EE324 Experiment 1

Monday Group 16

Jay Mehta (22B1281)
Tanay Bhat (22B3303)
Aagam Shah (22B1201)

August 2024

Contents

Chapter 1

Objective

1.1 Aim of the experiment

- Design PID controller for position control of DC Motor
- Implement the controller using Arduino Mega

1.2 Design Requirements

- Rotate DC motor by exactly 180° from any given starting position
- Rise Time must be less than 0.5s
- Settling Time must be less than 1s
- % overshoot must be less than 10%

1.3 Data Flow Block Diagram

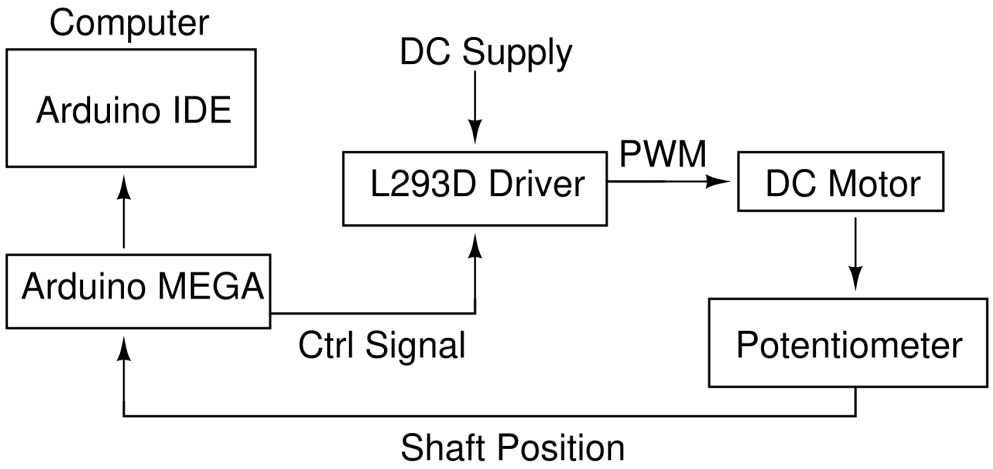

Figure 1.1: Data Flow

Chapter 2

Control Algorithm

2.1 General PID Control

The characteristics of a closed-loop system can be controlled based on the error (proportional controller), the cumulative error (integral controller), and the rate of change of error at a given instant (derivative controller). Cumulatively, they constitute the PID controller. We may denote this controller as:

$$h(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{2.1}$$

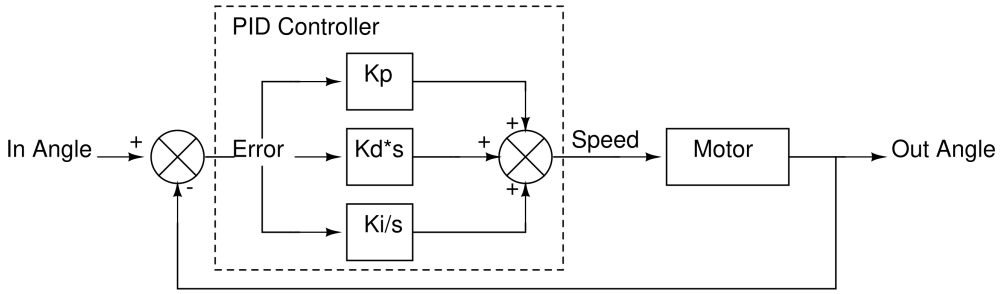

Figure 2.1: PID Control Block

This type of controller is not easily realizable (specifically the integral and differential parts) as the input error function may not be known beforehand. To combat this, we settle for some approximations:

$$e_{integral}(t) \approx K_i * \sum_{i=0}^{t} e(i) * \Delta t \tag{2.2}$$

$$e_{derivative}(t) \approx K_d * \frac{e(t) - e(t-1)}{\Delta t} \tag{2.3}$$

Usually, Δt is combined with the constants K_i and K_d; hence, it may be dropped. The P-controller allows us to control the general rate at which the system attains steady state, the D-controller enables adjustment of transient parameters like rise time, and settling time and the I-controller helps reduce steady-state error. We adjust these parameters iteratively to meet our requirements.

2.2 Approach

We shall approach this problem in the following manner:

- Firstly we determine the region of non-linearity for the DC motor. This means that the data read by the sensor in the motor box varies non-linearly with the input (which in this case is the shaft angle). We must ensure that the motor does not go through this region during rotation as it may lead to erroneous results.
- Next we test the DC motor to ensure that it's working properly. We do this by rotating the motor in both directions. Once this is done we determine the target sensor value that occurs at 180° from the starting position.
- We now implement the PID controller in code using the above approximations and adjust the constants K_p, K_d and K_i to meet our requirements.

The region of non-linearity was found between $\theta \in [97, 104]$. With this in mind, all starting positions $\theta_{start} \in [97, 104] \cup [277, 284]$ are forbidden. We should also be careful for starting positions that end near the non-linear band as the system may malfunction if it overshoots into this band while settling.

We then test the working of the motor using the following functions:

```
// Rotate the motor anticlockwise with speed = motorSpeed
void antiClockWise(int motorSpeed = 255){
  analogWrite(pinBlack, motorSpeed);
  analogWrite(pinRed, 0);
}

// Rotate the motor clockwise with speed = motorSpeed
void clockWise(int motorSpeed = 255){
  analogWrite(pinBlack, 0);
  analogWrite(pinRed, motorSpeed);
}

// Stop the motor for delay_value milliseconds
void halt(int delay_value = 1000){
  analogWrite(pinBlack, 0);
  analogWrite(pinRed, 0);
  delay(delay_value);
}
```

Listing 2.1: Functions to control Motor

After verifying the working of the DC motor, we calculate the final sensor readings as follows

```
void setup(){
    // Rest of the setup code is not listed here
    // 4 = Sensor value at 104 degrees, 1017 = Sensor value at 97 degrees
    // Hence approx slope in the linear region is around (1017 - 4)/(360 - 7) = 2.87
    // finalValue = 180*2.87 = 517
    // start angle decides the direction of rotation
    outValue = analogRead(inputPin);
    if (outValue > 4 && outValue < 500){
        finalValue = outValue + 517;
    }
    else if (outValue <1017 && outValue > 520){
        finalValue = outValue - 517;
    }
}
void loop() {
  // Rest of the loop code is not listed here

  outValue = analogRead(inputPin);
  val = map(outValue, 0, 1023, 0, 359);
}
```

Listing 2.2: Calculation of Final Value

Chapter 3

Challenges Faced

3.1 Direction Control of the Motor

Given the location of the non-linear region, the direction of rotation of the motor should be controlled accordingly. This was achieved as follows:

```
void loop() {

  // PID Controller implemented
  // All variables were declared as global
  curr_error = outValue - finalValue;
  error_sum += curr_error;

  proportional = kp*curr_error;
  integral = ki*error_sum;
  derivative = kd*(curr_error - prev_error);

  motorSpeedSigned = proportional + integral + derivative;

  // Direction control
  // This was found experimentally
  // +ve value -> clockwise, -ve value -> anticlockwise
  if (motorSpeedSigned>0){
    motorSpeed = motorSpeedSigned;
    if (motorSpeed > 255){
      motorSpeed = 255;
    }
    clockWise(motorSpeed);
  }
  else{
    motorSpeed = -motorSpeedSigned;
    if (motorSpeed > 255){
      motorSpeed = 255;
    }
    antiClockWise(motorSpeed);
  }

  prev_error =  curr_error;
  delay(1);
}
```

Listing 3.1: PID and Direction Control

3.2 Scaling the Input Data

The sensor data was scaled as defined above which made it easier to work with.

3.3 Parameter Tuning

We started by adjusting the K_p value and got the following graph:

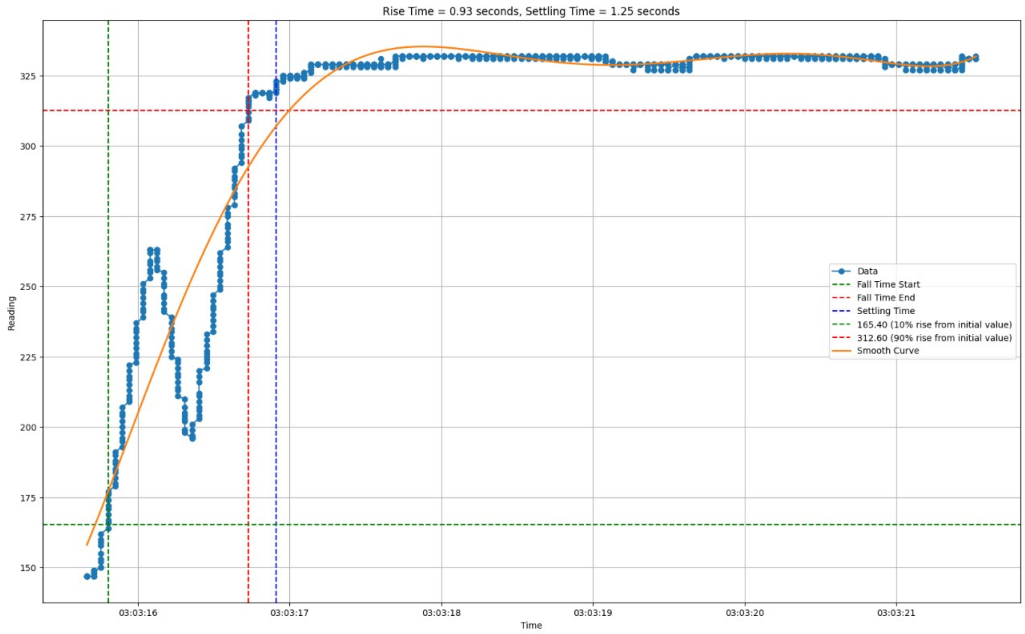

Figure 3.1: Run 1: Only P-Control

We adjusted the K_d value until the required transient characteristics were obtained. The following results were observed:

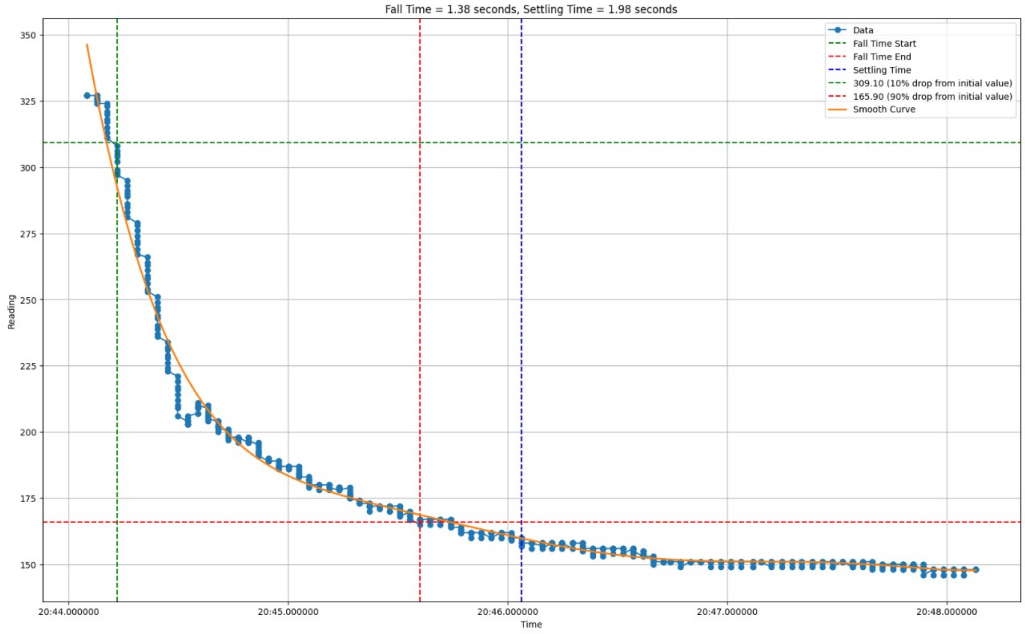

Figure 3.2: Run 2: PD-Control

The increase in K_d led to the graph not settling properly hence we increased K_i to see how the curve would change:

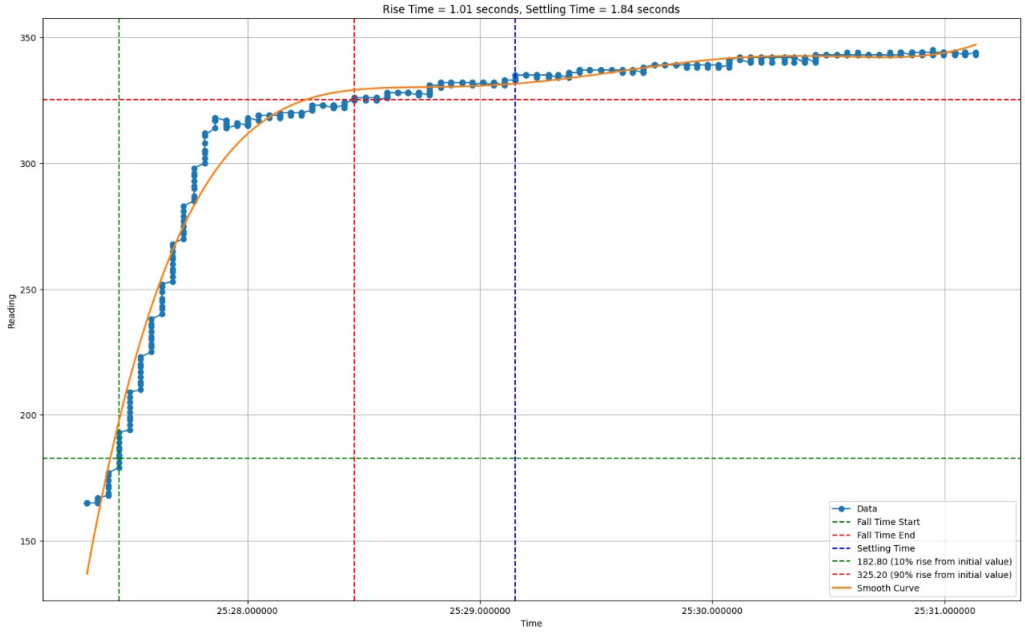

Figure 3.3: Run 3: PID-Control

The increase in K_i resulted in proper settling however, the transient characteristics were now incorrect. Hence K_d was reduced:

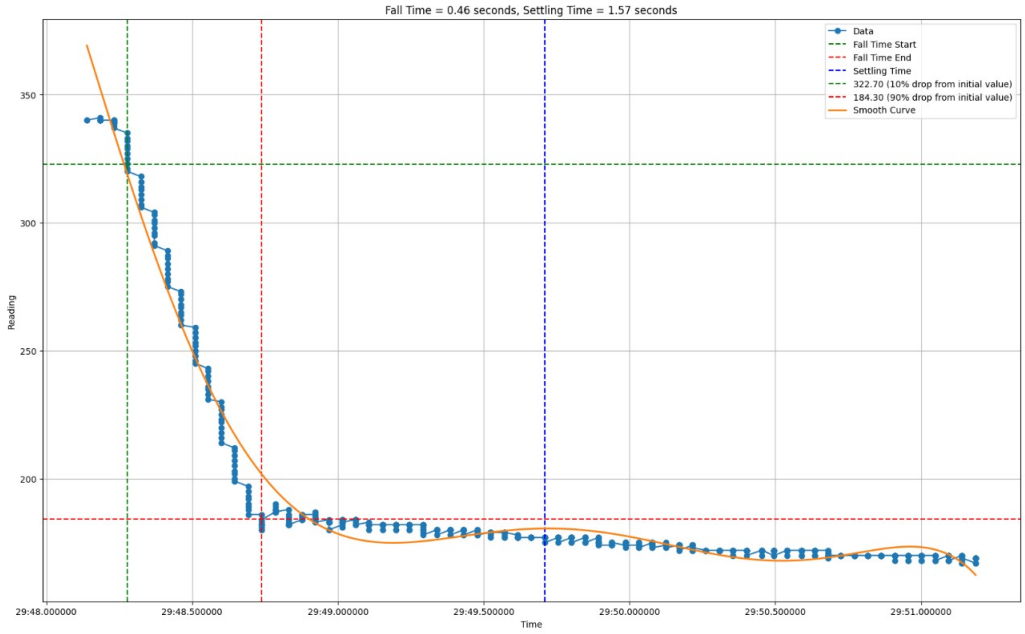

Figure 3.4: Run 4: PID-Control

Chapter 4

Result

The values of K_p, K_d and K_i were iteratively adjusted until we arrived at:
(i) $K_p = 22.5$ (ii) $K_i = 0.01$ (iii) $K_d = 30$
These values result in the following graph:

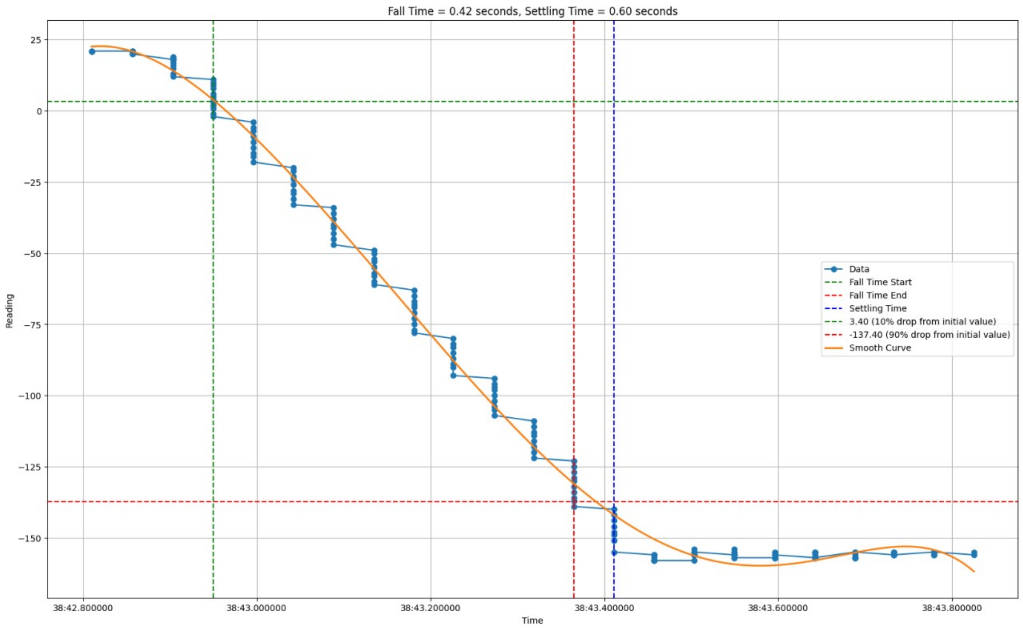

Figure 4.1: Final Results

The following table documents the transient characteristics of our system:

Specifications	Expected Values	Experimental Values
Rise Time (10% - 90%)	0.5s	0.42s
Settling Time (5%)	1s	0.60s
% Overshoot	10%	1.11%

Table 4.1: Design Specifications and experimental values

Chapter 5

Observation and Inference

Even though the transient characteristics were met, the system didn't always settle at exactly 180° from the starting position. During clockwise motion, the steady state error was small ($\approx 2°$) while during anticlockwise rotation from the start, the system showed a steady state error of approximately 5°. In conclusion, even though our system is able to meet the given requirements, it also shows some small steady-state error which can be reduced by further tuning the parameters.