Coil Gun Control System and User Interface

TA: Luke Wendt

Group: 52
Bryan Mbanefo (mbanefo2)
Adwaita Dani (aadani2)
Felipe Fregoso (fregoso2)

April 28, 2017

Contents

1 Introduction

1.1 Multi-stage Coil Gun System Overview

The multi-stage coil gun is a device that fires a small projectile at high speeds using an electromagnetic force. It works by generating a large current in a coil that winds around a projectile with a conduction path in the same direction as the winding. As mentioned in Ampere's law, a magnetic field is generated in the projectile's conducting path and by Faraday's law, the induced current in the projectile will create a magnetic field opposing that of the core. The projectile is then accelerated by a force proportional to the gradient of mutual inductance between the coils, coil and projectile currents.

The purpose of this project is to shed light on the many applications of electromagnetics. We chose this project because of the technical challenges it presents and because it aligns with the skills and academic focus of our group.

1.2 Objectives

The goal of this project is to design and build a control circuit and a user interface for the coil gun. The function of the control circuit is to accurately determine when to trigger the coils to increase the velocity of the projectile as it passes through. The goal is for the final speed of the projectile to be between 15-17 m/s. The launch speed and estimated distance travelled will be displayed on the user interface. We also need to ensure that our project is safe and adheres to the rules and regulations of IEEE and the University.

1.3 Benefits and Features of the coil gun

Benefits:

- Entertainment.
- Can be used as a teaching aid to display the effects of electromagnetism.
- Portable and easy to set up.
- Good application of power electronics and control systems.

Features:

- User Interface
- High mobility and easy set up
- Launching a projectile at speeds of 15m/s and above.

2 Design

2.1 Block Diagram

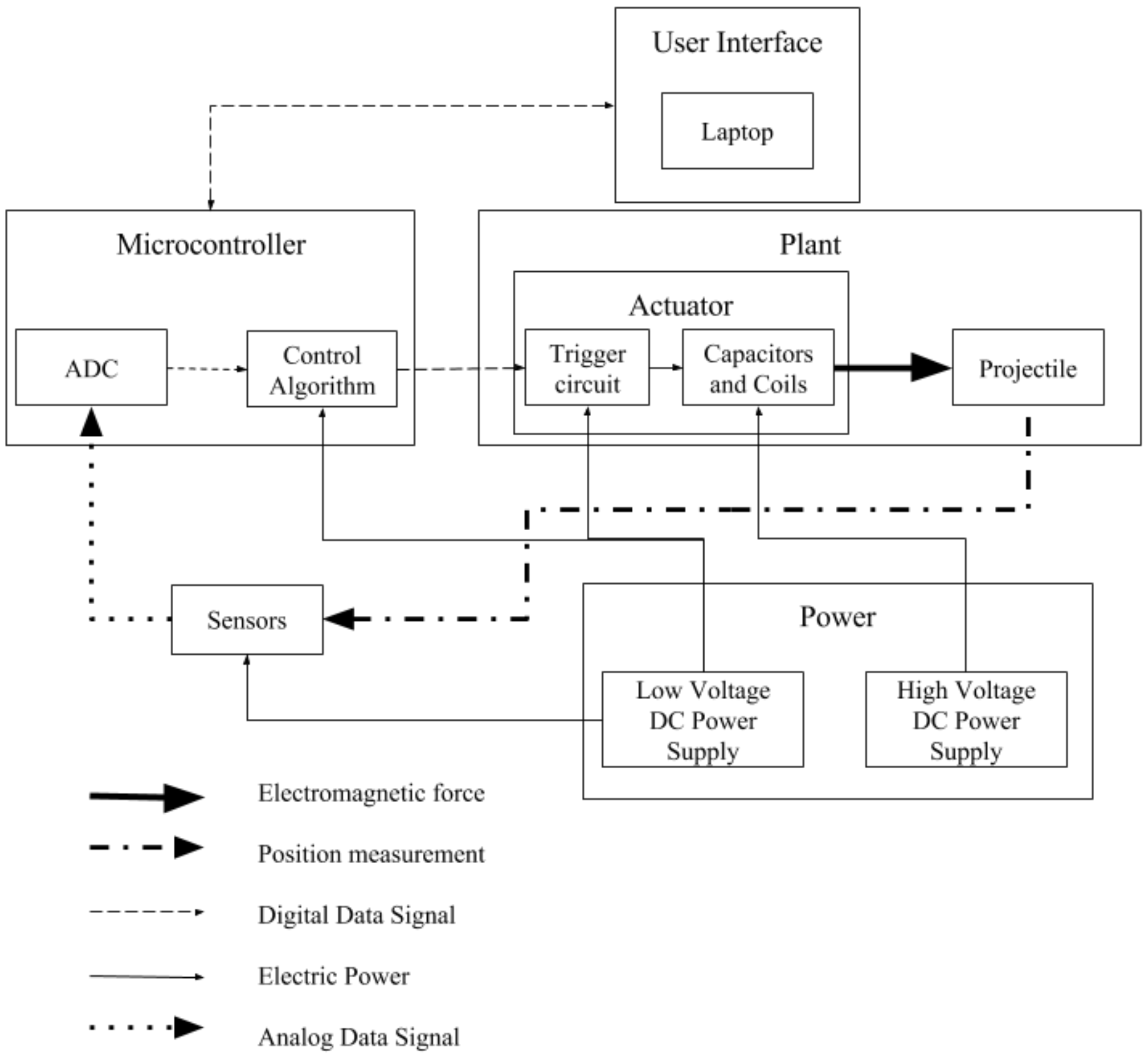


Figure 1: Block diagram for control system of the coil gun

In the final system, there is a LIDAR sensor that finds the current distance of the projectile within the barrel. Once the projectile has reached the optimal position to trigger the coil, the microcontroller will send a signal to the trigger circuit. The position of the projectile is sent over to the User Interface on the laptop through serial communication.

2.2 Physical design

2.2.1 MCU

The microcontroller needed to read in data and send it to the trigger circuit quickly enough for the coil gun to fire on time. The microcontroller we chose was the ATmega328. This microcontroller is also easy to program because the code can be uploaded onto an Arduino dev board and then the chip can be removed and placed on our PCB. The microcontroller receives analog data from the sensors that allow it to know where the projectile currently is within the barrel. When the projectile is within optimal firing position, the microcontroller will send a five volt digital signal to the trigger circuit that will discharge the projectile and propel it with more force. The microcontroller will also communicate with the laptop using serial communication. The microcontroller will send the data through the UART pin through and FTDI chip to the USB port. A diagram of the microcontroller can be seen below in figure 2.

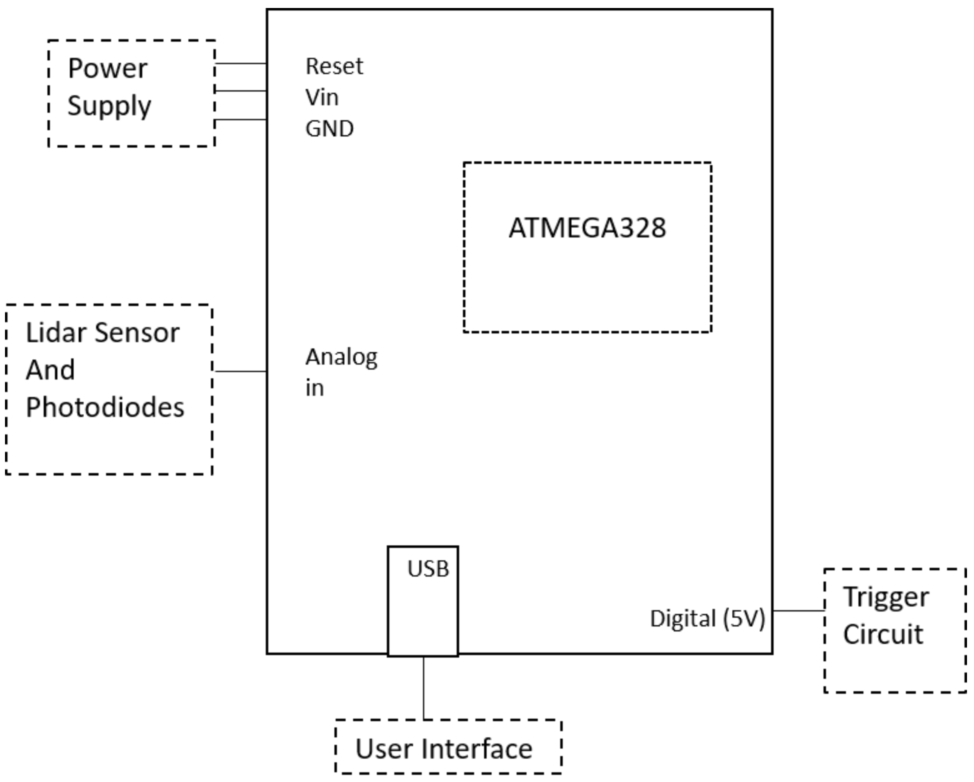


Figure 2: Interface between MCU and other devices

2.2.2 Sensors

The old IR sensors were replaced with a DT35-B15551 LIDAR sensor, this sensor was chosen mainly because of its speed of measurement and output time. It also provides both digital and analog output but for the purposes if the project, only the analog output was used. The LIDAR gives out an analog output with a 10V range going from 0V (minimum teach distance) to 10V (maximum teach distance). The Switching output of the LIDAR requires the use of IO LINK, however, the MCU does not understand this language which is why there is an IO LINK to SPI converter on the PCB, then we send the SPI data to the MCU for processing. The switching output is not used because the analog output was sufficient for out applications.

The LIDAR can work for various distances ranging from 5mm to 10,000mm and the range you need can be configured using the set and select buttons on the LIDAR. It also has various output speed modes ranging from 2ms to 64ms.

Table 1: LIDAR pin configuration

Pin (color)	Function
1 (brown)	+24V power
2 (White)	Analog/Switching output
3 (Blue)	COM
4 (Black)	Switching Output
5 (Grey)	+24V to reset to factory settings

Table 2: Output response time for LIDAR

	Super-Fast	Fast	Medium	Slow	Super-Slow
Output rate	2ms	4ms	8ms	16ms	64ms
Response time	4.5ms	12.5ms	24.5ms	48.5ms	192.5ms
Switching frequency	166Hz	50Hz	25Hz	12Hz	3Hz

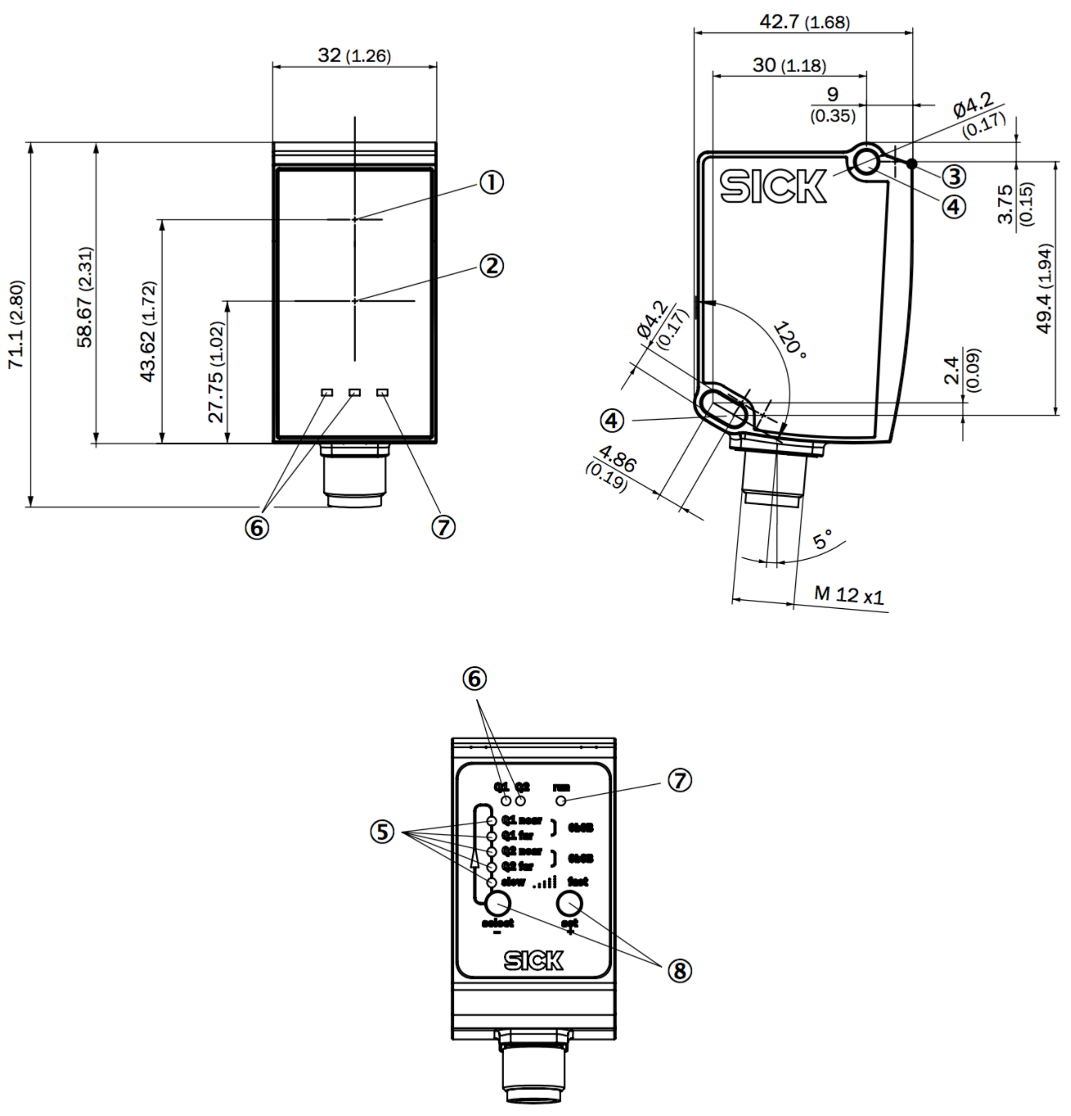


Figure 3: LIDAR Dimensions

2.2.3 User Interface

The user interface will consist of a laptop that has been dedicated for only the coil gun. The laptop communicates with the MCU to begin the fire. There is a red button that is clicked to fire the first coil and propel the projectile. The MCU also sends the current location of the projectile to the laptop for graphs to be created with the data. The MCU and the computer communicate with each other through USB. The button that is pressed and the graphs that are created are all in a Python GUI. This makes it easier for the user to fire the coilgun and read the data of the fire.

2.2.4 Power Supply

Sorensen SLC 48-4.1b

The power supply used is the Sorensen SLC 48-4.1b. Our power supply consists of 6 of these Sorensen power supplies. Each power supply supplies 50V, so we connect 4 of them in series to get an output voltage of 200V DC. This supplies power to our three capacitor banks as shown in the figure below:

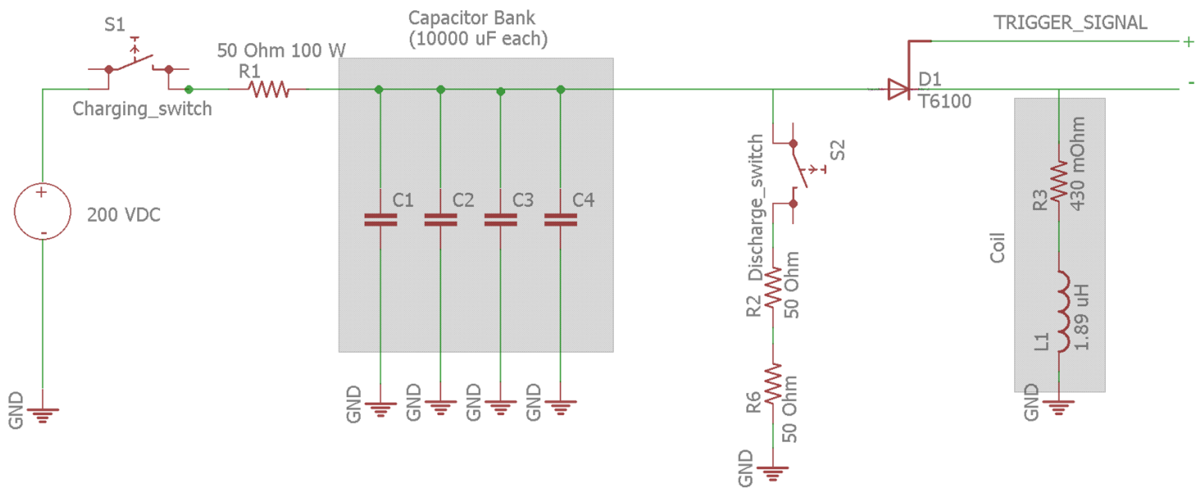


Figure 4: Power circuit to charge-discharge capacitors and energize coils

A discharge circuit is also included to enable us to discharge the capacitor banks if we choose not to fire the bullet after we have charged the capacitors. This helps to ensure safety especially when stowing away for a long time.

5V power supply

The system consists of a 24V AC/DC wall adapter that will go into a power jack in the PCB. The 24V can then go to a pin to power the LIDAR sensor. It is also connected to a buck converter that steps down the voltage to 5V. The circuit for the 24V and 5V supply can be seen below in figure 5.

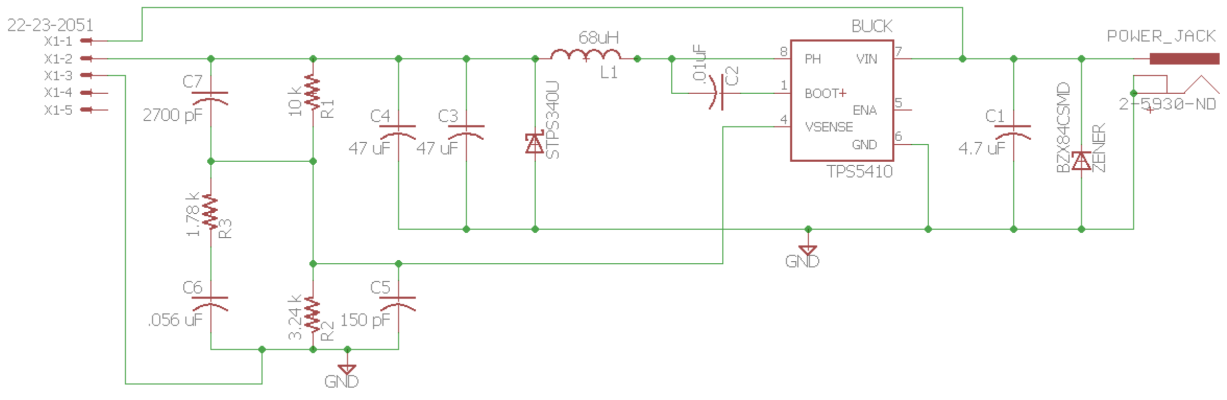


Figure 5: 24V and 5V power supply

2.2.5 Control Algorithm

The control algorithm that runs on the microcontroller will use a predictive control system to determine appropriate triggering times for the three stages of the coil gun. A mathematical model of the force applied to the bullet is calculated using equation 1.

$$f_e = i_{coil} i_{bullet} \frac{\partial M}{\partial z} \tag{1}$$

Mutual inductance between the coil and bullet is analytically calculated using elliptical integrals and yields a distance profile as shown in fig 6.

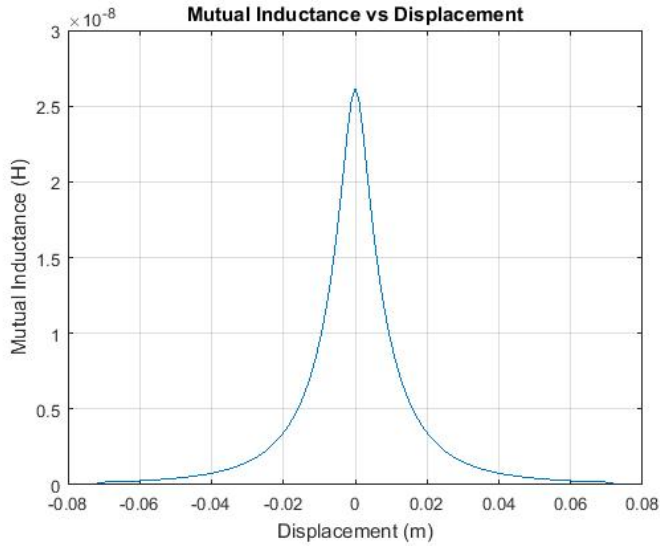


Figure 6: Mutual Inductance Vs Displacement

The current through the coils rises rapidly at first and then decreases creating a waveform as shown in figure 7.

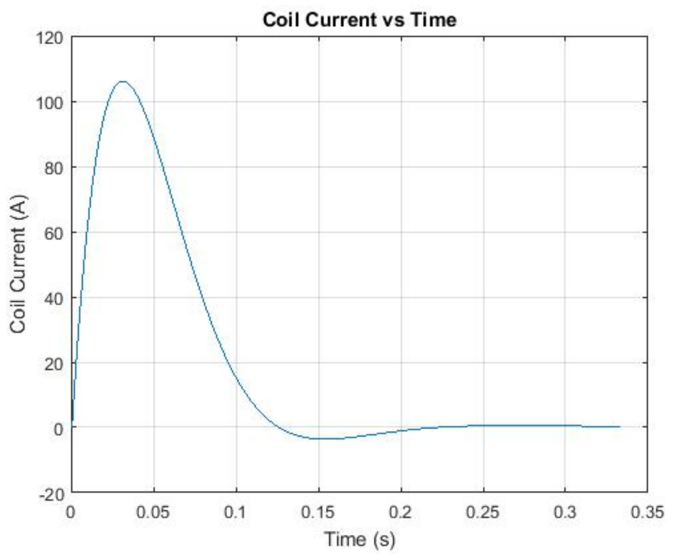


Figure 7: Coil current vs Time

This current profile in the coils leads to a brief force in one direction followed by a large force in the opposite direction. We observe that there is a maximum point for both these occurrences at points z = ±0.002m (Figure 8).

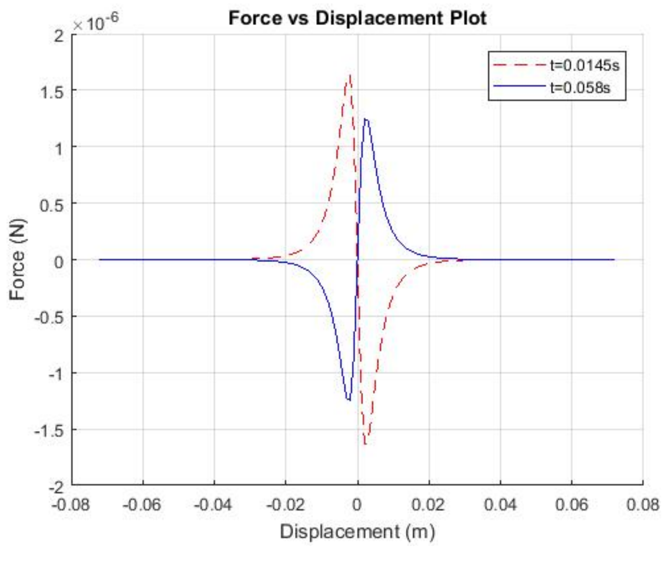
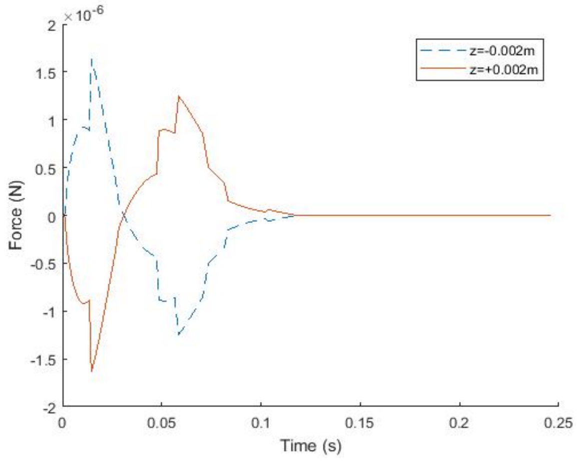


Figure 8: Force vs Displacement

The force and momentum values at these two points are compared in figure 9.

Since the net momentum at the point z = +0.002m is positive, it is the optimal firing position for the coil. We write our microcontroller code to cause the coil current to trigger when the bullet is at this optimal firing point.



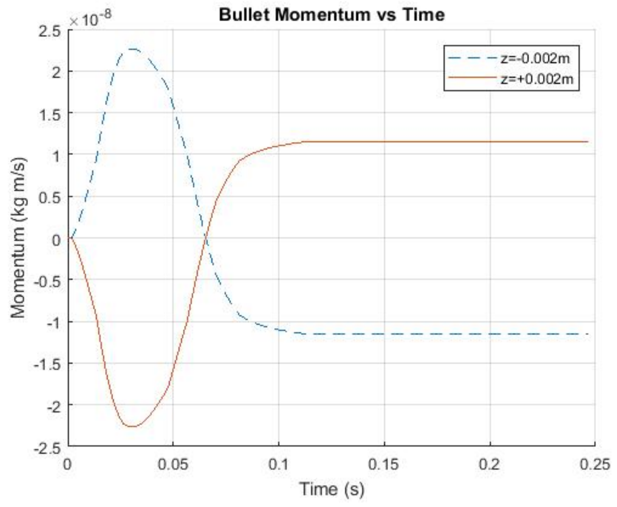


Figure 9: Force vs Time and Momentum vs Time

3 Design Verifications

3.1 MCU

We were able to sample our data into our MCU quickly because the ATmega328 has a clock speed of 16MHz. This allowed the MCU to send in sufficient data to the User Interface as well as trigger the coils at the proper time.

3.2 Power

The Sorenson Power Supply could give the voltage within the range of 195V-200V that we specified that it would be able to give. The 24V power supply was also able to be stepped down by the Buck Converter to be about 5V. Both tests were confirmed to be within their specified voltages with a multimeter.

3.3 Sensor

The sensor used was the DT35-B15551 LIDAR from SICK. This device was selected because it meets all our resolution and range requirements. It has a resolution of 0.1mm.

3.4 User Interface

The User Interface communicates to the microcontroller with USB as expected. The diagram below is the output displayed on the User Interface using the data we get from the microcontroller.

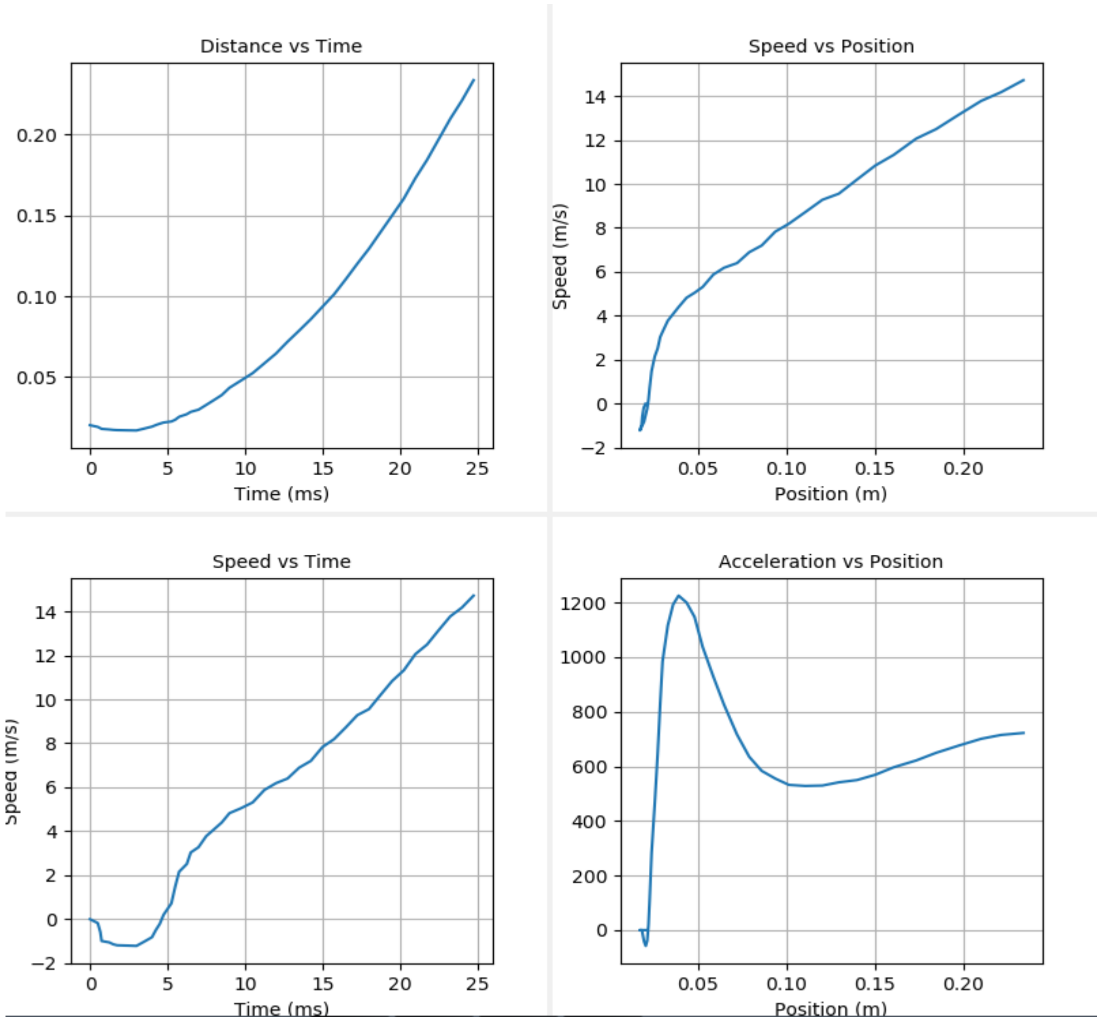


Figure 10: Data displayed by the User Interface

3.5 Trigger Accuracy

The delay in the microcontroller needed to be small so that it did not affect the timing of the firing. We required the trigger time delay to be less than 0.167 milliseconds. In the test, a square wave on a function generator was connected to an oscilloscope and the microcontroller. The microcontroller outputted a high signal onto the oscilloscope when it read in a high signal. The results of the test can be seen below in figure 11.

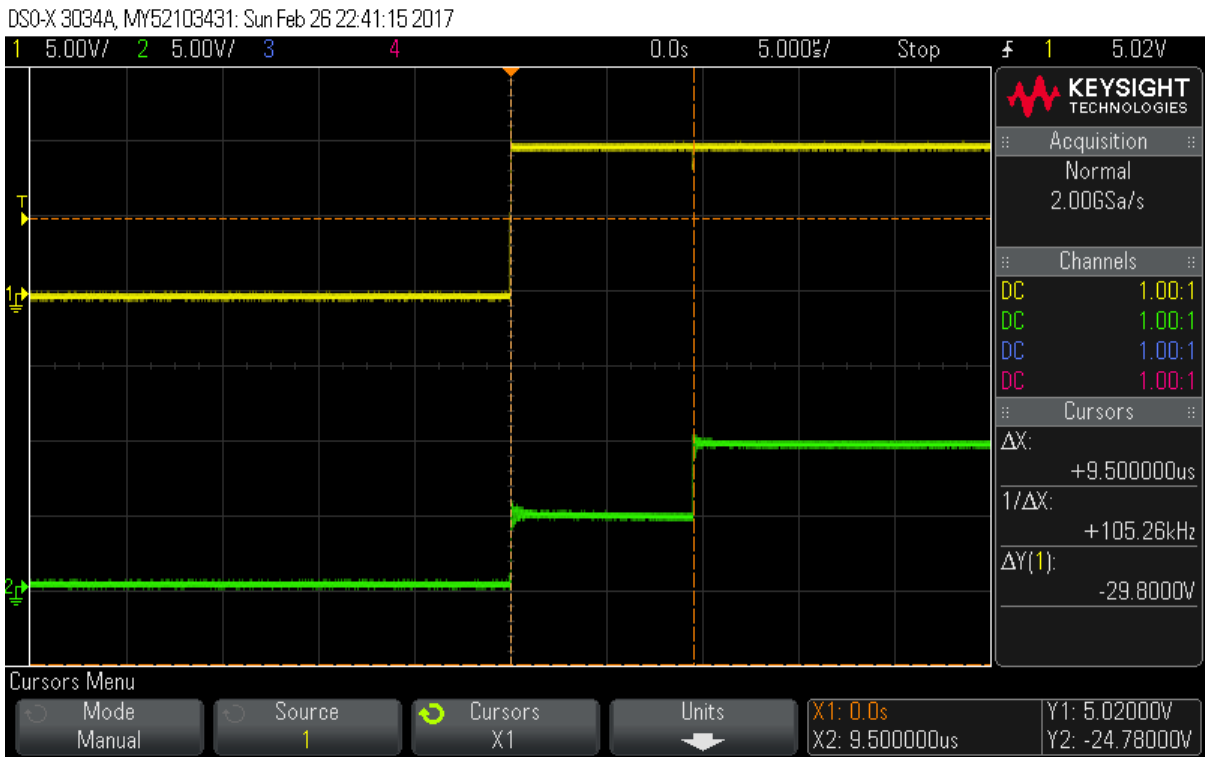


Figure 11: Timing delay in microcontroller

The top wave is the function generator and the bottom wave is the output from the microcontroller. As can be seen in the figure above, the distance measured in the delay is 9.5 microseconds. This is less than the required value to accurately trigger the SCRs.

4 Ethics and Safety

4.1 Ethics

We understand that this project has many applications including military purposes, however we would like to reiterate that we are doing this purely out of interest and for the purpose of learning the engineering process. We do not have any ambitions to cause harm or danger of any kind and we will uphold both the university and IEEE code of ethics

The most important code in the IEEE code of ethics for our project is #1; "to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment" [2]. This project involves making a weapon that could hurt people. It is important that guidelines are laid out for everyone to stay safe. Another important rule is #9, "to avoid injuring others, their property, reputation, or employment by false or malicious action” [2]. This project is meant to teach people about electromagnetics and pulsed power and it should not be used for malicious actions.

4.2 Safety

The LIDAR laser used in the project have a power of 250 mW. This means that the lasers fall into Class 3b of lasers. These lasers pose a threat if viewed directly with the eye [4]. Proper warning will be given to people to not use binoculars or other optical devices. People will also be advised to not stare at the laser.

Operating a gun is always dangerous, even more so when it involves the use of high voltage, current and fast moving projectiles. Appropriate safety precautions will be put in place to prevent undesirable and unforeseen circumstances. Some of these safety features include:

- A protective cover for the capacitor box.
- A discharge circuit for discharging charged capacitors.
- Three pole switch to prevent capacitor charge from flowing back into the source.

Some safety rules to follow:

- Never point the coil gun at something you do not want to shoot.
- Making sure the power is off before changing any of the circuit connections.
- Check and test for faulty equipment before use.
- Always assume that the capacitors are charged.
- Always discharge capacitors if stowing away for a long time.
- Always keep the gun pointed in a safe direction.
- Do not place your finger on the trigger button unless sure that you are ready to fire.
- Make sure everyone around you is following the safety rules.
- Do not directly touch the capacitors.
- The coil gun should never be operated by persons under the influence of alcohol or drugs.
- Children must never be left alone with the coil gun.

5 Cost

5.1 Labor

Name	Hourly Rate ($)	Hours = 12 weeks × 15 hours per week	Total × 2.5 ($)
Adwaita Dani	33	180	14,850
Bryan Mbanefo	33	180	14,850
Felipe Fregoso	33	180	14,850
Total	99	540	44,550

5.2 Parts

Parts	Part #	Quantity	Unit Cost ($)	Cost ($)
Zener Diode 47V 225MW	BZX84C47LT1GOSCT-ND	5	0.14	0.7
AC/DC WALL MOUNT ADAPTER 24V 24W	237-1461-ND	1	17.28	17.28
Power Jack	732-5930-ND	2	1.05	2.1
Zener Diode 30V 225MW	BZX84C30LT1GOSCT-ND	15	0.131	1.97
Capacitor 270pF 50V	311-1191-1-ND	15	0.075	1.13
Capacitor 0.1pF 250V	478-6549-1-ND	15	0.68	10.2
Capacitor 1pF 50V	311-1089-1-ND	15	0.093	1.4
OP-Amp	497-1580-1-ND	10	0.314	3.14
Capacitor 10pF 50V	311-1099-1-ND	15	0.059	0.89
FTDI Chip	768-1135-1-ND	7	2.38	16.66
Dual Diode	MMBD2837LT1GOSCT-ND	10	0.186	1.86
Voltage Regulator	296-42411-1-ND	20	1.231	24.62
Capacitor .47uF 10V	399-9247-1-ND	10	0.134	1.34
MCU	ATMEGA328-PU-ND	5	1.96	9.8
Resistor 10kΩ	311-10KARCT-ND	30	0.018	0.54
Resistor 1MΩ	311-1.00MCRCT-ND	15	0.021	0.32
Capacitor 1uF 16V	311-1365-1-ND	15	0.076	1.14

IO-Link Master	MAX14824GTG+-ND	8	5.21	41.68
16 MHz Clock	X433-ND	3	0.69	2.07
Buck Converter	296-26982-1-ND	5	2.7	13.5
Capacitor 2.7uF 10V	399-3127-1-ND	5	0.45	2.25
Capacitor 2700pF 50V	311-1130-1-ND	5	0.1	0.5
Capacitor .056uF 50V	478-1392-1-ND	5	0.16	0.8
Capacitor 150pF 50V	399-1125-1-ND	5	0.1	0.5
Capacitor 47uF 6.3V	311-1897-1-ND	10	0.346	3.46
Inductor 68uH 2mA	445-1069-1-ND	5	0.21	1.05
Schottky Diode 40V 3A	497-2465-1-ND	5	0.48	2.4
Pin Head .1"	SAM1184-05-ND	5	1.19	5.95
Capacitor 4.7uF 25V	490-3335-1-ND	10	0.112	1.12
Capacitor 10uF 10V	399-4925-1-ND	5	0.13	0.65
Capacitor 10000pF 50V	311-1136-1-ND	5	0.1	0.5
Resistor 10kΩ	311-10KARCT-ND	5	0.1	0.5
Resistor 3.24kΩ	311-3.24KCRCT-ND	5	0.1	0.5
Resistor 1.78kΩ	311-1.78KCRCT-ND	5	0.1	0.5
Capacitor .1uF 50V	478-1395-1-ND	5	0.1	0.5
PCB1		5		26
PCB2		5		26
LIDAR	DT35-B15551	1	413	413
IO-Link Cable		1	10	10
			total	648.50

Total of Labor and Parts = $45,198.50

6 Accomplishments and Future Work

6.1 Accomplishments

After a lot of hard work though the semester we happy to say that we had three major accomplishments. The first was that we could get a more accurate way of measuring the position of the projectile as it moves through the barrel and get a continuous stream of data. Using this data, we could then plot the speed profile of the projectile as it moves through the barrel.

Second, we attained a top speed of 14.36m/s using only two capacitor stages, which is ~4.36m/s faster than that achieved by the sensing system in the past using three capacitor stages. Lastly, we created a GUI platform to fire the projectile and visualize data related to the fire.

6.3 Uncertainties

The LIDAR sensor that measures projectile position currently has a resolution of 0.1mm for static targets. Since the current firing speeds are reasonably low, we have not seen any significant error in position detection even when the bullet is in motion. But if in the future the projectile speed was increased significantly, the LIDAR-based projectile position measurements might be less accurate due to Doppler shift in the Tx-Rx optical signals. The magnitude of this possible error has not been quantified as yet, and may be a subject for future work.

6.2 Future Work

Some future work to be added to optimize the system would be to add striped bullet sensing to improve the speed measurement. The photodiode and receiver will be placed at angles so that when the projectile is in the position, the return signal is reflected to the receiver. The projectile itself will have to be painted with black and white stripes. This will vary the intensity of the reflected light and give us a square wave that tells us how many strips have been passed and how far into the barrel has the projectile moved.

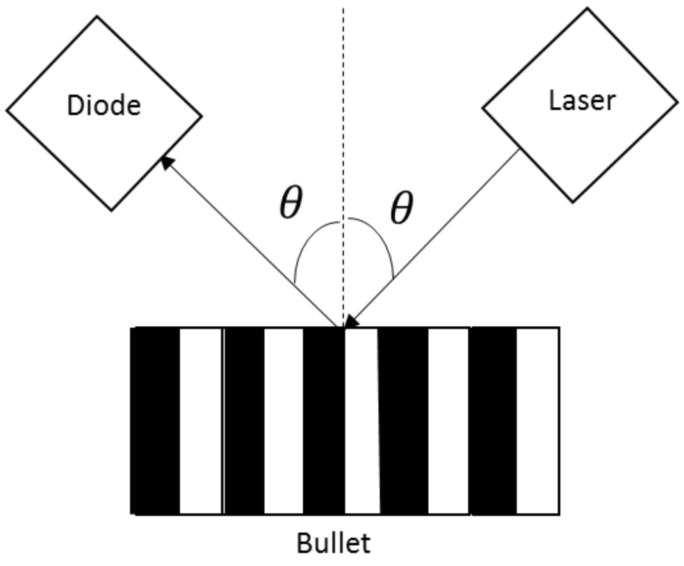


Figure 12 : Block diagram showing the placement of photodiodes

Another thing to be done would be to make the trigger signal a function of both the position and velocity of the bullet. This will help optimize the firing because the projectile does not always move at the same velocity so in different fire cycles the time it takes to for the projectile to travel a fixed distance will vary which will also vary the optimal time of fire because by the time the current rushes into the coil, the projectile may not be in the same optimal position

7 References

[1] Reinhard, K., “A Methodology for Selecting an Electromagnetic Magnetic Gun,” M.S. thesis, Dept. Elect. Eng., Univ. of Texas, Austin, TX, 1992.

[2] IEEE, “IEEE IEEE Code of Ethics”, 2017. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 8-Feb-2017].

[3] Dagdagan, J., Ko, Y., Nanavati, S., “Multistage Coil Gun”, 2013. [Online]. Available: https://courses.engr.illinois.edu/ece445/projects.asp. [Accessed: 1-Feb-2017].

[4] "Laser Classification | Environmental Health and Safety", *Ehs.research.uiowa.edu*, 2017. [Online]. Available: https://ehs.research.uiowa.edu/laser-classification. [Accessed: 25-Feb- 2017].

[5] V. profile, "AIXIZ 650nm 5mw 12x30mm laser module 3.2VDC", *Darutid.blogspot.com*, 2017. [Online]. Available: http://darutid.blogspot.com/2013/03/aixiz-650nm-5mw-12x30mm-laser-module.html. [Accessed: 25- Feb- 2017].

[6] S. I. Babic and C. Akyel, "Magnetic Force Calculation Between Thin Coaxial Circular Coils in Air," in IEEE Transactions on Magnetics, vol. 44, no. 4, pp. 445-452, April 2008.

8 Appendix

8.1 Arduino Code

```
int lidar = A3;

int lval = 0;

int dpin = 5; // the pin that will go off to fire the first coil

int fire = 6; // pin that will go off to fire the second coil

int last = 7; // pin to fire the last coil

int go = 0;

int beg = 0;

byte thebyte;

int elapse;

int elapse_st;


void setup()

{

  Serial.begin(115200);

  pinMode(dpin, OUTPUT);

  pinMode(fire,OUTPUT);

  pinMode(last,OUTPUT);

}

void loop()

{


  if (Serial.available() > 0) { //only run this in loop first time checks for the fire command

       thebyte = Serial.read();

       if (thebyte == '6') {

         go = 1;

       }
```

```
}


if(go == 1) //start once the fire command has been given
{
 if (beg == 0)
 {
   elapse_st = millis();
   beg = beg + 1;
 }
 elapse = millis( )- elapse_st; //calculate te elapsed time of the fire
 lval = analogRead(lidar); //get the analalog value from the sensor




 Serial.println(lval);    //send the value to python
 Serial.println(elapse);  //send time to python
 digitalWrite(dpin, HIGH); // first coil fired


 if(lval > 347)
 {
    digitalWrite(fire, HIGH);      //second coil fired
 }


 if(lval > 684)
 {
    digitalWrite(last, HIGH);      // third coil fired


 }
 if(lval > 1022)     //send that the projectile has left the coil reset pins for next fire
```

```
  {
    Serial.println('e');
    digitalWrite(dpin, LOW);
    digitalWrite(fire, LOW);
    digitalWrite(last, LOW);
    go = 0;
    beg = 0;
  }
 }


}
```

8.2 Python GUI

```
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import tkinter as tk
import serial
import matplotlib.pyplot as plt
import csv


file_name = 'three_stage_solidAl_1May17_2'


values = []
```

```
time = []

plt.ion()

cnt = 0


serialArduino = serial.Serial(port='COM8',baudrate=115200, timeout=10.0)


#class to create the buttons and cap entries

class MyFirstGUI:

   def __init__(self, master):

      self.master = master


      master.title("Coil Gun GUI")


      self.fire_button = tk.Button(master, text="FIRE",height =2,
width=10,bg="red",fg="white",command=self.fire)

      self.fire_button.grid(row=0,column=0)





   def fire(self):

       flag =1

       if(flag== 1):

         try:

            serialArduino.write(bytes(b'6'))

         except:

            print('This shit doesnt work.')

    #   print("capacitor is fired!")

       def funct():
        valueRead = None
        while(valueRead != b'e'):
            while(serialArduino.inWaiting() == 0):
                pass
            valueRead = serialArduino.readline()
            timeRead = serialArduino.readline()
            if (valueRead==b'e' or timeRead == b'e'):
                return()
    #check if valid value can be casted
            try:
              valueInInt = float(valueRead)
              timeInInt = float(timeRead)
              print(valueInInt)
              if valueInInt <= 1024:
                  if valueInInt >= 0:
                      values.append(valueInInt * 0.00035)
                      time.append(timeInInt)
                      if valueInInt == 1023:
                          return()
#                  values.pop(0)
                  #drawnow(plotValues)
                  else:
                      print('invalid number')
              else:
                  print('number too large')
            except ValueError:
              print('invalid cannot cast')
              continue
        serialArduino.flushInput()
        serialArduino.flushOutput()
        funct()

        meter_values = values

        # Code to filter values
        # Data : meter_values, time
        low_pass_data = []
        window_size = 20 # Size of averaging window
        for idx in range(len(meter_values)):
          averaging_data = []
          for avg_idx in range(window_size):
            if((idx-avg_idx)>=0):
              averaging_data.append(meter_values[idx-avg_idx])    # Use window size from previous data
values
          low_pass_val = np.mean(np.array(averaging_data))
          low_pass_data.append(low_pass_val)


        # Filter Time Data to make it continuous
        time_low_pass = []
        window_size_time = 4 # Size of averaging window
        for idx in range(len(meter_values)):
          averaging_data = []
          for avg_idx in range(window_size_time):
            if((idx-avg_idx)>=0):
              averaging_data.append(time[idx-avg_idx])    # Use window size from previous data values
```

```
    low_pass_val = np.mean(np.array(averaging_data))
    time_low_pass.append(low_pass_val)

 fig4 = Figure(figsize=(4,4))
 d = fig4.add_subplot(111)
 d.grid(True)

 d.plot(time_low_pass,low_pass_data,label='values')
 d.set_title("Distance vs Time",fontsize=10)
 d.set_ylabel("Position (m)", fontsize=10)
 d.set_xlabel("Time (ms)", fontsize=10)




# Calc speed of projectile wrt time
# Data : meter_values, time
 speed_calc_window = 1
 speed_data = []
 for idx in range(len(meter_values)):
    if((idx-speed_calc_window) >= 0):
        delta_position = low_pass_data[idx] - low_pass_data[idx-speed_calc_window]
        delta_time = time_low_pass[idx] - time_low_pass[idx-speed_calc_window]
        if(delta_time == 0):
            delta_time = 0.0833
        speed_val = (delta_position*1000)/(delta_time)
        speed_data.append(speed_val)
    else:
        speed_data.append(0.0)
        speed_low_pass_data = []

        window_size_speed = 12 # Size of averaging window

        for idx in range(len(speed_data)):

          averaging_data = []

          for avg_idx in range(window_size_speed):

             if((idx-avg_idx)>=0):

                averaging_data.append(speed_data[idx-avg_idx])    # Use window size from previous data
values

             else:

                averaging_data.append(0.0)

          low_pass_val = np.mean(np.array(averaging_data))

          speed_low_pass_data.append(low_pass_val)

        plt.figure()

        plt.plot(time_low_pass,speed_low_pass_data)



        with open('{0}.csv'.format(file_name), 'w') as writefile:

          csvwriter = csv.writer(writefile)

csvwriter.writerow(['Time','Time_Low_Pass_Val','ADC_value','Low_Pass_Val','Speed_Val','Speed_Low_P
ass_Val'])    # Write Header

          for idx in range(len(time)):

             csvwriter.writerow(['{0}'.format(time[idx]), '{0}'.format(time_low_pass[idx]),
'{0}'.format(values[idx]), '{0}'.format(low_pass_data[idx]), '{0}'.format(speed_data[idx]),
'{0}'.format(speed_low_pass_data[idx])])



        min_position = min(low_pass_data)
```

```
max_position = max(low_pass_data)

min_time = min(time)

max_time = max(time)

avg_speed = (max_position-min_position)*1000/(max_time-min_time)

print('Average Speed:{0} m/s'.format(avg_speed))

print('Exit Speed:{0} m/s'.format(speed_low_pass_data[-1]))



# Calc acceleration vs position

accel_calc_window = 10

accel_data = []

for idx in range(len(meter_values)):

  if((idx-accel_calc_window) >= 0):

     delta_speed = speed_low_pass_data[idx] - speed_low_pass_data[idx-accel_calc_window]

     delta_time = time_low_pass[idx] - time_low_pass[idx-accel_calc_window]

     accel_val = (delta_speed*1000)/(delta_time)

     accel_data.append(accel_val)

  else:

     accel_data.append(0.0)


accel_low_pass_data = []

window_size_accel = 5 # Size of averaging window

for idx in range(len(speed_data)):

  averaging_data = []

  for avg_idx in range(window_size_accel):

     if((idx-avg_idx)>=0):

        averaging_data.append(accel_data[idx-avg_idx])    # Use window size from previous data
values

     else:
        averaging_data.append(0.0)
    low_pass_val = np.mean(np.array(averaging_data))
    accel_low_pass_data.append(low_pass_val)



plt.figure()
plt.plot(low_pass_data,accel_low_pass_data)

print('Average Speed:{0} m/s'.format(avg_speed))
print('Exit Speed:{0} m/s'.format(speed_low_pass_data[-1]))

fig1 = Figure(figsize=(4,4))
acc = fig1.add_subplot(111)
acc.grid(True)
acc.plot(low_pass_data,accel_low_pass_data,label='values')
acc.set_title("Acceleration vs Position",fontsize=10)
acc.set_ylabel("Acceleration (m/s^2)", fontsize=10)
acc.set_xlabel("Position (m)", fontsize=10)




fig5 = Figure(figsize=(4,4))
spd = fig5.add_subplot(111)
spd.grid(True)
meter_values = values
spd.plot(time_low_pass,speed_low_pass_data,label='values')
spd.set_title("Speed vs Time",fontsize=10)
spd.set_ylabel("Speed (m/s)", fontsize=10)
spd.set_xlabel("Time (ms)", fontsize=10)
```

```
        fig6 = Figure(figsize=(4,4))
        dist_spd = fig6.add_subplot(111)
        dist_spd.grid(True)
        meter_values = values
        dist_spd.plot(low_pass_data,speed_low_pass_data,label='values')
        dist_spd.set_title("Speed vs Position",fontsize=10)
        dist_spd.set_ylabel("Speed (m/s)", fontsize=10)
        dist_spd.set_xlabel("Position (m)", fontsize=10)

        canvas = FigureCanvasTkAgg(fig4, master=self.master)
        canvas.get_tk_widget().grid(row =2 ,column=6,padx=2,pady=2)
        canvas.draw()

        canvas = FigureCanvasTkAgg(fig5, master=self.master)
        canvas.get_tk_widget().grid(row = 3,column=6,padx=2,pady=2)
        canvas.draw()

        canvas = FigureCanvasTkAgg(fig6, master=self.master)
        canvas.get_tk_widget().grid(row = 2,column=25,padx=2,pady=2)
        canvas.draw()

        canvas = FigureCanvasTkAgg(fig1, master=self.master)
        canvas.get_tk_widget().grid(row = 3,column=25,padx=2,pady=2)
        canvas.draw()

#class mclass:
```

```
root = tk.Tk()



my_gui = MyFirstGUI(root)

root.mainloop()
```