Sensors and Motor Control Lab

Individual lab report – 01 || October 16, 2015

Shivam Gautam

Team I

Dorothy Kirlew

Pranav Maheshwari

Richa Varma

Mohak Bhardwaj

1. Individual Progress

I undertook the following tasks:

1. Interfacing the force sensor with the Arduino.
2. DC motor control via-
    a. User defined Angular Position
    b. User defined Speed
3. PID tuning for position and speed control

1. Interfacing the force sensor with the Arduino

The force sensor, as the name suggests, measures the amount of force applied on a circular patch of the sensor. The sensor comes in various form factors with sizes ranging from a 5mm diameter circle (20mm length) to a 38mm length square. The sensor provided had a 5mm diameter for the sensing patch and was 38mm in length.

The principle of operation is that the resistance of the force sensor changes with applied force, hence its name- the Force Sensing Resistor (FSR). The resistance of the FSR decreases with increasing force.

The circuit for the FSR is depicted in figure 1. The soldered circuit is depicted in figure 2.

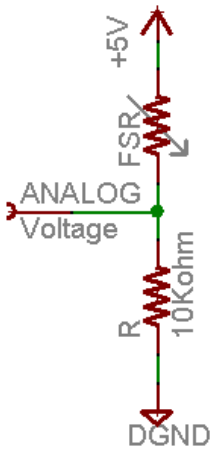

Figure 1- Circuit Diagram

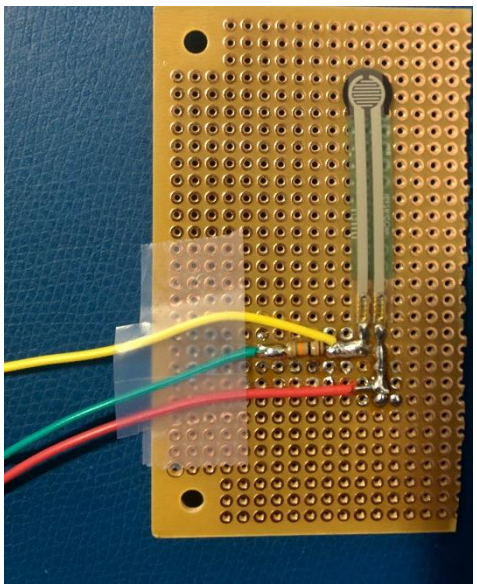

Figure 2- Soldered Circuit

The voltage and the resistance of the sensor can be estimate as-

$$Va = Vcc - I*Rfsr$$

$$Va = Vcc - (Vcc* Rfsr)/ (Rfsr+ R)$$

$$Va = Vcc *R/ (Rfsr+R)$$

$$Ra = ((Vcc-Va)*R)/Va$$

Where Va= Voltage across FSR

Vcc= Suply Voltage (+5V)

Rfsr= Resistance of force sensor

R= Resistance in series with Rfsr

The value of R was selected based on the voltage vs force graph depicted in Figure 3. The graph depicts that with the 10k resistor, a linear region with maximum voltage range is obtained. For other values, the linear region exists but the range is not greater than that with the 10k resistor.

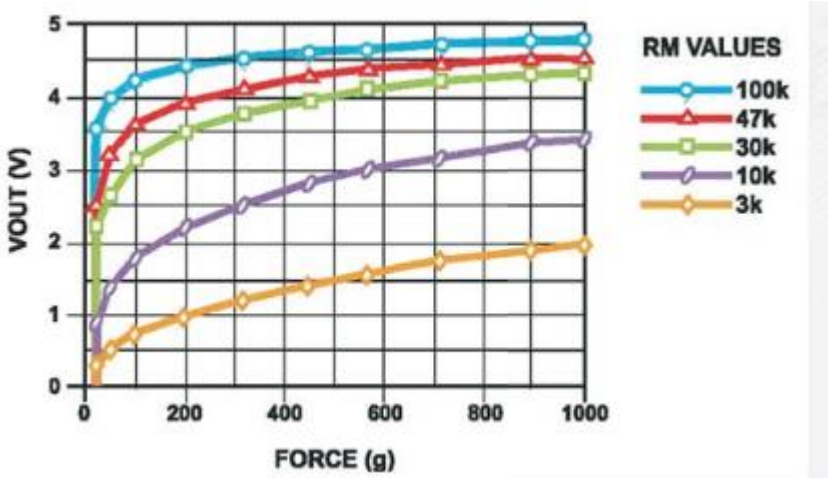

Figure 3

The datasheet for the force sensor depicts the resistance vs force graph (Figure 4)

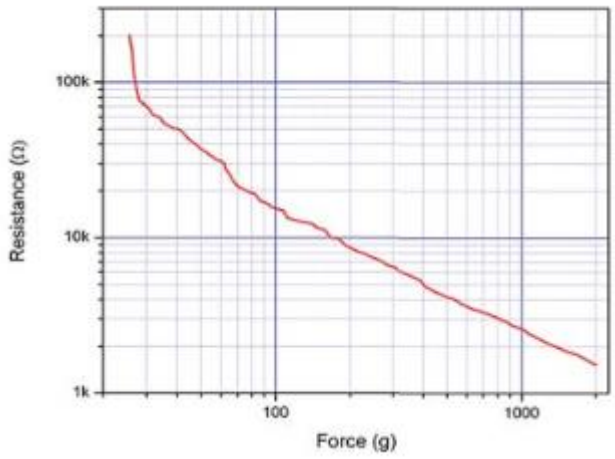

Figure 4

Using this graph, I came up with the equation to convert resistance values to actual force measurements. From the graph it is evident that if the resistance is greater than 80k, the readings of the force are considered to be zero. For all other cases the following formula is applied.

$$\text{Force} = 1 + \left((-R_{fsr} + 80000) / 77500 \right)$$

2. DC Motor Control

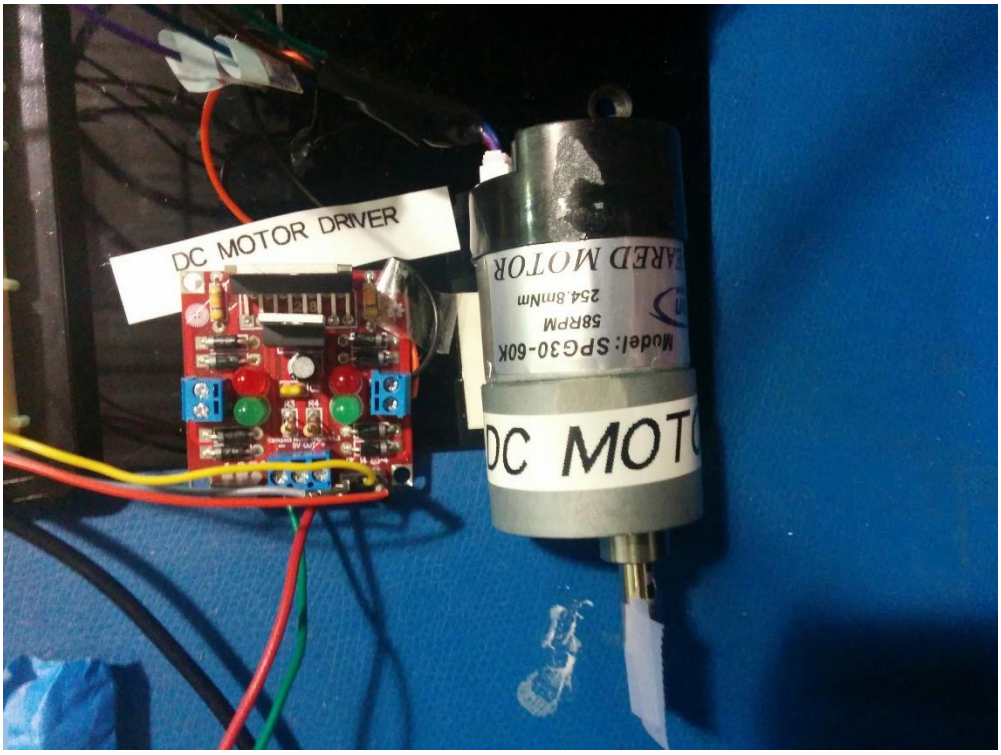

Figure 5- DC Motor with motor driver

The DC motor was controlled by:

a. Angular Position
b. Speed

a. Angular Position

This objective entails the rotation of the motor at a user specified number of degrees i.e. instead of a continuous rotation, the DC motor would rotate for a fixed angle.

The first step in controlling the angular position is finding out what the current angular position is. This was done with the help of the built in encoder in the Cytron SPG-30 60K motor. The encoder, for this 1:60 geared motor, provided 180 counts per main shaft revolution.

The angular position control is implemented after taking the current reading of the encoder. Once the current angular position is known, the desired angle is then added to the current reading to generate the value of a set-point for the PID controller. These values of the set-point and the current orientation are fed into a PID calculator which computes the desired PWM output for the motor.

b. Speed Control

The reading from the encoder denotes the current value of the encoder position , which can be used to estimate the angular speed. After measuring the amount of rotation after a fixed interval of time, the simple speed-time formula is applied to calculate the current velocity.

The speed obtained after this measurement was between 0 and 0.82 which had to be mapped to 0-255. This was done with the help of a linear equation-

```
spMap = (Setpoint_v * 255) / 0.84
```

The desired speed is used to decide on a set point for the PID calculation and the output of the PID calculator is fed as the PWM output for the motor.

3. PID Tuning

The PID tuning was performed by observing the values of the set point and the current state vs time of the DC motor. The tuning was done via textual serial outputs from the Arduino.

The methodology used was-

1. Start with Kp=1, Ki=0, Kd=0
2. Increase Kp in increments of 0.2 until the response shows overshoot. Reduce the Kp at the point where motor starts to overshoot.
3. Increase Ki in increments of 0.05 until the steady state error is minimal.
4. In case of oscillations, increase Kd in increments of 0.02

4. Challenges

Among the several challenges, the ones that were worth taking a note of are-

1. The force sensor needed to be integrated on a strong base and could not be simply added to the breadboard. Improper integration would have led to incorrect readings from the force sensor. To counter this, I developed a custom board for the force sensor and by soldering it to a veroboard.
2. The integration of the PID code with the overall code proved to me a challenge on many levels.
    a. The setpoint assignment required the setpoint to be calculated after either every serial input, or after the change of a sensor input reading. Figuring out the best location for this call determined if the setpoint would be updated appropriately and it took some time to figure this out.

    b. The angular position control needed to turn the motor at an angle by adding it to the current encoder reading. However, if the current location and the setpoint changed after every sensor update, the motor would assume that its initial position too had changed. This would mean that the motor would continuously rotate as the setpoint would always lead the current reading by a certain angle. To counter this and any noisy sensor fluctuations, the setpoint was programmed to change only if the new setpoint was greater than 30 units (on a scale of 0- 1023).

c. Relying on the serial output of the current position and the error to tune PID was a challenge as it did not provide any intuition with respect to the rise time, the steady state error and the oscillations in the system. The serial outputs could, however, provide a rough estimate of the error of the system and what parameters to tweak.

d. The PID outputs for angular control and speed control had to be handled differently. As expected, not only were their gains different, the way the motors received this command was also different. For the angular control, the output of the PID loop had to be fed as PWM to the motor whereas for the speed control, the output of the PID had to be added to the setpoint and then fed as a PWM output to the motor.

3. Another bug in the program was the usage of the inbuilt function 'map' which could map only integer values to other integers. This function proved useless for mapping the speed and the force as both of them were floating points. A quick fix was writing my own equation for the mapping.

5. Teamwork

The full system integration was the result of the cohesiveness of the team. Through the assignment, the team worked together in the lab which proved to be more effective than working separately. The troubleshooting time and miscommunication of information was greatly reduced as each member had a high level idea of what the others were working on and what challenges they were facing.

The major areas which involved required me working with another team member were- code integration, electronics integration and troubleshooting. I worked with Pranav on the integration of the code and troubleshooting problems that arose with it. I had to work with Richa on the integration and helped in creating the base platform for integration. Troubleshooting COM port related problems with Dorothy and Mohak was also a good experience.

6. Plans

Based on our Conceptual Design Review report, the team would be working on a set of tasks for the next progress review on the 22nd of October. I would be primarily working on finalizing the communication system and its procurement. As outlined in the report, the two options being considered are wireless P2P routers and XBee S2B radios.

I have also been working on finalizing a mobile platform that we would seek from the sponsor and one that we could buy off the shelf.

7. Code

Force Sensor

```
void setup()
{
 Serial.begin(9600);
}

void loop()
{
 int frc = analogRead(A2);
 Serial.print("Analog Value is: ");
 Serial.println(frc);
 float voltage = frc * (5.0 / 1023.0);
 Serial.print("Voltage is: ");
 Serial.println(voltage);
 float frcRes = ((5 - voltage)*10000)/voltage; //frc resistance= ((Vcc-Va)*R)/Va

 Serial.print("Force Sensor Resistance (kilo ohms) is: ");
 Serial.println(frcRes/1000);

 float force=0;

 if (frcRes>80000)
 {
  force=0;
 }
 else
 {
  force= 1 + ((-frcRes+80000)/77500);
 }
 Serial.print("FSR force(N) is: ");
 Serial.println(force*10);
 delay(500);

}
```

Angular Position Control

```
#include <PID_v1.h>
#define m1 5//motor pin 1 (Grey Wire from Motor Board)
#define m2 6 //motor pin 2 (Yellow Wire)
#include <Encoder.h>
Encoder myEnc(2, 3);
int desiredAngle = 0, currentPosition = 0, desiredPosition = 0, error = 0, m_speed = 0, m_direction = 0;
double Setpoint, Input, Output;
double Kp = 1.3, Ki = 0.2, Kd = 0.03;
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);
void velocityControl( int m_speed, int m_direction);

void setup() {
 pinMode(m1, OUTPUT);
 pinMode(m2, OUTPUT);
 Serial.begin(9600);
 Serial.println("Testing Angular Position control using PID control");
 myPID.SetMode(AUTOMATIC);
 currentPosition = myEnc.read();
 myPID.SetOutputLimits(-255, 255);



}

void loop() {

 if (Serial.available() > 0)    // Used to get a set point,
 {                      //This piece of code not needed in the final integration
  desiredAngle = Serial.read();
  desiredAngle = 180;  //The resolution of the motor is such that setpoint=desiredAngle/2s

  currentPosition = myEnc.read();
  Setpoint = currentPosition + desiredAngle;
  Serial.print ("Current Position is : ");
```

```
   Serial.println(currentPosition);
   Serial.print("Set Point was : ");
   Serial.println(Setpoint);
  }

  Input = myEnc.read();
  myPID.Compute();
  Serial.print("Output was : ");
   Serial.println(Output);
  m_speed = abs(Output);
  m_direction = Output / abs(Output);
  velocityControl( m_speed,  m_direction);

}
void velocityControl( int m_speed, int m_direction)
{
  if (m_direction == -1)
  {
   analogWrite(m1, m_speed);
   digitalWrite(m2, LOW);
  }
  else if (m_direction == +1)
  {
   analogWrite(m2, m_speed);
   digitalWrite(m1, LOW);
  }


}
```

Speed Control

```
#include <PID_v1.h>

#define m1 6 //motor pin 1 (Grey Wire from Motor Board)
#define m2 5 //motor pin 2 (Yellow Wire)
```

```
#include <Encoder.h>
Encoder myEnc(2, 3);

double Setpoint_v = 0, Input_v = 0, Output_v = 0;
double Kpv = 2, Kiv = 0.5, Kdv = 0.03;
int m_speed = 0, m_direction = 1;
double desiredSpeed = 0, currentSpeed = 0;
void velocityControl( int m_speed, int m_direction);



PID myPID_v(&Input_v, &Output_v, &Setpoint_v, Kpv, Kiv, Kdv, DIRECT);

void setup() {
  pinMode(m1, OUTPUT);
  pinMode(m2, OUTPUT);
  Serial.begin(9600);
  Serial.println("Testing Speed control using PID ");
  myPID_v.SetMode(AUTOMATIC);

}

void loop() {

  float initialCount = myEnc.read(), finalCount = 0;
  delay(50);
  finalCount = myEnc.read();
  currentSpeed = (finalCount - initialCount) / 50;

  Input_v = currentSpeed;

  if (Serial.available() > 0)
  {
    int x = Serial.read();
    desiredSpeed = desiredSpeed + 0.2;
    Setpoint_v = desiredSpeed;
```

```
   Serial.print ("Current Speed is : ");
   Serial.println(currentSpeed);
   Serial.print("Set Point was : ");
   Serial.println(Setpoint_v);
  }

  myPID_v.Compute();



  //m_direction = Output_v >0 ? +1 : -1;
  m_direction = 1;


  float spMap = (Setpoint_v * 255) / 0.84 ;
  m_speed = Output_v + spMap;
  velocityControl( m_speed,  m_direction);



  Serial.print("Output was : ");
  Serial.println(Output_v);
  Serial.print ("Current Speed is : ");
  Serial.println(currentSpeed);
  Serial.print("Set Point was : ");
  Serial.println(Setpoint_v);
  delay(200);

}

void velocityControl( int m_speed, int m_direction)
{
  if (m_direction == -1)
  {
    analogWrite(m1, m_speed);
    digitalWrite(m2, LOW);
  }
```

```
  else if (m_direction == +1)
  {
    analogWrite(m2, m_speed);
    digitalWrite(m1, LOW);
  }
}
```