if (Z >= 10)    {
    error=refAngle-revAngle;
    integral=integral+dt*error;
    derivative = (error-prevError)/dt;
    controlOut = error*kp+ki*integral+kd*derivative;
    Z = 0;
    prevError = error;
 }*/
}

void main()  {

 port_b_pullups(TRUE);
 enable_interrupts(GLOBAL);
 clear_interrupt(int_ext);
 setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256);
 set_timer0(236);
 enable_interrupts(int_timer0);
 enable_interrupts(int_ext);
 setup_adc_ports(AN0_AN1_AN3); //A0 A1 A3 are configured for analog input pin
 setup_adc(ADC_CLOCK_DIV_32); //enable ADC and set clock for ADC
 setup_ccp1(CCP_PWM); //4kHz PWM signal output at CCP1 pin 17
 setup_timer_2(T2_DIV_BY_16, 255, 1);

 set_tris_b(0x00);
 output_b(0x00);
```

1. **Microcontroller Configuration:**
- It is configured for a PIC16F877A microcontroller.
- The ADC (Analog-to-Digital Converter) is set to 10-bit resolution (#device ADC=10).

2. **Configuration Settings:**
- Various configuration settings are specified using #FUSES. These include settings related to watchdog timer, brownout, low-voltage programming, crystal oscillator, code protection, power-up timer, and debugging features

3. **Serial Communication:**
- Serial communication is set up using the #use rs232 directive with a baud rate of 9600, transmit pin (TX) as PIN_C6, and receive pin (RX) as PIN_C7.

4. **PID Controller Parameters**:
- Proportional (kp), integral (ki), and derivative (kd) constants are defined.
- dt represents the time step for the control loop.

5. **Timer0 Interrupt**:
- An interrupt routine (tmr_int()) is defined using #int_timer0.
- It increments counters Z and T, and there is commented-out code for PID calculations.

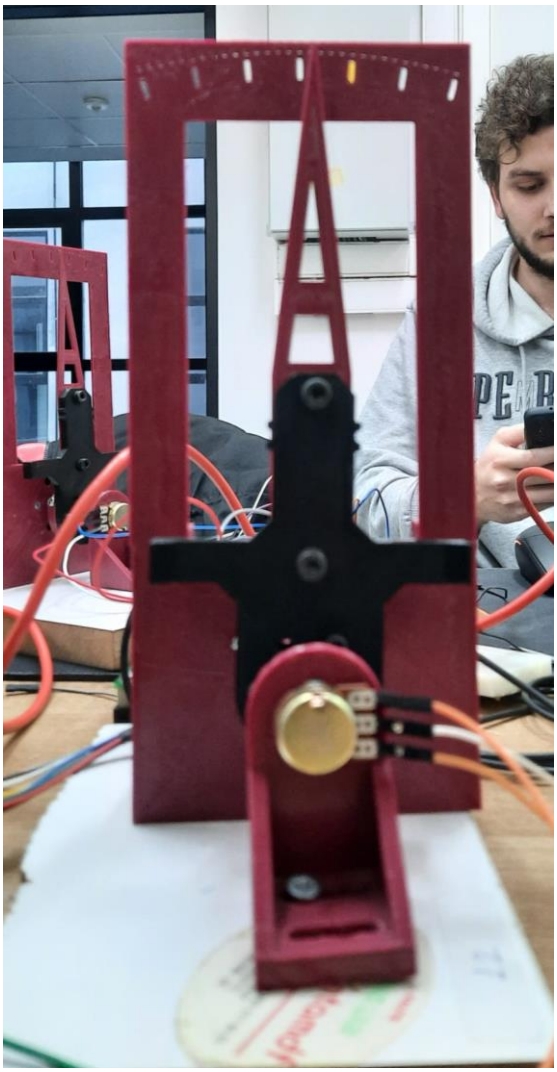
6. **Main Function**:
- The main() function initializes various settings, sets up the ADC, PWM, and timers.
- It enters a continuous loop (while(TRUE)) for the main control logic.
- At this part we Implement our parameters into a code **(row 67)**

7. **ADC and Motor Control**:
- Analog readings are taken from channel 0 (AN0) using the ADC.
- The angle (revAngle) is calculated based on the ADC reading.
- PID control is applied to compute the control output (controlOut).
- The motor direction and PWM duty cycle are adjusted based on the control output.
- There are limits set for the control output (controlOut) to avoid exceeding certain values.

8. **Timing and Delays:**
- Timing-related variables (elapsedTime and remainingTime) are used to control the overall loop execution time.
- The program uses a delay (delay_ms) to control the loop execution time.

9. **Serial Output**:
- Serial output statements using printf are present to display information, including the current angle (revAngle), on the serial port.

Output and Conclusion



Starting with the application of PID control for a DC motor with an incremental encoder, our project used a Microcontroller (PIC16F877A) and a Power Card in order to prioritize practicality. As part of this project, LabVIEW and DAQ Assistant were used to obtain critical voltage and time data from the encoder. Accurate velocity measurements were obtained by processing the data using a MATLAB code that explored different motor velocities throughout different duty cycles. By combining four input-output pairs for validation, system identification and transfer function estimation were carefully carried out, leading to a comprehensive understanding of the system's dynamics.

A position control block diagram was created using Simulink, and PID values were determined via simulation. The use of PID control in practice confirmed the validity of the theoretical model and produced useful insights into PID tuning and control systems in addition to good results. Our understanding and skill set for upcoming projects were improved by the iterative process of going from theory to application, which proved to be both fascinating and enlightening.

In conclusion, we have learned a great deal from our investigation into PID control for a DC motor using an incremental encoder. In order to navigate the complexity of the hardware and coding worlds, MATLAB was essential. Our ability to transfer mechanical power was improved by overcoming mechanical obstacles in the control portion. The robustness of the established PID control system was demonstrated by the project's successful achievement of precisely stopping the needle at ninety degrees within the restricted range of zero to one hundred and eighty degrees. The project effortlessly combined a Microcontroller and a Power Card for real-time PID control, with LabVIEW and DAQ Assistant performing essential roles in data collecting. Our comprehension of the dynamics of the system was improved by MATLAB processing and system identification. In addition to achieving the project's goals, the iterative process—which included data collection, MATLAB simulation, and the final implementation of PID control in real-world settings—also deepened our understanding of control systems and PID tuning. Our excitement for next initiatives is fueled by this success, as we look forward to using and developing our newly acquired knowledge and abilities. All things considered, the PID Control course project has been a rewarding and instructive experience that has prepared me for exciting future endeavors in the field of control systems and beyond.