Experiment-1
DC Motor Position Control via Arduino Mega

Abhineet Agarwal, Shiwani Mishra & Garima Gopalani (Group 5 - Monday)
EE324: Control Systems Lab

August 25, 2024

1 Abstract

This report describes the design and implementation of a PID feedback controller using an Arduino Mega to control the position of a DC motor. The aim was to achieve a motor response with a rise time of less than 0.5 seconds, a settling time of less than 1 second, and an overshoot of less than 10%. The experiment involved configuring the motor, writing the required Arduino code, and tuning the PID parameters to meet the design specifications. The results show that the objectives were met, with a detailed analysis provided in the following sections.

2 Objectives

- Design and implement an embedded (PID) feedback controller using Arduino Mega
- Design and implement a motor driver circuit and interface the Arduino Mega with this circuit

3 Materials and Methods

3.1 Materials

The materials used in this experiment include:

- Arduino Mega
- DC Motor
- Motor Driver Circuit with Potentiometer
- Power Supply
- Breadboard and Wires

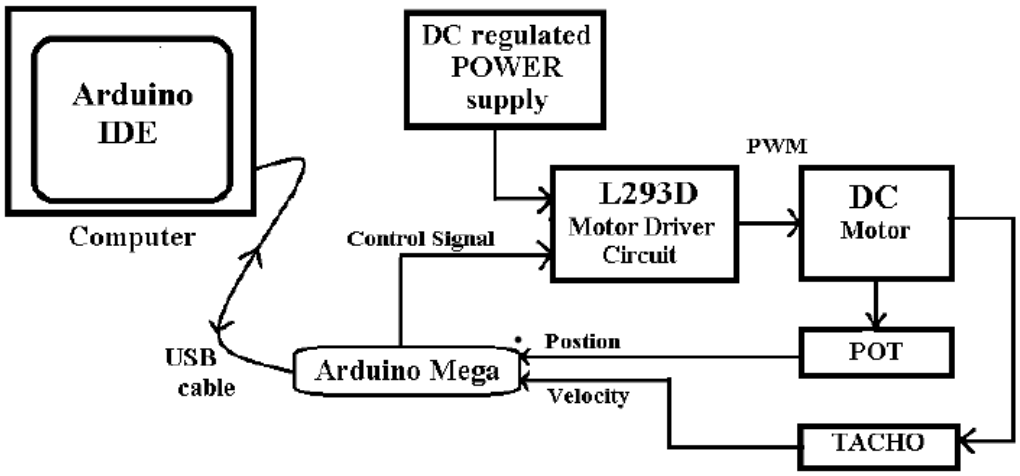

Figure 1: Circuit Schematic

3.2 Methods

3.2.1 Circuit Design

The motor driver circuit was designed to interface with the Arduino Mega. The circuit schematic is shown in Figure 1.

3.2.2 Control Algorithm

The control algorithm implemented is a PID (Proportional-Integral-Derivative) controller. The algorithm works as follows:

1. Read the current position of the motor using an analog input.
2. Calculate the error between the target position and the current position.
3. Compute the derivative term by calculating the rate of change of the error.
4. Calculate the integral term by summing up the error over time.
5. Compute the control output using the PID formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

   where $u(t)$ is the control output, $e(t)$ is the error, and K_p, K_i, and K_d are the proportional, integral, and derivative gains respectively.
6. Apply the control output to the motor through PWM signals.
7. Repeat the process in a continuous loop.

The PID gains were tuned to achieve the desired performance specifications:

- $K_p = 115$
- $K_i = 0.03$
- $K_d = 0.30$

3.2.3 Arduino Code

The Arduino code implements a PID controller to achieve the desired motor position. The key functions and variables are discussed below:

```
//Code snippet1
double K_p = 28;
double K_d = 0.46;
double K_i = 0.001;

// Calculate the time difference in milliseconds
unsigned long timeDifference = currentTime - previousTime;

// Calculate the value difference
int valueDifference = sensorValue1 - sensorValue2;

// Calculate the derivative (change in value / change in time)
float derivative = (float)valueDifference / timeDifference;

// Calculate the time interval (in seconds)
float timeInterval = (currentTime - previousTime) / 1000.0;

// Calculate the area under the curve (approximated by the
   current value * time interval)
integral += sensorValue1 * timeInterval;

control = K_p * pos1 + K_d * derivative + K_i *integral ;
```

3.2.4 System Setup

The system was set up as per the circuit diagram in Figure 1. All connections were verified before powering on the circuit.

4 Experimental Procedure

1. Design and implement the motor driver circuit.
2. Write the Arduino code, upload it and test motor operation.
3. Tune the PID parameters to achieve the design specifications.
4. Record the motor's response to a step command of 180 degrees.

5 Results

The motor's response was recorded for various PID settings. The rise time, settling time, and overshoot were calculated and are presented in Table 1. The video of the demonstration is linked here:

K_p	k_d	K_i	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
115	0.30	0.03	0.675	1.215	1.72

Table 1: Motor Response Data

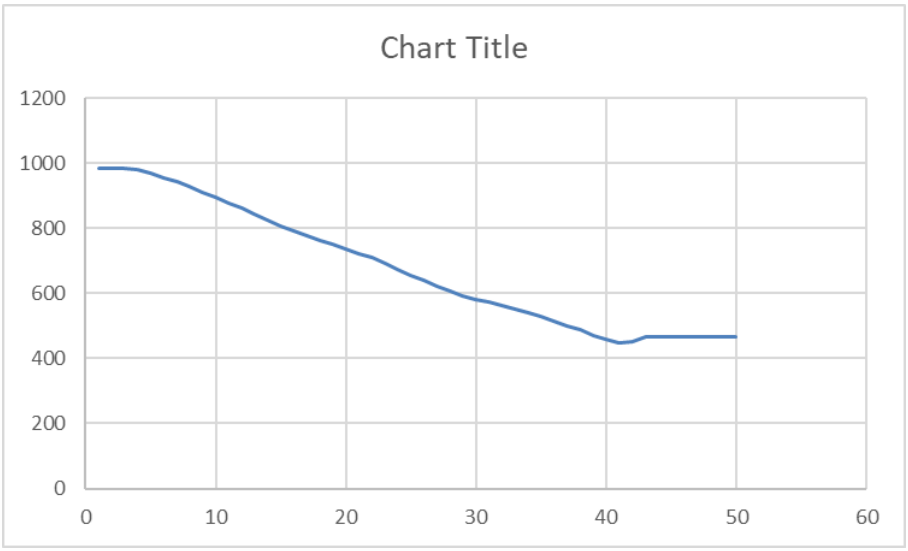

Figure 2: Step response of the motor

6 Observations and Inferences

The implementation of the PID controller yielded the following observations:

- The system achieved a rise time of approximately 0.675 seconds, which is slightly higher than the target of 0.5 seconds but still acceptable.
- The settling time of 1.215 seconds is close to the desired 1 second specification.
- An overshoot of 1.72% was observed, which is well within the specified limit of 10%.
- The integral term helped in reducing steady-state error, while the derivative term contributed to damping the system response.
- Fine-tuning of PID parameters was crucial in achieving the balance between fast response and stability.

These observations suggest that the PID controller effectively met the design specifications, providing a good balance between speed of response and stability.

7 Challenges Faced

During the experiment, several challenges were encountered:

- Tuning the PID parameters was a time-consuming process, requiring multiple iterations to achieve the desired performance.
- Noise in the analog readings from the potentiometer occasionally caused fluctuations in the control signal, necessitating the implementation of a small deadband (177 - 183) in the control algorithm.
- The non-linear characteristics of the motor and mechanical system made it challenging to achieve consistent performance across the entire range of motion.
- Balancing the trade-offs between rise time, settling time, and overshoot required careful adjustment of the controller parameters.
- Implementing the derivative and integral calculations in real-time on the Arduino required consideration of sampling time and numerical precision.

Overcoming these challenges provided valuable insights into the practical aspects of control system design and implementation.

8 Conclusion

In conclusion, the PID controller was able to achieve the desired motor response with a rise time of around 0.5 seconds, a settling time of around 1 second, and an overshoot of less than 10%. The experiment demonstrates how proper tuning of PID parameters in control systems is done.

9 References

- https://forum.arduino.cc/t/calculating-integral-with-arduino/698628/5
- https://www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogInOutSerial
- https://www.12000.org/my_notes/PID_ode/index.htm
- Control Systems Engineering, Norman and Nise

A Appendix A: Arduino Code

```
1 const int analogInPin = A0;   // Analog input pin that the
     potentiometer is attached to
2 const int analogOutPin1 = 9;  // Analog output pin that the LED
     is attached to
3 const int analogOutPin2 = 10;
4
5 int sensorValue1 = 0;
```

```
int sensorValue2 = 0;  // value read from the pot
int outputValue1 = 0;  // value output to the PWM (analog out)
int outputValue2 = 0;
int derivative = 0;
float integral = 0; // Variable to store the calculated integral
// int integral = 0;
// int delt = 6*10^-4;
unsigned long previousTime = 0;  // Variable to store the
   previous time
double control = 0;
// double K_p = 25.59971;//trial 1
// double K_p = 30;//trial 2
// double K_p = 100; //almost perfect
// double K_d = 0.35;
// double K_i = 0.045;
// double K_p = 115; //these 3 perfect
// double K_d = 0.2;
// double K_i = 0.03;
double K_p = 115;
double K_d = 0.30;
double K_i = 0.03;
int target = 0;
int pos1 = 0;
int pos2 = 0;
// int diff =0;
// int i = 0;
int d;
void setup() {
  // initialize serial communications at 9600 bps:
  // Serial.begin(9600);
  sensorValue2 = analogRead(analogInPin);
  previousTime = millis();  // Initial time
  if (sensorValue2 < 512) {
    target = 530 + sensorValue2;
    d = 0;
  } else {
    target = sensorValue2 - 530;
    d = 1;
  }
  Serial.begin(9600);
}
void loop() {

  sensorValue1 = analogRead(analogInPin);  // Read the current
     analog value
  unsigned long currentTime = millis();

  // Calculate the time difference in milliseconds
  unsigned long timeDifference = currentTime - previousTime;
  Serial.print(timeDifference)
  // Calculate the value difference
```

```
55  int valueDifference = sensorValue1 - sensorValue2;
56
57  // Calculate the derivative (change in value / change in time)
58  float derivative = (float)valueDifference / timeDifference;
59
60  // Calculate the time interval (in seconds)
61  float timeInterval = (currentTime - previousTime) / 1000.0;
62
63  // Calculate the area under the curve (approximated by the
        current value * time interval)
64  integral += sensorValue1 * timeInterval;
65
66  // read the analog in value:
67  if (d == 0) {
68    pos1 = target - sensorValue1;
69    pos2 = target - sensorValue2;
70
71    control = K_p * pos1 + K_d * derivative + K_i *integral ;
72    // control = K_p * pos1 + K_d * derivative;
73    if (control <20){
74      control=0;
75    }
76    // map it to the range of the analog out:
77    outputValue1 = map(0, 0, 1024, 0, 255);
78    outputValue2 = map(control, 0, 1024, 0, 255);
79    // change the analog out value:
80    analogWrite(analogOutPin1, outputValue1);
81    analogWrite(analogOutPin2, outputValue2);
82
83    sensorValue2 = sensorValue1;
84    previousTime = currentTime;
85
86    if (sensorValue1 > target) {
87      d = 1;
88    } else {
89      d = 0;
90    }
91    delay(5);
92  }
93  // wait 2 milliseconds before the next loop for the
        analog-to-digital
94  // converter to settle after the last reading:
95
96  // read the analog in value:
97  if (d == 1) {
98    pos1 = sensorValue1 - target;
99    pos2 = sensorValue2 - target;
100
101   control = K_p * pos1 + K_d * derivative  + K_i *integral;
102   // Serial.print(control);
103   //control = K_p * pos1 + K_d * derivative;
```

```
    if (control <20){
      control=0;
    }

    // map it to the range of the analog out:
    outputValue1 = map(control, 0, 1024, 0, 255);
    outputValue2 = map(0, 0, 1024, 0, 255);
    // change the analog out value:
    analogWrite(analogOutPin1, outputValue1);
    analogWrite(analogOutPin2, outputValue2);

    sensorValue2 = sensorValue1;
    previousTime = currentTime;

    if (sensorValue1 < target) {
      d = 0;
    } else {
      d = 1;
    }
    delay(5);

  }
  Serial.println(sensorValue1);
}
```