Chapter 2

DC Motor position control via Arduino Mega

2.1 Problem statement:

To design and implement PID feedback controller using Arduino Mega. To rotate DC motor 180 degrees using this controller that should satisfy rise time of less than 0.5 second, settling time of 1 second and should not overshoot more than 10%.

2.2 Experiment setup

The experiment setup consist of following equipments:

1. **Motor unit**

The DC motor used in the experiment is a permanent magnet DC geared motor.

- **Operating Voltage:** 12V D.C.
- **Full load Current:** 1.2 Amp.
- **Rated Speed:** 50 rpm
- **Torque:** 750gm-cm

2. **Arduino Mega**

The Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins. Out of this 54 pins, 15 can be used as PWM outputs. It has 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection and a power jack with a reset button. Its technical specificatons are as follows:

- **Microcontroller:** ATmega 2560
- **Operating Voltage:** 5V DC.
- **Digital I/O Pins:** 54 (of which 15 can give PWM output)
- **Analog Input Pins:** 16
- **DC Current per I/O Pin:** 20 mA
- **DC Current for 3.3V Pin:** 50 mA
- **Flash Memory:** 256 KB
- **SRAM:** 8 KB
- **Clock Speed:** 16 MHz

3. **L293D IC**

L293D is an H-Bridge IC that can support sufficiently high current. It can provide bidirectional drive currents. It can to drive inductive loads such as relays solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads. It has wide supply-voltage range of 4.5 V to 36 V. It provides 600 mA output current per channel and can get peak current of 1.2 A per channel. A single L293D can drive two motors at a time.

The major applications of L293D is to drive stepper motors, DC motors and latching relays drivers.

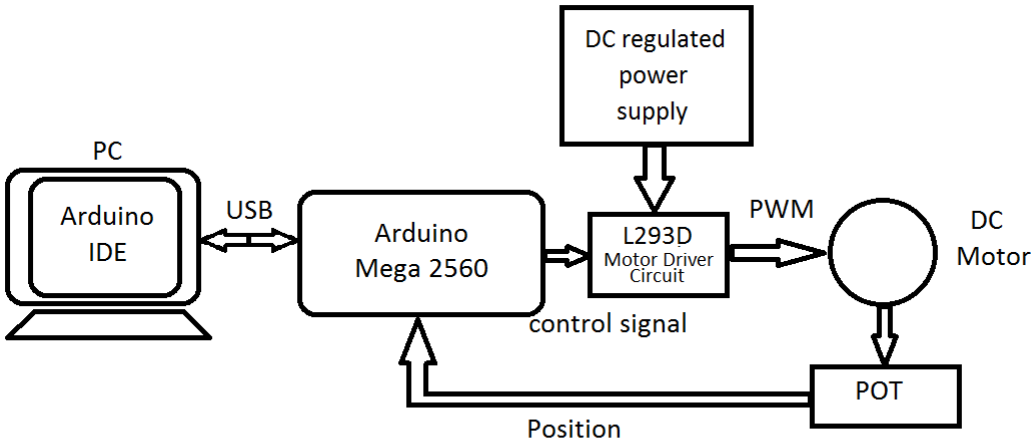

Figure 2.1: Block Diagram of DC position control setup.

2.3 Implemented algorithms with Code snippets:

The source code is written in Arduino IDE. Different algorithms of the main source code are briefly explained here as follows:

2.3.1 Pulse Width Modulation

Pulse WidthModulation is a technique by which power is delivered to the drive by means of switchingthe switch between power source and drive at very high rates. This very high switchingreduces power losses in the drives occurs significantly in other resistive control methods.The average power supplied to the drive is proportional to the duty cycle which is ONperiod over total period of the pulse (Fig. 2.2)

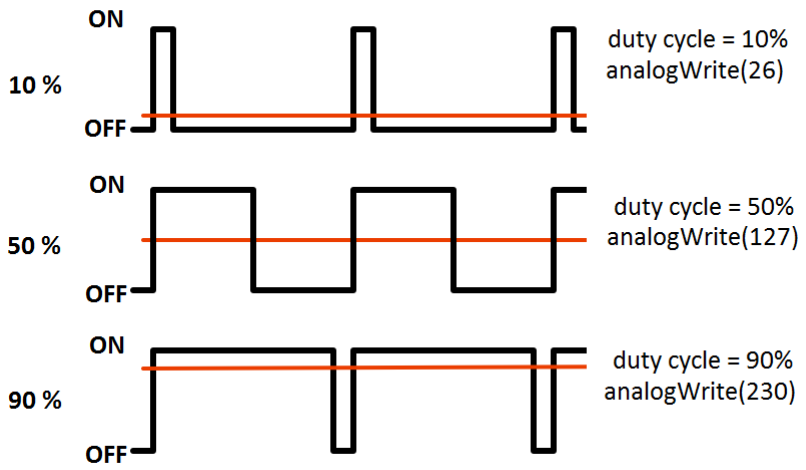

Figure 2.2: Pulse Width Modulation in Arduino IDE.

In arduino IDE, the default PWM frequency is at about 500 Hz. The duty cycle is manipulated using the function analogWrite(constant) where constant vary from 0 to 255, 0 being zero duty cycle and 255 being 100% duty cycle.

The code snippet below shows how pwm values is taken into account to drive the motor. The motor positive and negative terminals are connected to PIN 11 and PIN 9 respectively.The PID value `Drive` comes after PID calculations are done. The correction term 60 in `Drive` is made for motor to overcome the static friction and start. The `if()` condition ensures that corrected pwm values for drive does not overshoot the limit i.e. 255. This value is then fed into `analogWrite()` which sets voltage according to it on 11th PIN. The absolute value of `Drive` along with its two `if()` statements makes the DC motor to

run in one direction.

```
if (Drive < 0){
        Drive = Drive -60;
       if (abs(Drive)>255) {
         Drive=255;
      }
         analogWrite(11, abs(Drive));
         digitalWrite (9,LOW);
      }
      if (Drive > 0){
        Drive = Drive +60;
        if (abs(Drive)>255) {
         Drive=255;
      }
         analogWrite(9, Drive);
         digitalWrite (11,LOW);
      }
```

2.3.2 Proportional Integral Derivative control

PID is a generic control loop feedback mechanism which continuously determines error values as a difference between present state and desired state and it attempts to minimize the error by adjusting a measured process variable. It involves three separate correction methods that are proportional to error, accumulated error and real time difference of the error. If $u(t)$ is control variable and $e(t)$ is the error value then

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

where K_p, K_i, K_d are all non-negative, denote the coefficients for the proportional, integral, and derivative terms.

The objective was to ensure that DC motor rotates exactly 180 degrees from its starting position. For this we required to calculate the starting position of the rotating disc. This was done by writing separate function `StartingPosCal()`. as shown:

```
 void StartingPosCal()
{
```

```
  while(i!=1)
  {
    StartingPos=Angle;
    i++;
  }
}
```

After calculating `StartingPosCal()`, we would make our set point to StartingPos + 180. Integral windup concept is also taken into account. The following code snippet shows implementation of PID algorithm in our DC position control with Arduino Mega:

```
int IntThresh=0;
    float P, I, D;
    int Integral=0;
    Actual=Angle;
    SetPt=StartingPos+180;
    if(SetPt>360)
    {
      SetPt=SetPt-360;
    }
    Error = SetPt - Actual;
    if (abs(Error) < IntThresh){  // prevent integral 'windup'
       Integral = Integral + Error;  //accumulate the error integral
    }
    else {
       Integral=0;          // zero it if out of bounds
    }
    P = Error*kP;           // calc proportional term
    I = Integral*kI;        // integral term
    D = (Last-Actual)*kD;     // derivative term
    Drive = P + I + D;        // Total drive = P+I+D
    Drive = Drive*ScaleFactor;
    Last=Actual;
```

Initially all control constants should be set to zero. We start tuning our controller by increasing kP first. With increase in kP the position overshoot increases and starts oscillating around setpoint. It is tweaked till the rotating disc reaches the setpoint and is at brink of oscillating around the setpoint.

After kP is tuned, kI is increased until steady state error is eliminated in sufficient amount of time. The integral term accelerates the rotating disc to reach setpoint, and can cause increase in overshoot due to responding to accumulated errors. Therefore Integral windup concept is taken into account. After kI is tuned such that steady state error in the

disc is eliminated, then kD is increased and tuned such that the objective is reached. Too much increase in kD will lead the disc to respond very fast and will cause overshooting and oscillations again.

2.3.3 Calibration

In the experiment setup, the POT values from DC motor ranged from 0 to 1023. This value from POT was recieved from Analog PIN A5 of the Arduino board. In order to make a complete rotation equivalent to 360 degrees, a variable `Angle` was made proportional to the `potValue`. But due to the presence of a dead band, the disc was not rotating exactly 360 degrees. The equivalent 360 degrees of mechanical rotation was calibrated from 339 degrees. Following code snippet shows the calibration of `Angle`.

```
potValue = analogRead(A5);
   Angle = potValue * (339.0 / 1023.0);
   StartingPosCal();
```

2.4 Source code

```
//int ledPin = 9;    // LED connected to digital pin 9
int pwm=127;
int Actual,Last, Error=0, SetPt;
float kP=3;
float kI=0;
float kD=0.1;
float Drive=0;
float ScaleFactor=1;
int potValue;
float Angle;
float StartingPos,i=0;
void setup() {
  Serial.begin(9600);
  pinMode(9, OUTPUT);
  pinMode(11, OUTPUT);
  Serial.println("CLEARDATA");
  Serial.println("LABEL,Time,Milisec,Setpoint,Angle");
}
```

```
void printData();
void pid();
void StartingPosCal();

void loop() {
    potValue = analogRead(A5);
    Angle = potValue * (339.0 / 1023.0);
    StartingPosCal();
    pid();
    printData();
    delay(1);
  }

void printData()
{
    Serial.print("DATA,TIME,");
    Serial.print(millis());Serial.print(",");
   // Serial.print("Stpt= ");
    Serial.print(SetPt);Serial.print(",");
   // Serial.print(", Angle= ");
    Serial.print(Angle);Serial.print(",");
  //  Serial.print(", E= ");
   // Serial.print(Error);
    Serial.print(", Dr= ");
    Serial.print(Drive);
    Serial.print(", StPos= ");
    Serial.println(StartingPos);
}

void StartingPosCal()
{
  while(i!=1)
  {
    StartingPos=Angle;
    i++;
  }
}

void pid()
{
```

```
int IntThresh=0;
float P, I, D;
int Integral=0;
Actual=Angle;
SetPt=StartingPos+180;
if(SetPt>360)
{
  SetPt=SetPt-360;
}
Error = SetPt - Actual;
if (abs(Error) < IntThresh){ // prevent integral 'windup'
   Integral = Integral + Error;// accumulate the error integral
}
else {
   Integral=0; // zero it if out of bounds
}
P = Error*kP;  // calc proportional term
I = Integral*kI;  // integral term
D = (Last-Actual)*kD;  // derivative term
Drive = P + I + D;   // Total drive = P+I+D
Drive = Drive*ScaleFactor;

if (Drive < 0){
  Drive = Drive -60;
 if (abs(Drive)>255) {
   Drive=255;
}
   analogWrite(11, abs(Drive));
   digitalWrite (9,LOW);
}
if (Drive > 0){
  Drive = Drive +60;
  if (abs(Drive)>255) {
   Drive=255;
}
   analogWrite(9, Drive);
   digitalWrite (11,LOW);
}

if (abs(Drive)>255) {
```

```
            Drive=255;
        }
        if(Drive ==0){
          digitalWrite(9, LOW);
          digitalWrite (11,LOW);
        }
        Last = Actual;  // save current value for next time
}
```

2.5 Results

The following resuts were obtained succesfully:

- **Rise Time:** 420 ms
- **Overshoot:** 3.33
- **Settling Time:** 750 ms

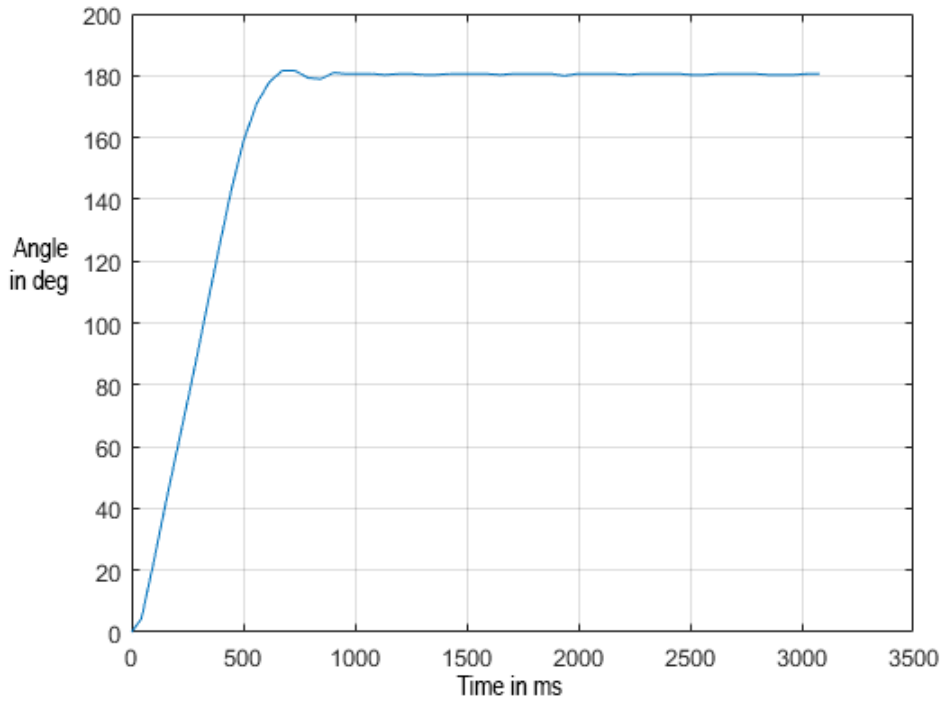

Figure 2.3: Position with respect to time

2.6 Challenges

The experiment took two weeks to complete. The challenges we faced during performing the experiment are discussed below.

- There was a dead band in potentiometer which caused trouble in calibrating a complete rotation of the disk in degrees. In order to make a complete rotation equivalent to 360 degrees, a variable Angle was made proportional to the potValue. This problem was solved by making the equivalent 360 degrees of mechanical rotation calibrated to 339 degrees.