EE 324: Experiment 1
DC Motor Position Control

Sravan K Suresh	22B3936
Swarup Dasharath Patil	22B3953
Amol Milind Pagare	22B3971

October 10, 2024

Contents

1 Objective

Design and implement a PID position controller using Arduino Mega with the following goals:

- Rotate the DC motor by an angle of 180 degrees from any given starting point.
- Ensure the design meets the following constraints:
  - **Rise time:** 0.5 seconds
  - **Settling time:** 1 second
  - **Overshoot:** Limited to 10%

2 Control Algorithm

To develop a PID control algorithm for rotating a DC motor by 180 degrees, we define the system's input as the motor's speed, which is controlled by adjusting the voltage across its terminals. The output is obtained from a Servo Pot potentiometer, which provides the motor's angle of rotation, mapped from 0 to 1023, corresponding to 0 to 360 degrees. The potentiometer has a non-linear region, which we identified between 84 and 97 degrees on the 360-degree dial.

Our objective is to rotate the motor by 180 degrees, with a target potentiometer value of 512. To avoid the motor entering the non-linear region, we first read the initial position of the dial. Based on this position, the motor is rotated in the direction opposite to the non-linear region.

In the implementation, we calculate the error as the difference between the current potentiometer position (read using `analogRead()`) and the target position.

Using this error, we construct a PID controller where the output variable represents the motor speed, which is then used to control the motor.

We also track the time elapsed since the start of the program, which is crucial for calculating the derivative and integral terms.

- **Proportional Controller:** The proportional term is calculated as $k_p \times \text{error}$.
- **Derivative Controller:** The derivative term is calculated as $k_d \times \frac{\text{change in error}}{\text{time interval between samples}}$.
- **Integral Controller:** The integral term is calculated as $k_i \times (\text{error} \times \text{time interval between samples})$.

The final control signal, stored in the variable `output`, is calculated as:

```
int output = int(k_p*error + k_i*integral + k_d*derivative)
```

This `output` value is then used to control the motor speed.
The Arduino code we used is given below-

```
int pot=0;
bool initial = false;
int init_angle;

float k_p =1;
float k_i =0.00001;
float k_d =0.01;
int target = 0;

int error = 0;
int prev_error = 0;
int integral = 0;
int derivative = 0;

int time = 0;
int prev_time = 0;
int dt = 0;


void setup() {
  Serial.begin(9600);
  pinMode(4,OUTPUT);
  pinMode(5,OUTPUT);
  pinMode(A0,INPUT);
}

void loop() {
time =millis();
dt = time - prev_time;
pot = analogRead(A0);

int angle=pot;
  if(!initial){
    if(angle < 512){
      target = angle + 512 - 60;
    }
    else{
      target = angle - 512 + 60;
    }
      initial = true;
      init_angle = angle;
  }
  error = target - angle;

  integral = error*dt;
  derivative = (error - prev_error)/dt;
  prev_error = error;

  int output = int(k_p * error + integral*k_i + derivative*k_d);
  if (abs(output) > 255)
  {
    if (output>0) output = 255;
    else output = -255;
  }

  if (init_angle>10 && init_angle<504){
    if (output<0){
      analogWrite(4,abs(output));
      analogWrite(5,0);
    }
    else{
      analogWrite(4,0);
      analogWrite(5,output);
    }
  }
  else if (init_angle>522 && init_angle<1015){
    if (output>0){
      analogWrite(4,0);
      analogWrite(5,output);
    }
    else{
      analogWrite(4,abs(output));
      analogWrite(5,0);
    }
  }
  prev_time = time;
}
```
Listing 1: Arduino PID Control Code

3 Challenges Encountered

- Our DC motor was changed in between the experiment so we had to find the non-linear region of the new given motor. We found out the non linear region and also figured it out that the only thing that matters is in which half does the given angle lies.
- Tuning the PID control parameters was a big challenge as it was to be done through trial and error. We figured out the required parameters through this iterate and reiterate process.

4 Results

PID controller parameters:-

- $K_i = 10^{-5}$
- $K_d = 10^{-2}$
- $K_p = 1$

Design Specifications:-

- **Rise time** = 351 ms
- **Settling time** = 543 ms
- **% overshoot** = 2.87 %

All of these parameters fall within our requirements from the controller.Thus, our designed controller is valid.

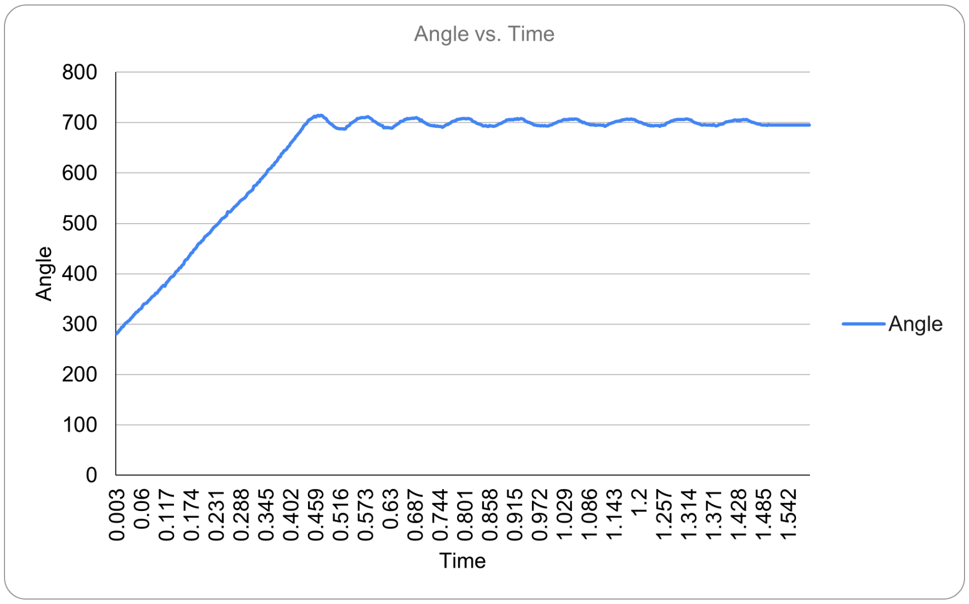

Figure 1: Response

5 Observations and Inferences

- The PID controller successfully achieved the desired performance criteria with a rise time within 0.5 seconds, settling time within 1 second, and overshoot under 10%.
- The final tuned PID parameters were Kp = 1, Kd = 10^{-2}, and Ki = 10^{-5}. These values provided a balance between responsiveness and stability, as observed in the final response plots.