Alexander Kyu
BME 385

BME 385 Project Report

I. Task I

Methods

In Task I, we were asked to create a wheatstone bridge configuration that resulted in a positive differential between the positive and negative outputs. The negative side of the wheatstone bridge used two identical resistors (10k Ohms). The positive side of the wheatstone bridge consisted of a variable resistor (the FSR) and a resistor that was slightly smaller than the measured resistance of the FSR with a 500g mass on it.

These outputs were then fed into the AD623 Instrumentation Amplifier (IA) to amplify the signal and to provide a buffer to the system. The output of the IA was then measured by the arduino through an Analog Input pin. The input for four different masses--500g, 200g, 100g, and 50g--were recorded for three trials and a best fit curve was determined to relate voltage to mass.

Results

The FSR resistance with the 500g mass is measured to be approximately 3.3-3.4k Ohms. There were minor fluctuations in the measurement. With this measurement, a 3.3k (actual measurement 3.23k) ohms was chosen to complete the wheatstone bridge. This was chosen to be close to this value as it results in a voltage divider on the positive side close to, but slightly larger than 3.3/2V so that the difference between the positive and negative outputs are always positive. The wheatstone bridge was powered by a 3.3V power supply from the arduino. This means that the output of the wheatstone bridge varied from 1.67V to 0V differential output.

As stated in the methods section above, both positive and negative outputs were inputs to the IA. The IA was configured to amplify the signal with a gain of 2, resulting in an output that ranged from 0-3.3V before being input into the Arduino. This gain was achieved by using a resistor of 100k Ohms between -Rg and +Rg based on the following equation:

$$V_O = \left(1 + \frac{100\ \text{k}\Omega}{R_G}\right) V_C$$

Equation 1: Equation used to determine the gain of the Instrumentation Amplifier

The final circuit diagram for this section is shown below:

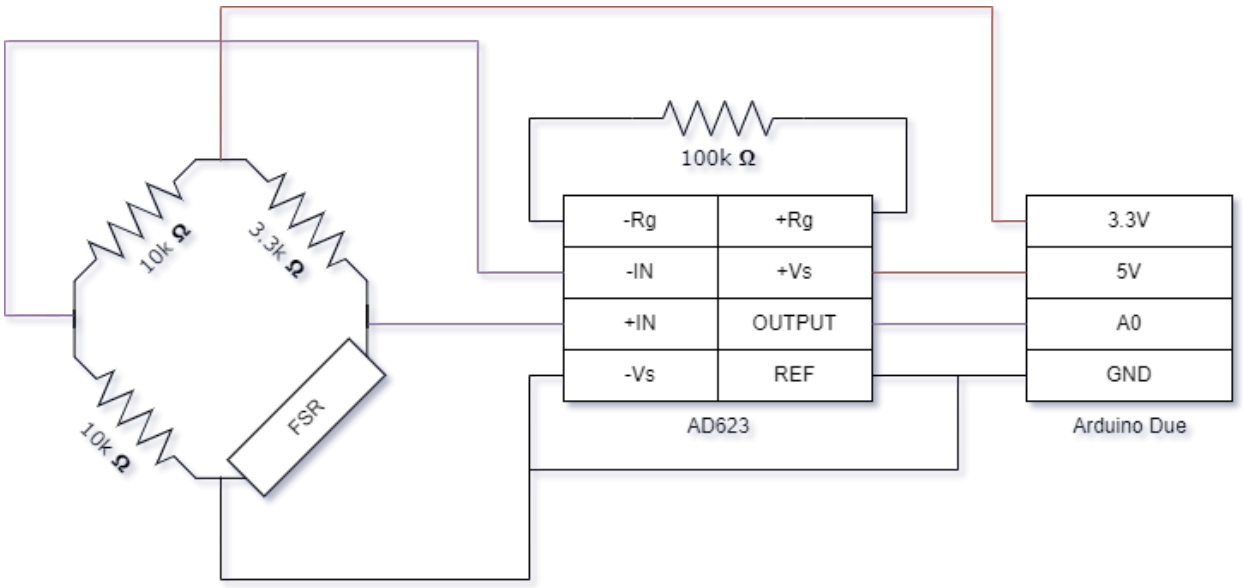


Figure 1: Circuit diagram for Task 1

The Force to Voltage table for all trials is also shown below:

Table 1: Voltage vs Mass for 3 Trials and the Average Voltage for each mass

	Voltage (V)			
Mass (g)	Trial 1	Trial 2	Trial 3	Average
0	3.3	3.3	3.3	3.3
50	1.995	1.765	1.982	1.914
100	1.471	1.636	1.418	1.508333
200	0.983	0.749	0.886	0.872667
500	0.274	0.322	0.242	0.279333

Three separate curves were created which are shown below. Their R^2 and RMSE values are also shown below:

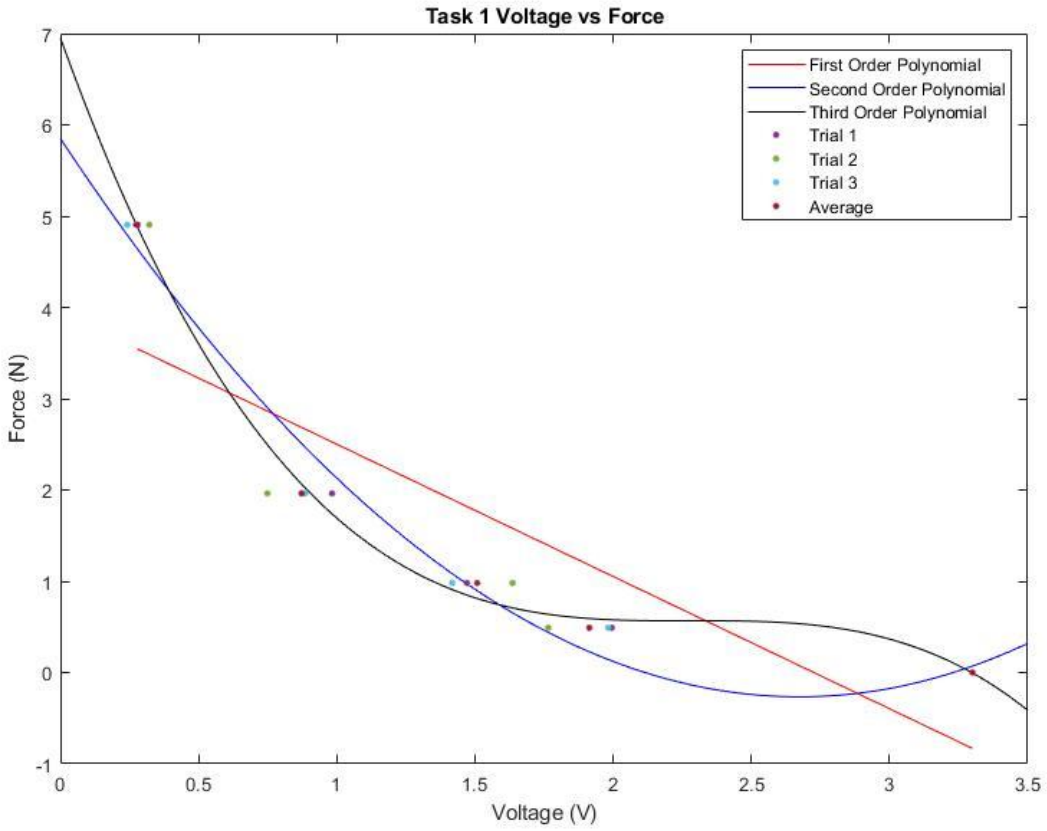


Figure 2: Plot for Task 1, Voltage versus Force for all 3 Trials, the Average, and 3 Different Fitted Curves

First Order Polynomial: R^2 = 0.7272, RMSE = 1.1760

Force = -1.449*Voltage + 3.949

Second Order Polynomial: R^2 = 0.9703, RMSE = 0.4750

Force = 0.8559*Voltage^2 - 4.576*Voltage + 5.849

Third Order Polynomial: R^2 = 0.9967, RMSE = 0.2241

Force = -0.5404*Voltage^3 + 3.694*Voltage^2 - 8.415*Voltage + 6.952

Discussion

Overall, Task 1 was completed successfully. However, there were some inconsistencies as the 500g mass should have resulted in a value very close to 0V with the resistor value being so close to the measured FSR resistance. This was not the case though with a value around 0.3V for the 500g mass. This is probably due to the small difference in the resistor used and the FSR resistance and the variability within the FSR itself and with force distribution on the FSR.

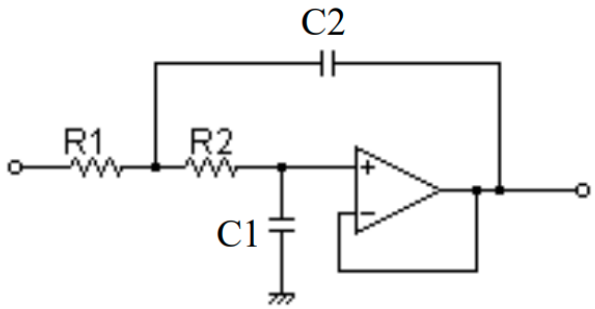
Furthermore, this variability also is the reason for the variability in the trials with the different masses.

For the curved fits, the third-order polynomial curve fit the best with a very high R^2 and very low RMSE.

II. Task II

Methods

In task 2, we were asked to develop a Sallen-Key Butterworth 4th-order lowpass filter with a cutoff frequency of 20 Hz. This was done by using two stages (each 2nd order) with the Sallen-Key Lowpass Filter topology and the related equations:



Choose different R1, R2, C1, and C2 values for different filters

- Butterworth, Chebyshev, Bessel

$$H(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2}$$

$$H(s) = \frac{\frac{1}{R_1 R_2 C_1 C_2}}{s^2 + \frac{sC_1(R_1 + R_2)}{R_1 R_2 C_1 C_2} + \frac{1}{R_1 R_2 C_1 C_2}}$$

$$\omega_n = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}} \qquad f_n = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}}$$

$$2\xi = \frac{C_1(R_1 + R_2)}{\sqrt{R_1 R_2 C_1 C_2}} \qquad Q = \frac{1}{2\xi} = \frac{\sqrt{R_1 R_2 C_1 C_2}}{C_1(R_1 + R_2)}$$

Figure 3: Lowpass Sallen-Key Topology for a single stage (2nd-Order per stage)

To achieve the Butterworth Filter characteristics, the following values were taken from a design table for each stage:

Stage 1: FSF = 1.0 Q = 0.5412
Stage 2: FSF = 1.0 Q = 1.3065

The following math was done to determine the R1, R2, C1, and C2 values for each stage:

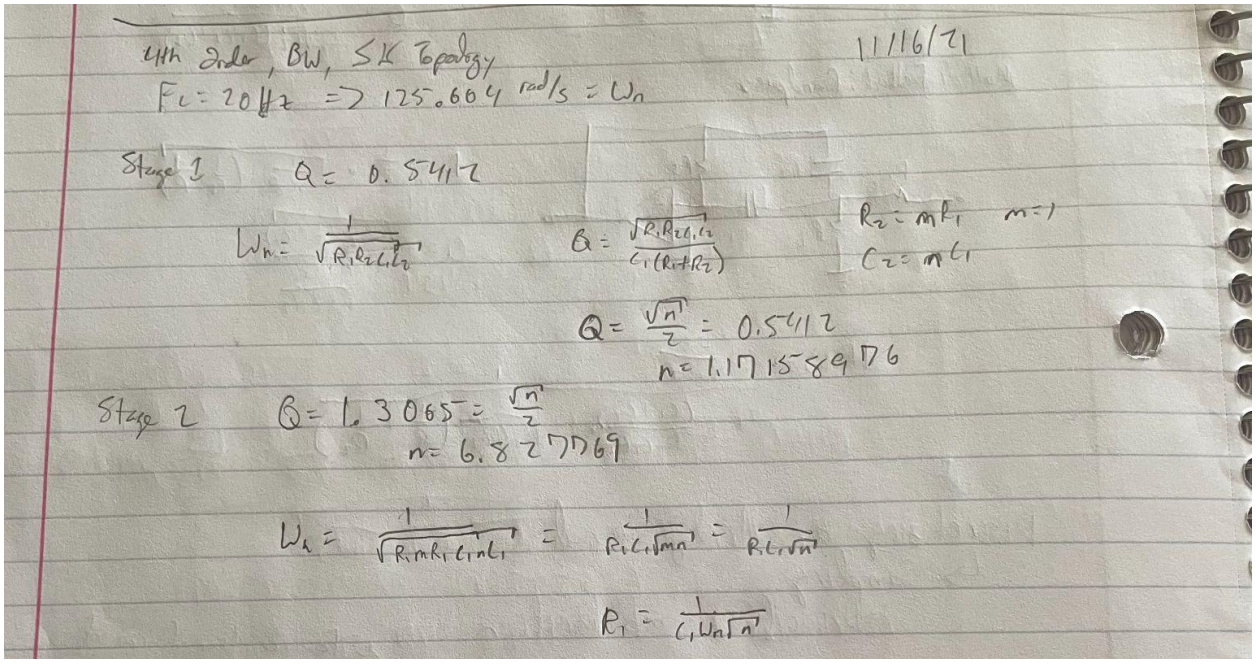


Figure 4: Math done to determine Resistor and Capacitor Values for each stage.

This was then inputted into MATLAB to streamline the process of determining reasonable capacitor and resistor values:

wn = 20 * 2 * pi;
n1 = 1.17158976 for stage 1 or 6.827769 for stage 2;
c1 = 1e-6 for stage 1 or .1e-6 for stage 2;
c2 = n1*c1;
r1 = 1/(c1*wn*sqrt(n1));
r2 = r1;

This resulted in the following component values, transfer function, and the frequency response:

Stage 1:
- C1 = 1uF
- C2 = 1.17uF
- R1 = R2 = 7.35k Ohms

Stage 2:
- C1 = 0.1uF
- C2 = 0.68uF
- R1 = R2 = 30.454k Ohms

$$H(s) = \frac{1.039e09}{s^4 + 480.9\,s^3 + 1.393e05\,s^2 + 1.921e07\,s + 1.039e09}$$

Equation 2: Transfer Function for Designed Ideal Filter

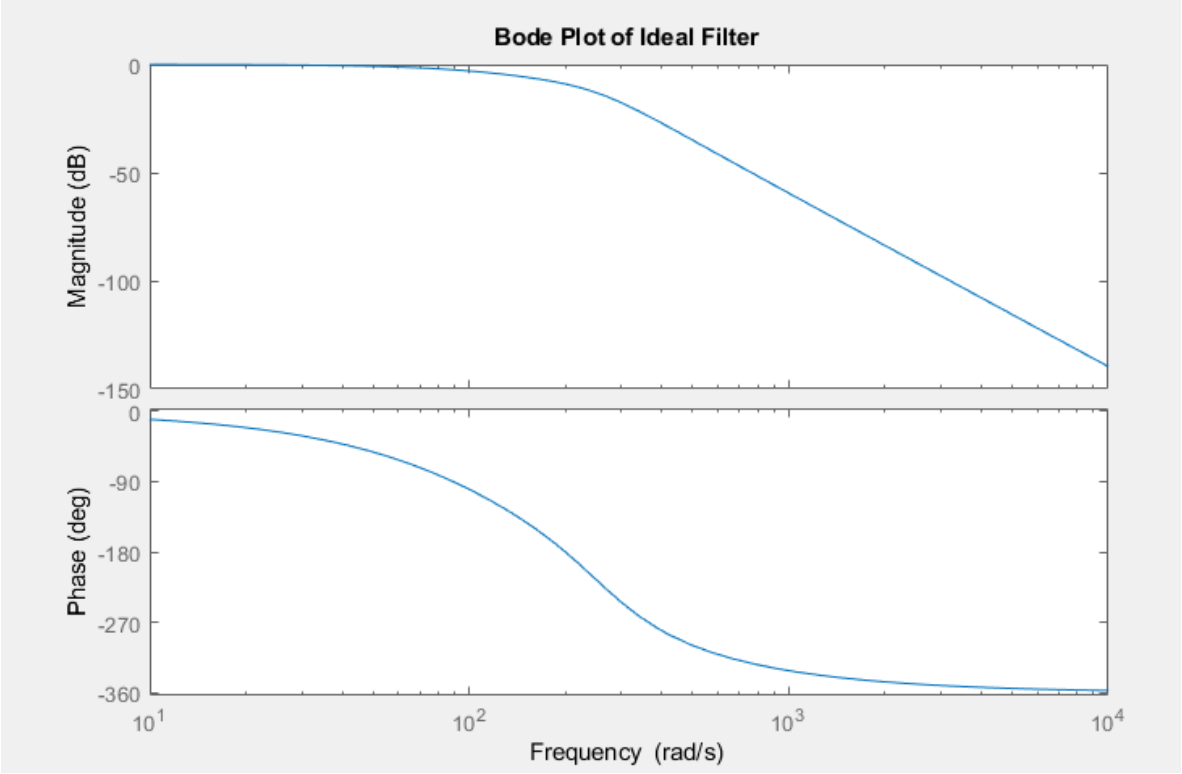


Figure 5: Frequency response of Designed Ideal Filter

However, because these are ideal values, real resistor and capacitor values were chosen to achieve similar performance as shown in **Figure 6** and **Figure 7:**

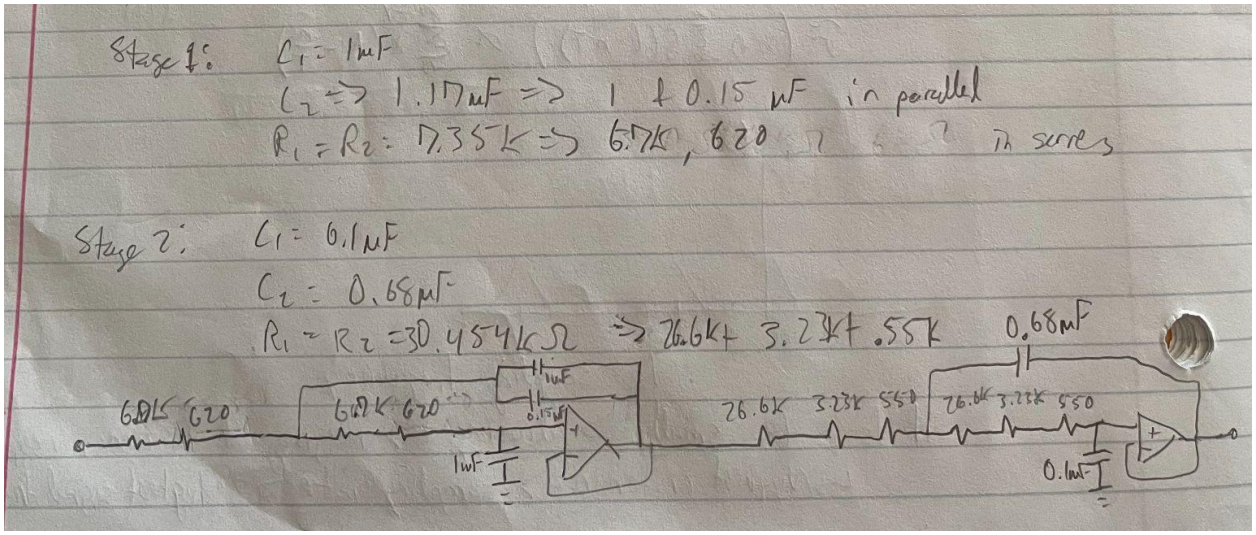


Figure 6: The finalized topology for the designed filter.

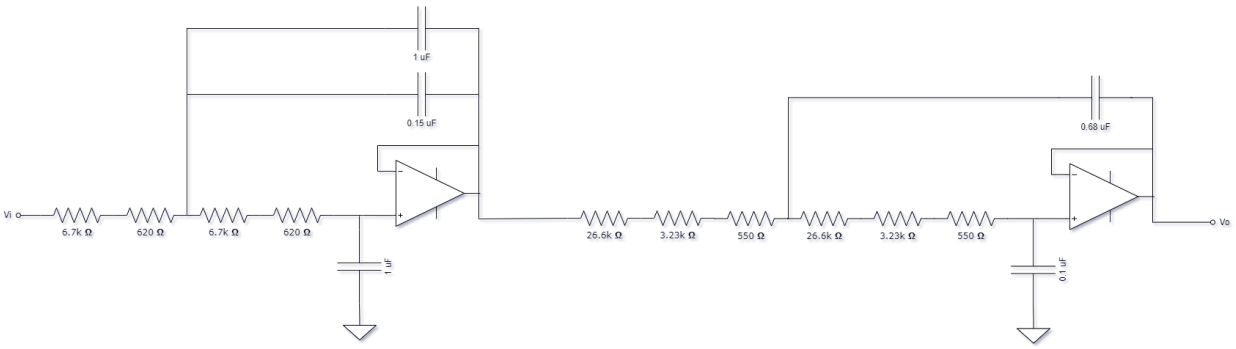


Figure 7: Finalized Topology for the Filter Design of a Sallen-Key 4th Order Lowpass Butterworth Filter

This resulted in the following transfer function:

$$H(s) = \frac{1.073e09}{s^4 + 486.9\,s^3 + 1.416e05\,s^2 + 1.976e07\,s + 1.073e09}$$

Equation 3: Transfer Function for Non-ideal Designed Filter

To test the filter, several sine waves were produced by the Arduino Due's PWM Pin and the outputs were analyzed to check that the output was appropriate (**Appendix A**). Each sine wave had an amplitude of 1V and was centered at 1.65V (3.3V/2). The three frequencies this was done at were 2 Hz, 20 Hz, and 200 Hz or 12.566 rad/s, 125.664 rad/s, and 1256.637 rad/s, respectively.

Using the Frequency response, the following change in amplitude from the original sine waves are expected:

2 Hz:	0.9941
20 Hz:	0.6062
200 Hz:	0.00043651

This was done using the bode function within MATLAB which computes the magnitude at the given frequencies (converted to rad/s first).

Results

The outputs for each of the frequencies are shown below in **Figure 8**, **Figure 9**, and **Figure 10** with the following min and max voltage values for 2 Hz, 20 Hz, and 200 Hz, respectively**:**

2 Hz:	Min = 0.65V	Max = 2.66V	Amp: 1.005V
20 Hz:	Min = 0.85V	Max = 2.46V	Amp: 0.805V
200 Hz:	Min = 1.5V	Max = 1.79V	Amp: 0.145V

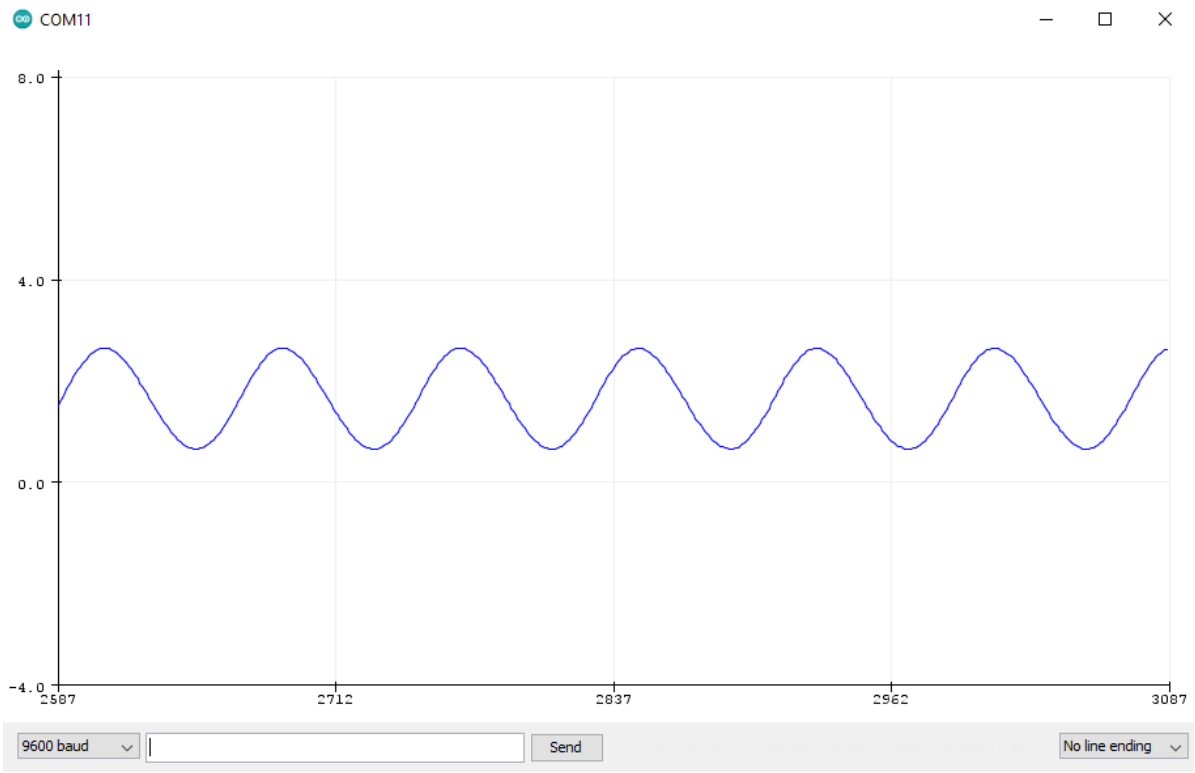


Figure 8: Filter Response at 2 Hz

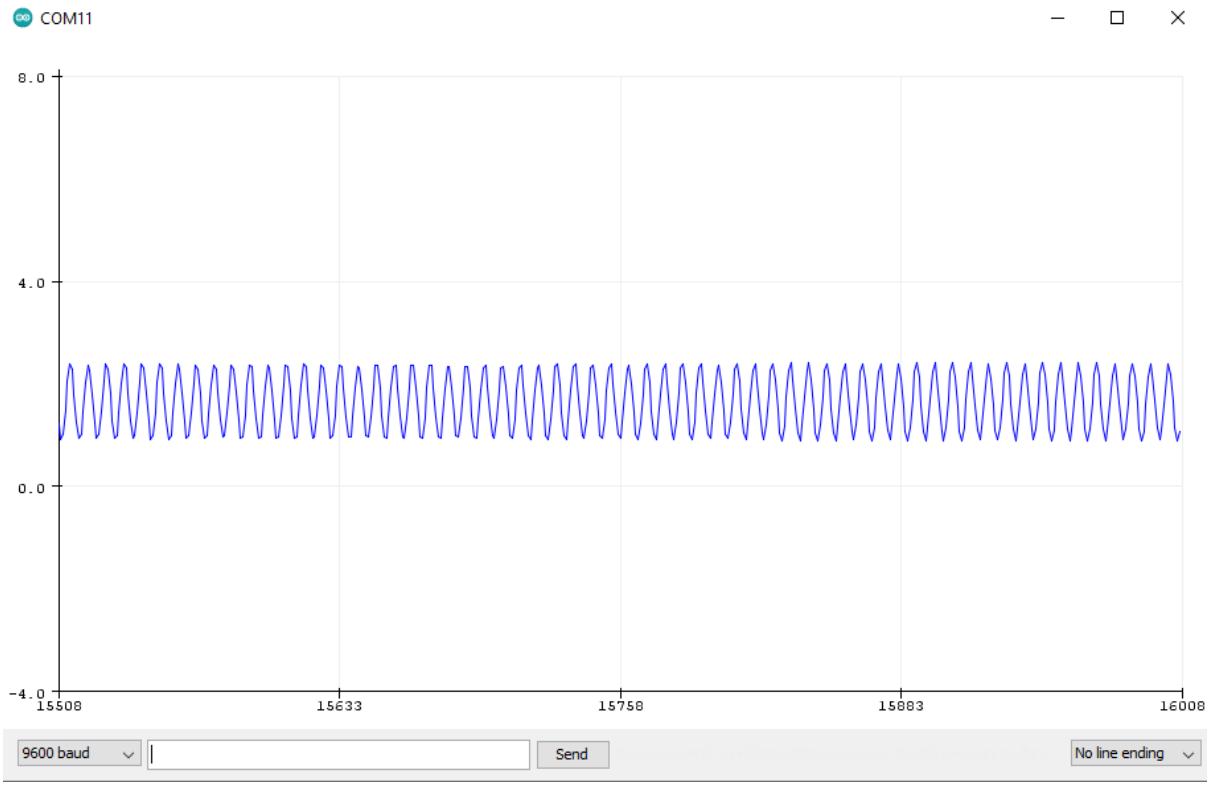


Figure 9: Filter Response at 20 Hz

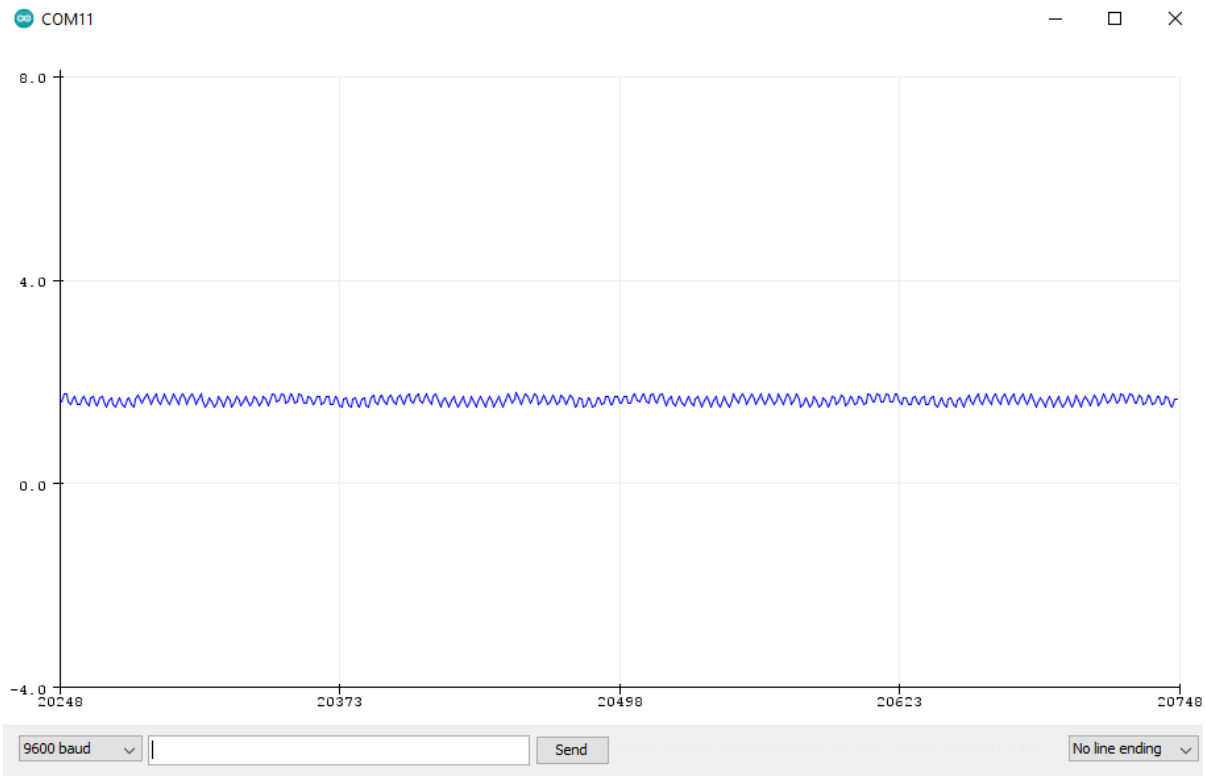


Figure 10: Filter Response at 200 Hz

Discussion

While the filter did attenuate the higher frequency inputs, the attenuation was not as expected:

Table 2: Expected vs Actual Signal Attenuation through Filter

Frequency (Hz)	Expected (V)	Actual (V)
2	0.9941	1.005
20	0.6062	0.805
200	0.00043651	0.145

This is likely due to a few reasons: first, the component (resistors and capacitor) values used in the expected calculation are slightly different from the actual component values; second, the output of the arduino is not a true analog signal. It is instead a PWM (Pulse-Width Modulation) signal which is just a proportion of time for which the signal is on and off. This may change the behavior of the actual filter; third, there is an error within the calculations or constants used in calculating the desired component values; and fourth, the op-amp (LM 358P) we used has some characteristics we did not account for or was damaged.

Despite the differences, the filter appropriately diminished higher frequency signals.

III. Task III

Methods

For Task 3, we were asked to make the given servo motor move from 0-90 degrees at a 0.5 Hz frequency. This was done using a Micro Servo 9g A0090 Servo Motor attached to the Arduino Due through the following circuit diagram:

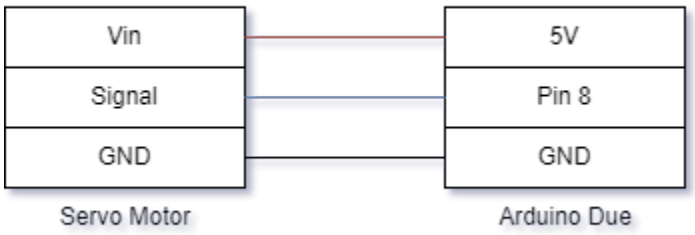


Figure 11: Circuit Diagram for the Servo Motor

Next, a sine wave was generated with the arduino at a 0.5 Hz frequency, amplitude of 45 and centered at 45. This was created to correspond with the input of the servo.write() function which takes in an angle to control the motor. The Arduino Servo library was included to help control the motor and was used in the code (**Appendix B**).

Results

The servo moved between 0 and 90 degrees as expected at a 0.5 Hz frequency. See the attached recording for reference.

Discussion

The design worked as expected.

IV. Task IV

Methods

In Task 4, we were asked to combine the force sensor, the filter, and the servo motor so that the force applied to the FSR changed the position of the motor. This required putting the output of the IA from Task 1 into the filter in Task 2. The output of this filter was read and processed by the arduino and a corresponding PWM signal was generated and outputted to the Servo Motor on a separate pin from the Arduino Due. The complete Circuit is shown below:

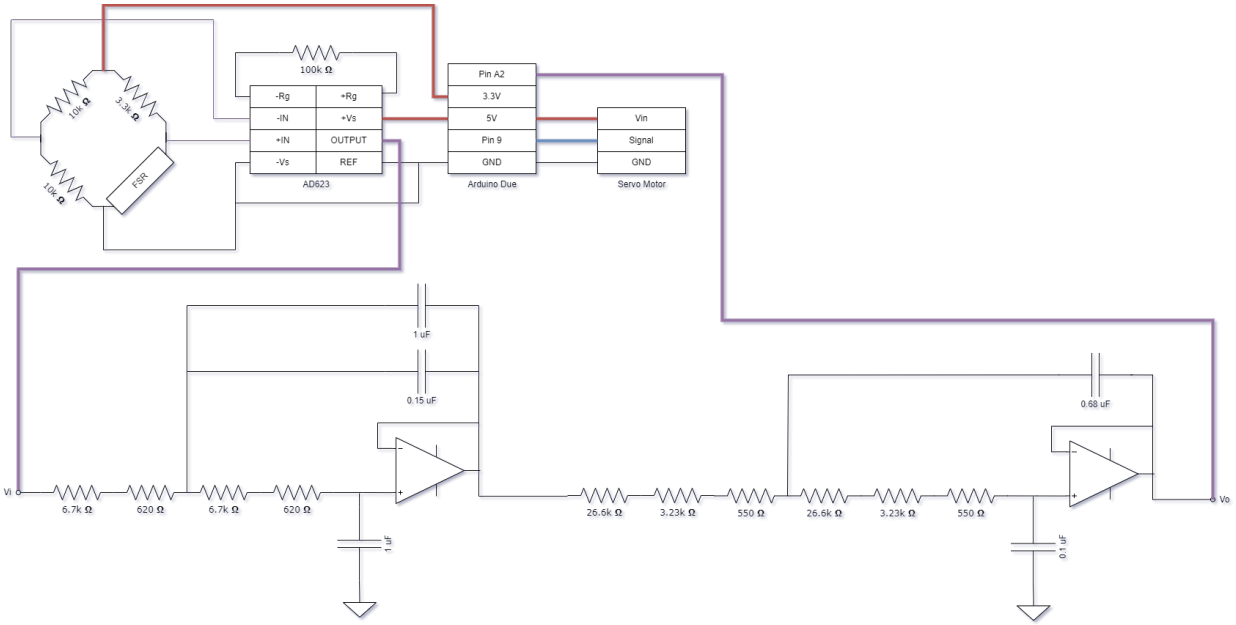


Figure 12: Finalized Circuit Diagram for Task 4 (**See Appendix D** for an image of the built circuit)

The signal into Pin A2 from the filtered amplified wheatstone bridge was interpreted by converting the signal from 0-3.3V (or 0-4095 or 12 bits) to 0-90 degrees. As shown in task 1, this was the inverse of force, such that an increase in force drove the motor towards 0 degrees. This degree conversion was then used as an input for the servo function. See **Appendix C** for the full code.

Results

The complete circuit worked as intended. Higher force resulted in 0 degrees while no force resulted in 90 degrees. See the attached recording for reference.

Discussion

The design worked as expected.

V. Appendix

A. Task 2 Code

```
int sinOutPin = 9;
int sinInPin = A2;

int time1;
int time2;

float hz = 2.0;
float w = hz * 2 * PI;

//  // additional code for min/max
//float win[100];
//int count;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  pinMode(sinOutPin, OUTPUT);
  pinMode(sinInPin, INPUT);

  analogReadResolution(12);

  time1 = micros();
  time2 = micros();

//  // additional code for min/max
//  count = 0;
}

void loop() {
  // put your main code here, to run repeatedly:
  time2 = micros();
  int output_val = 255/3.3*sin(w*time2/1000000) + 255/2;

  analogWrite(sinOutPin, output_val);

  if ((time2 - time1) > 2000) {
```

```
    time1 = micros();
    float input_val = analogRead(sinInPin)/4095.0 * 3.3;

//    // Additional Code for computing min and maxes
//    win[count%100] = input_val;
//    count++;
//    float min_val = 3.3;
//    float max_val = 0.0;
//    for (int i = 0; i < 100; i++) {
//      min_val = win[i] < min_val ? win[i] : min_val;
//      max_val = win[i] > max_val ? win[i] : max_val;
//    }

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

B. Task 3 Code

```
#include <Servo.h>

Servo servo;

int time1;
int time2;

int servoPWMPin = 8;
float hz = 0.5;
float w = hz * 2 * PI;

void setup() {
  // put your setup code here, to run once:
  servo.attach(servoPWMPin);
  time1 = micros();
  time2 = micros();
}

void loop() {
  // put your main code here, to run repeatedly:
  time2 = micros();
  int output_val = 45*sin(w*time2/1000000) + 45;

  if ((time2 - time1) > 100000) {
    servo.write(output_val);
  }

}
```

C. Task 4 Code

```
#include <Servo.h>

Servo servo;

int servoOut = 9;
int forceIn = A2;

int time1;
int time2;
int time3;

float hz = 2.0;
float w = hz * 2 * PI;

#define NUM_WIN 100
int window[NUM_WIN] = {0};
int count = 0;
int angle;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  pinMode(servoOut, OUTPUT);
  pinMode(forceIn, INPUT);
  servo.attach(servoOut);

  analogReadResolution(12);

  time1 = micros();
  time2 = micros();
  time3 = micros();
  angle = 0;
}

void loop() {
  // put your main code here, to run repeatedly:
  time2 = micros();
```

```
  if ((time2 - time1) > 2000) {
    time1 = micros();
    angle = analogRead(forceIn)/4095.0 * 90;
    window[count%NUM_WIN] = angle;
    count++;
  }
  if ((time2 - time3) > 100000) {
    time3 = micros();
    int avg = 0;
    for (int i = 0; i < NUM_WIN; i++) {
      avg += window[i];
    }
    avg = avg/NUM_WIN;
    servo.write(avg);
    Serial.println(avg);
  }
}
```

D. Image of Finalized Circuit for Task 4