EE324: Controls System Lab

Experiment 1: DC Motor Position Control

Team Members :
Aman Rishal C H (22B3914)
Hrishikesh S (22B4217)
Prajwal Nayak (22B4246)

Thursday Batch - Table 14

September 5, 2024

1 Aim

Design and implement a PID position control system using Arduino Mega.

2 Objective of the Experiment

- To rotate the DC motor by an angle 180 degrees from an arbitrary initial position.
- To ensure that the rotation happens successfully within the constraints of a 0.5-second rise time, a 1-second settling time, and less than 10% overshoot.

3 Control Algorithm

3.1 Theory

We have used a PID-controller in order to dictate the Control Law for the given system. A PID (Proportional Integral Derivative) controller is having 3 control blocks as listed below :

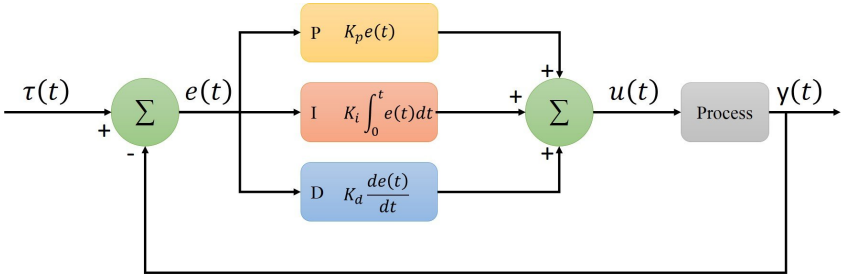

Figure 1: PID Controller

- **Proportional :** The proportional controller block is responsible for providing an output which is proportional to the difference between the target value and the current value at each stage of the transient stage.
- **Derivative :** The derivative controller plays a crucial role by responding to the rate at which the error is changing with respect to time and thus produces an output corresponding to the rate of change of error to effectively control the system. This term helps to reduce overshooting and oscillations by anticipating and correcting for rapid changes in the process variable.
- **Integral :** The integral controller accumulates the error over time, producing an output proportional to the total accumulated error, which helps to eliminate steady-state error in the system.

3.2 Circuit

3.3 Logic For Control Algorithm

1. Just after building the circuit, we check for the non-linear region. We got the non linear region to be from around 355°to 5°. We avoid this region.

2. On reading from the potentiometer, we get a value proportional to the initial angle θ of the motor.

3. We define the final value as θ + 180°, if θ is greater than 180°or θ - 180°; if θ is smaller than 180°. This is done because we read values from the potentiometer as an integer from 0 to 1023. 180°is equivalent to 511. So, if an angle is greater than 180°i.e, greater than 511, then adding 511 again to the value makes it greater than 1023 which can never be achieved.

4. The error is calculated as the difference between the final and the initial value.

5. To start off, we give random values to K_p, K_d and K_i. The proportional term is defined as the product of K_p and error, the derivative term is defined as the product of K_d and change in output = product of K_d and change in error = product of K_d and (present error - previous error), and the integral term is defined as the product of K_i and (summation of all errors). The total feedback value is computed by taking the sum of the proportional, derivative, and integral terms. This feedback acts as a control signal, feeding back to the system as a Pulse Width Modulated value.

6. Let us see how the non-linear region is avoided. The code adjusts the direction of rotation based of the sign of the error. If the error is positive, it increases the angle and if it is negative, it decreases it, which makes sure that it never goes to the non-linear region.

7. The above mentioned mechanism also helps in case of overshoot.

8. After multiple iterations based on logic and, hit ad trial we finally arrived at the following values of K_i, K_p and K_d:

K_p	K_d	K_i
2.5	1	0.009

3.4 Arduino Code

```
int motor_pin_input1 = 5;
int motor_pin_input2 = 6;
int enable_pin = 4;
int potentiometer_out = A4;
int zero = 0;
int sensor_value, output_value, target_value, previous_error;
int difference, integral;
float kp = 2.5, kd = 1, ki = 0.009;
int callibration_constant = 511;
void setup() {

```

```
12    Serial.begin(9600);
13    pinMode(motor_pin_input1,OUTPUT);
14    pinMode(motor_pin_input2,OUTPUT);
15    pinMode(potentiometer_out,INPUT);
16
17    sensor_value = analogRead(potentiometer_out);
18
19    if (sensor_value > callibration_constant)
20    {
21      target_value = sensor_value - callibration_constant;
22    }
23    else
24    {
25      target_value = sensor_value + callibration_constant;
26    }
27    previous_error = target_value - sensor_value;
28    integral = previous_error;
29  }
30
31  void loop() {
32
33    sensor_value = analogRead(potentiometer_out);
34    int error = target_value - sensor_value;
35    difference = error - previous_error;
36    integral += error;
37
38    output_value = (int)abs(kp*error+kd*difference+ki*integral);
39    if(output_value > 255)
40      output_value = 255;
41    if(error < 0)
42    {
43      analogWrite(motor_pin_input1,output_value);
44      analogWrite(motor_pin_input2,0);
45    }
46    else
47    {
48      analogWrite(motor_pin_input1,0);
49      analogWrite(motor_pin_input2,output_value);
50    }
```

```
51    previous_error = error;
52    abs(error))*180/callibration_constant);
53    Serial.print(error);
54    Serial.print('\n');
55    delay(10);
56  }
```

4 Problems Faced

1. In the initial phase, we faced difficulty in figuring out the non-linear region of the potentiometer as the potentiometer wasn't rotating along with the motor sometimes giving us non-linear region for a huge range of angles. We had the motor changed and the problem was solved.
2. After compiling the code successfully we were not able to make the motor to settle at the settling point. We changed the output to Analog Write instead of Digital Write in the code.
3. Finding the perfect values of the controller was the biggest problem because for most values, the motor would either oscillate around the final value or the speed would be so high that it wouldn't stop. Fine tuning of the values took quite some time.
4. There also seemed to be a problem with either the Arduino, the port of the Arduino or the IDE, because the IDE would suddenly indicate that the Arduino is disconnected, and would stop reading values and controlling the motor.

5 Results Obtained

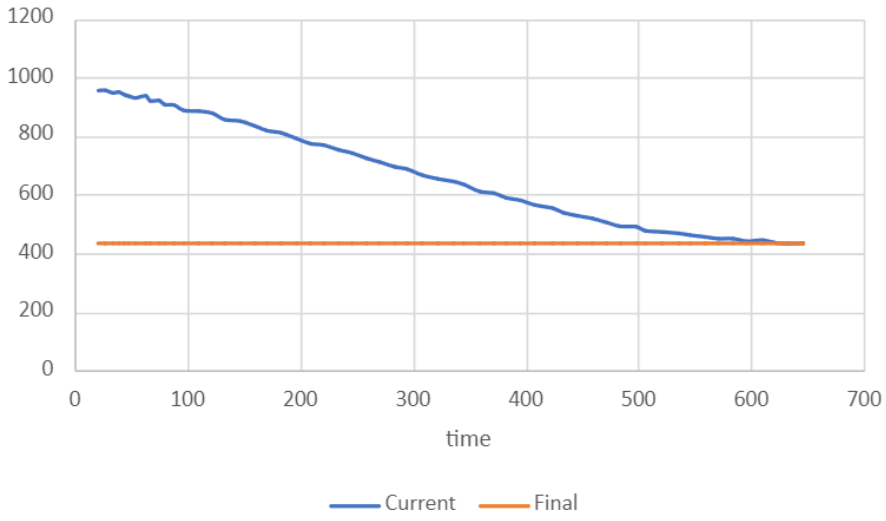

Figure 2: Motor Response

The motor has started from 945, and it has stabilized at 432.

6 Calculations

1. **Rise Time:** The rise time of the system is defined as the time taken for the system to go from 10% to 90% of the target final position. In this case, it is the time taken by the DC motor to rotate from reading 894 to 486.
   The rise time was observed to be **412 ms**.

2. **Settling Time:** The settling time is the time taken for the system's response to stabilize within a small percentage of the setpoint or final value, which in this case is 180 degrees from the initial position. We choose the settling time to be within 5% of the setpoint i.e. the time it takes to reach the range 409 to 459. The settling time was observed to be **479 ms**.

3. **Percentage Overshoot:** The overshoot is the maximum value that the system exceeds the setpoint before it stabilizes. The Overshoot Percentage was calculated using the following formula:

$$Overshoot(\%) = \frac{|y_{max} - y_{final}|}{y_{final}} \times 100\%$$

   In this case, $y_{final} = 434$, and $y_{min} = 428$.

$$Overshoot(\%) = \frac{|428 - 434|}{434} \times 100\% \approx 1.38\%$$

These results indicate that the design specifications were successfully met.

7 Observation and Inference

Through this experiment, we were able to understand the working of a PID controller and how it can be used to achieve a good level of precision when used to control the DC motor. In order to achieve the desirable response and to meet the design constraints, one needs to be careful while tuning the parameters for each of the terms. After carefully selecting the parameters, we observed the impact of each of the constants on the behavior of the system.

- K_p
  On increasing K_p, we are able to reduce the rise time, but only at the cost of also increasing the overshoot percentage and increasing oscillations.

- K_I
  K_I was found to be vital in reducing the steady-state error, but if not tuned properly, it could lead to more oscillations.

- K_D
  K_D helps to control the oscillations and overshoot and adjust the response time of the system.