MECH420 Lab 3

Ratthamnoon Prakitpong
#63205165

A1.
To convert count to position/displacement, we can use factor of 200/1440 mm/counts. Derivation is in part 6.1.

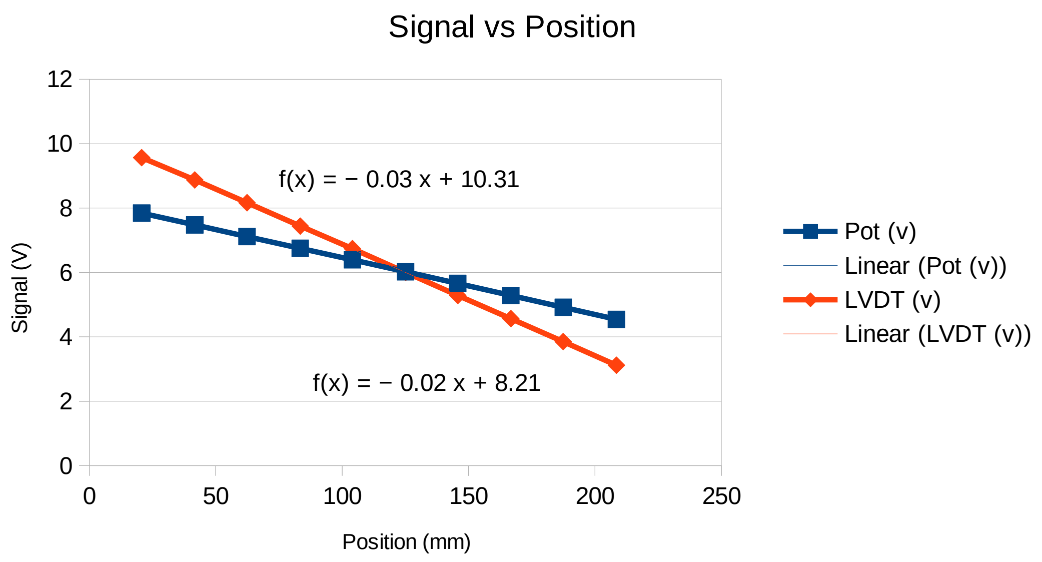


Fig A1: Plot of average signal

A2.
V_Pot = -0.0344x + 10.306
V_LVDT = -0.01757x + 8.2147
Where V is in volts, and x is in mm.

B1.
We get velocity from displacement with following formula:

$$v_n = (x_n - x_{n-1}) / 0.005\ s$$

To get position at voltage, we rearrange potentiometer voltage to position formula from A2:

$$x = (V_{Pot} - 10.306) / (-0.0344)$$

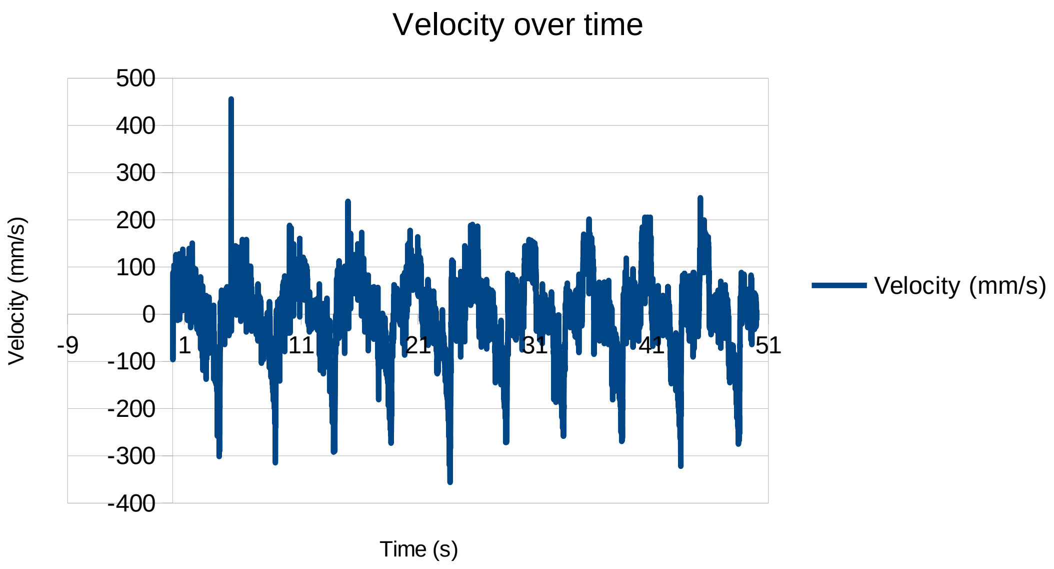


Fig B1: Velocity over time

The calculated velocities are very noisy.

B2.
We need to find a way to categorize velocity data. By plotting position over time, we can see that we have clean thresholds at 200 and 90 mm, where position data in between these threshold are clean data. We can use these threshold values to section out sets of velocity and tachometer voltage values. For example, because the first set of velocities is positive, we can say that all velocity and tachometer values between the first time position cross 90 mm and the first time position cross 200 mm belongs in the same set, which is set one. For second set, because velocity is negative, we pick values between time position crossed 200 mm and 90 mm. Repeat this until we run out of data.

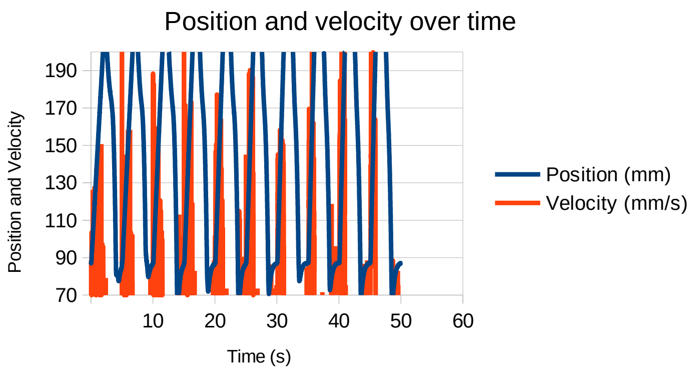


Fig B2.1: Position and velocity over time

By translating that logic into python code, we can separate values into sets, find average of each sets, and plot it to get the following plot:

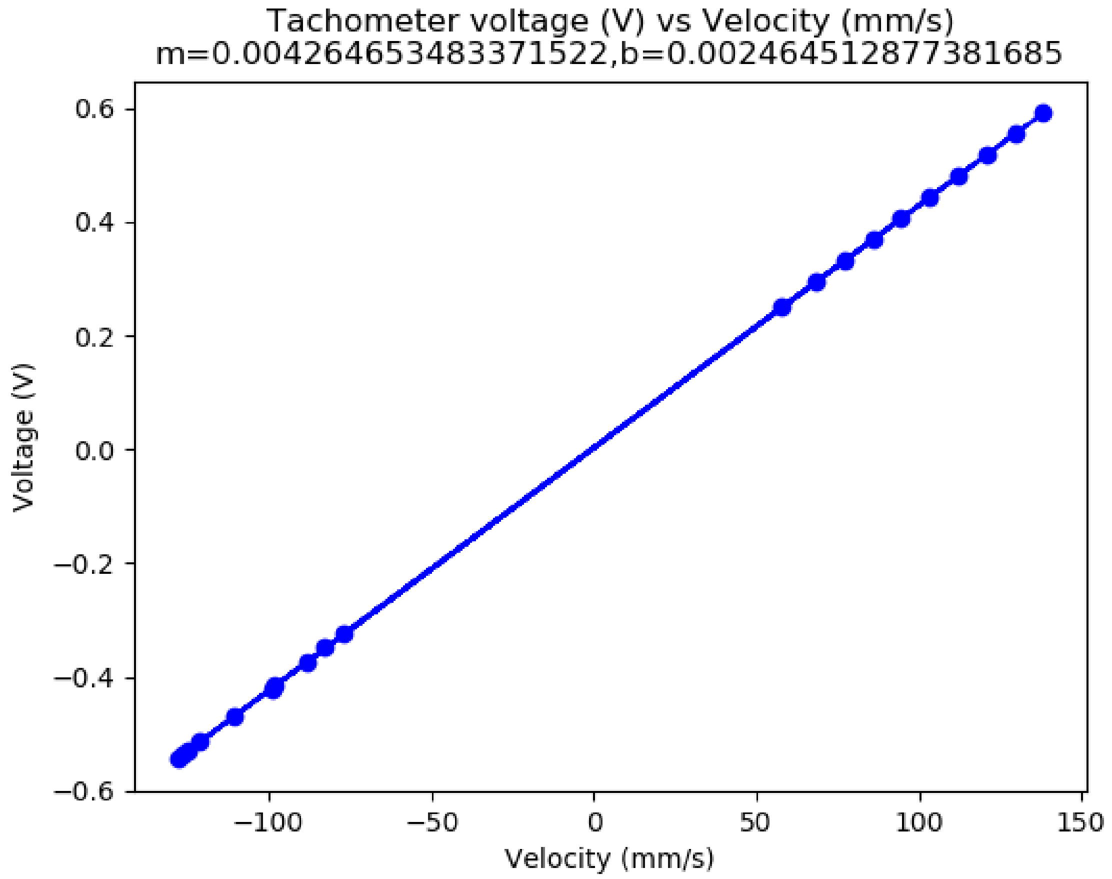


Fig B2.2: Tachmeter voltage vs velocity

B3.
V_tach = 0.00426v + 0.00246
where V_tach is in volts, and v is in mm/s.

C1.
For tachometer's acceleration values, we can find it from change in velocity:

a_n = (v_n – v_n-1) / 0.005 s

To get velocity at voltage, we rearrange tachometer voltage to velocity formula from B3:

v = (V_tach - 0.00246) / 0.00426

For accelerometer's acceleration values, we need sensitivity and offset. From datasheet, typical sensitivity of accelerometer is 1000mV/9.81m/s^2, or 1/9810 V/mm/s^2. To get offset from measurements, we can pick 2.5V as stated in datasheet as initial offset, and adjust offset so that the tachometer and accelerometer plots overlay. This way, we end up with 2.65V as offset, resulting in the equation:

a = (V_accelerometer – 2.65) * 9810

Translating these logic into python code, we can process measurements and plot them as follows:

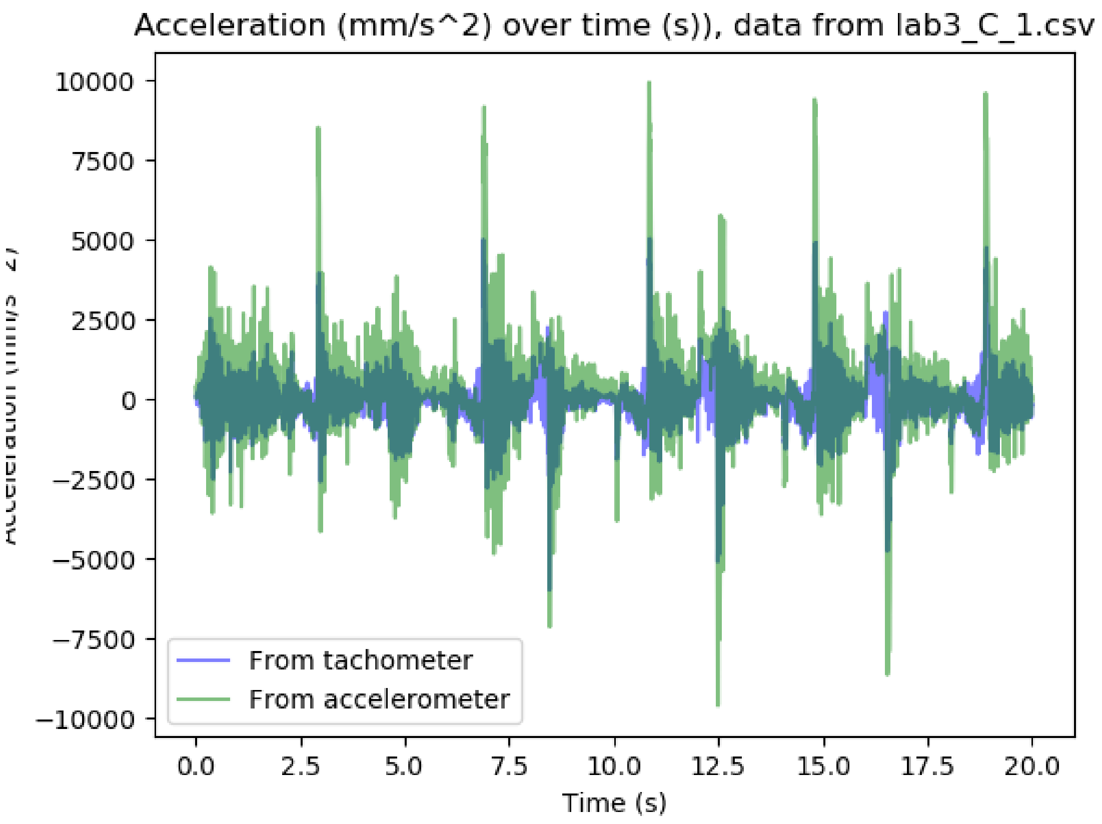


Fig C1.1: Accelerations over time, from dataset #1

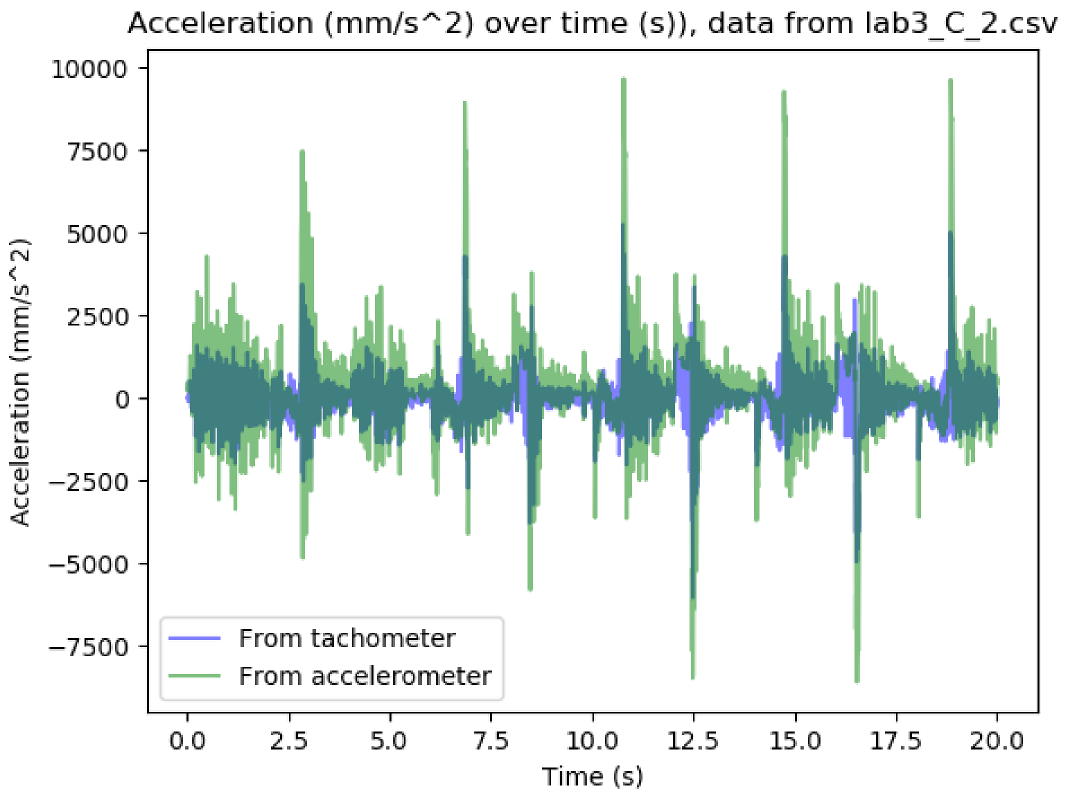


Fig C1.2: Accelerations over time, from dataset #3

The two plots compare favourably, although we can see that the acceleration from tachometer are scaled down from the acceleration from accelerometer. This may be due to some error in tachometer sensitivity, since accelerometer sensitivity is pulled directly from datasheet is it’s less likely to be wrong. This may also be due to some unaccounted offset in accelerometer data, since potentiometer and tachometer data has lined up well so far, and it’s statistically unlikely that these two independently-calibrated sensors are off at the same time.

C2.

Positions from tachometer and potentiometer line up nearly perfectly. This is likely because conversion equation for tachometer voltage was derived directly from potentionmeter’s equation. Position from accelerometer shows a quadratic drift. There may be an offset signal in accelerometer that wasn’t accounted for, so when integrated twice, it appears as a quadratic drift.

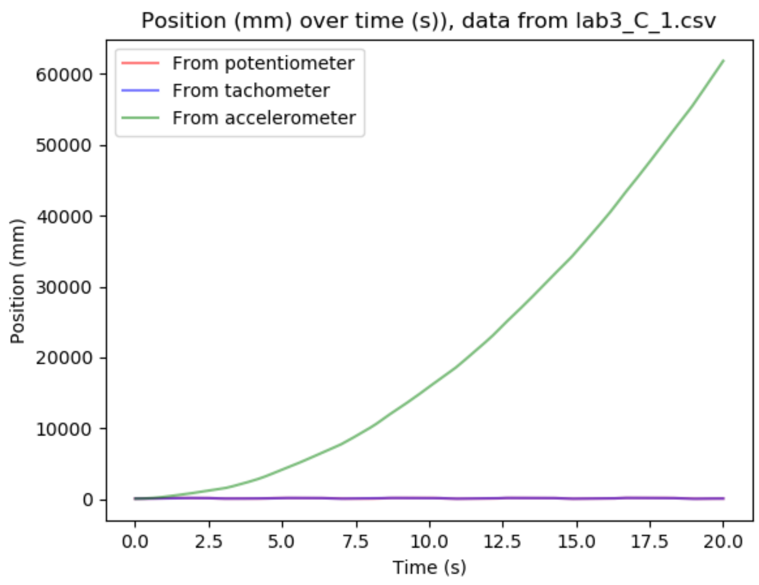


Fig C2.1: Positions over time, from dataset #1

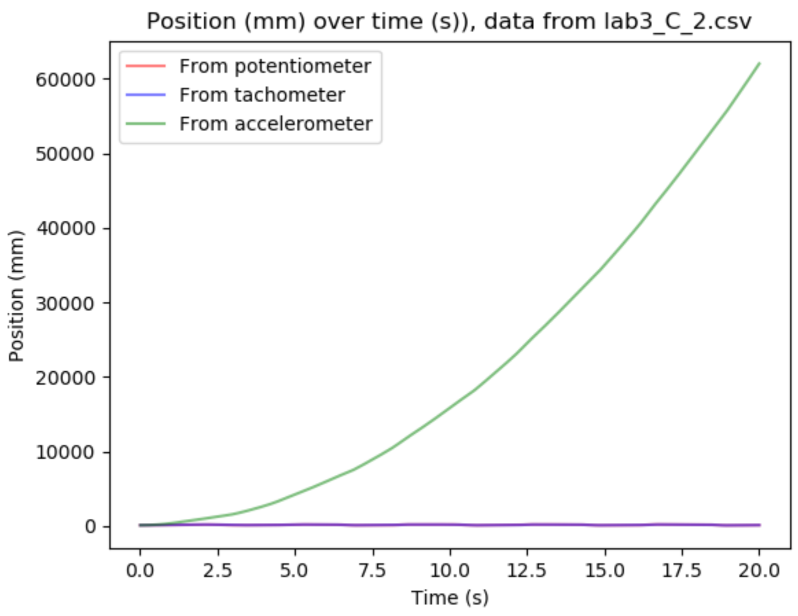


Fig C2.1: Positions over time, from dataset #2

With manual rescaling, we find that the values matched up decently well.

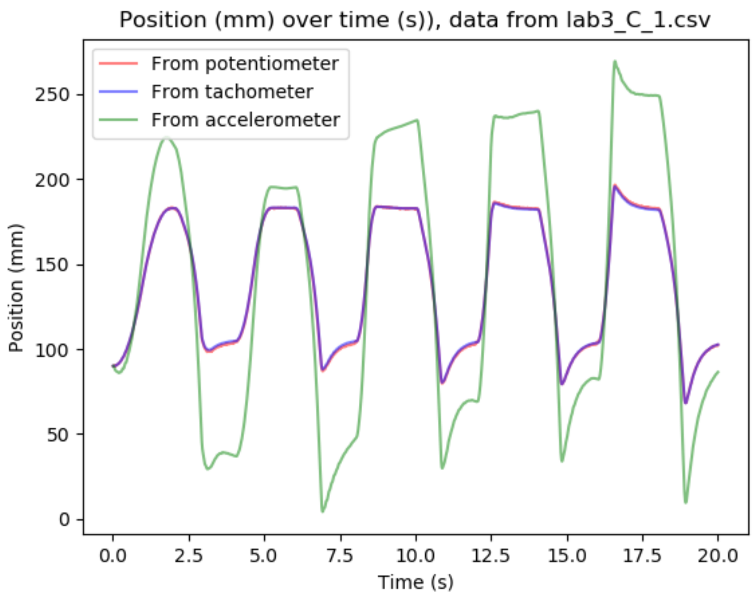


Fig C2.3: Positions over time, rescaled, from dataset #1

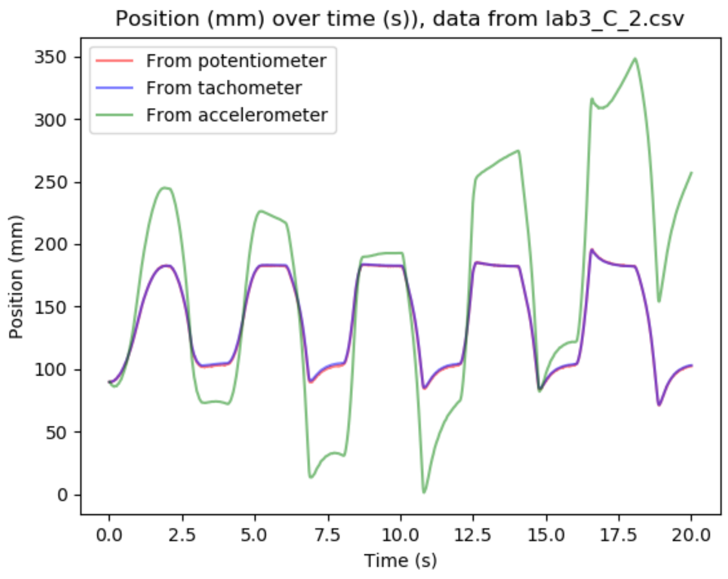


Fig C2.4: Positions over time, rescaled, from dataset #2

Combined with result in C1, we can conclude with higher likelihood that there is some error with accelerometer’s calibration.

6.1.
We know that the encoder reads 1440 counts per revolution, the belt's pitch is 10mm, and the belt's pulley has 20 teeth per revolution. From this, we can come up with a simple conversion factor of 200/1440 mm/count to get mm from counts.

6.2.A.
+/- 1.7g * 1000mV/g = +/- 1.7V

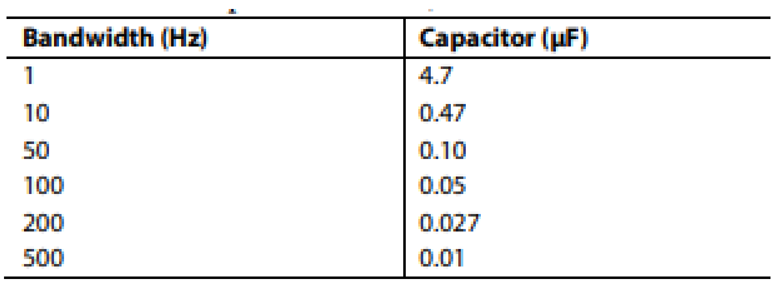
6.2.B.
5.5 kHz

6.2.C.
It is possible to achieve 1 to 500 Hz with commonly available capacitors. See 6.2.D for table.

6.2.D.
The relationship is:

$$f_{-3\,dB} = 1/(2\pi(32\text{ k}\Omega) \times C_{(X,\,Y)})$$
$$f_{-3\,dB} = 5\ \mu\text{F}/C_{(X,\,Y)}$$

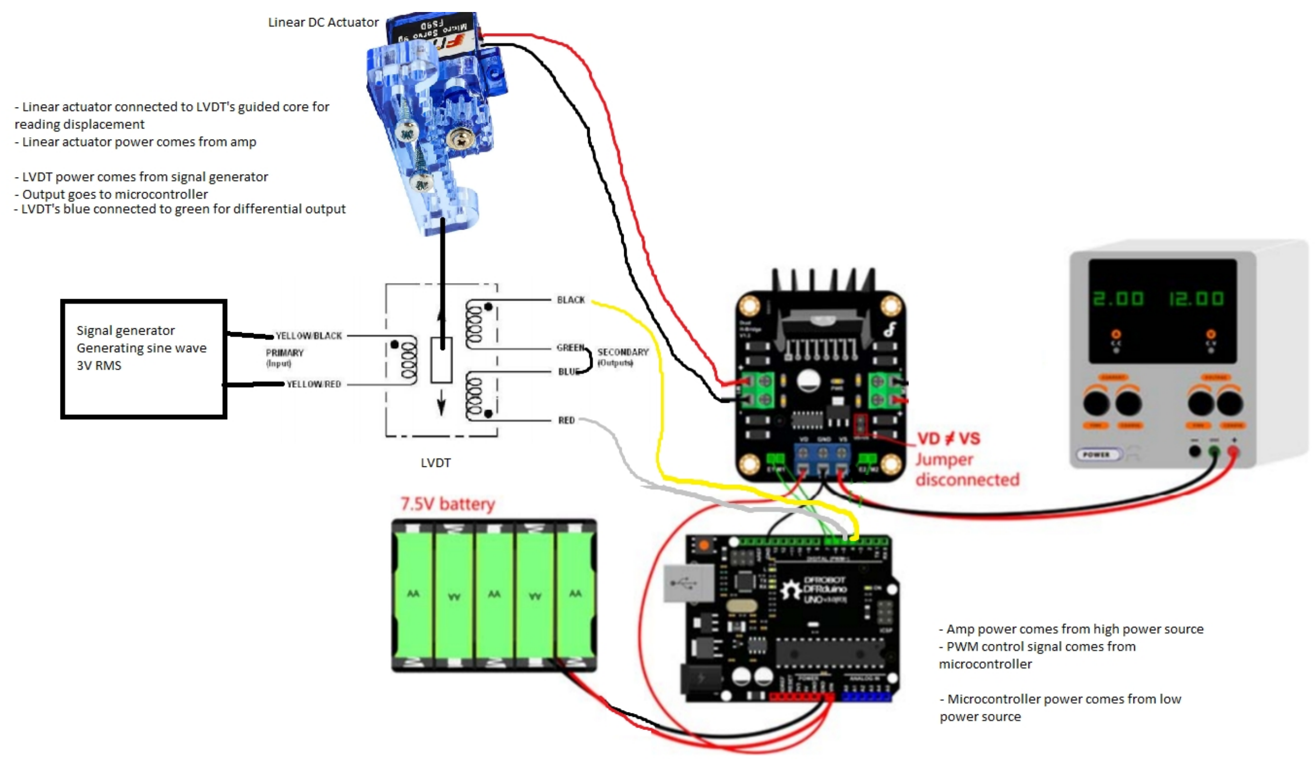
We can take common capacitance and calculate bandwidth per each capacitance value, and tabulate the results:



Bandwidth (Hz)	Capacitor (µF)
1	4.7
10	0.47
50	0.10
100	0.05
200	0.027
500	0.01

6.3.

Part	Datasheet
Linear DC actuator	https://resources.kitronik.co.uk/pdf/2595-linear-actuator-assembly-instructions.pdf
Microcontroller	https://www.farnell.com/datasheets/1682209.pdf
Actuator driver (PWM Amp)	https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DRI0002_Web.pdf
LVDT	https://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=HR&DocType=Data+Sheet&DocLang=English



- Linear actuator connected to LVDT's guided core for reading displacement
- Linear actuator power comes from amp

- LVDT power comes from signal generator
- Output goes to microcontroller
- LVDT's blue connected to green for differential output

- Amp power comes from high power source
- PWM control signal comes from microcontroller

- Microcontroller power comes from low power source

Fig 6.3: Simple schematic diagram

Appendix:

partB.py:

```
import os
import matplotlib.pyplot as plt
import numpy as np
dir_path = os.path.dirname(os.path.realpath(__file__))

name = 'lab3_B.csv'
avgVel = []
avgTach = []
highPosThres = 200
lowPosThres = 90
with open(dir_path+'\\'+name) as f:
    next(f) # flush header line
    next(f)
    counting = True
    dir = True # true = up, false = down
    tempVel = []
    tempTach = []
    for line in f:
        items = line.split(',')
        if len(items) > 1:
            curPos = float(items[5])
            if counting:
                if (dir and curPos > highPosThres) or (not dir and curPos < lowPosThres):
                        counting = False
                        avTempVel = sum(tempVel)/len(tempVel)
                        avTempTach = sum(tempTach)/len(tempTach)
                        avgVel.append(avTempVel)
                        avgTach.append(avTempTach)
                        tempVel = []
                        tempTach = []
            else:
                if dir:
                    if curPos < highPosThres:
                        counting = True
                        dir = not dir
                else:
                    if curPos > lowPosThres:
                        counting = True
                        dir = not dir

            if counting:
                tempVel.append(float(items[6]))
                tempTach.append(float(items[4]))

x = avgVel
```

```
y = avgTach
m, b = np.polyfit(x, y, 1)

plt.clf()
plt.plot(x, y, 'bo--')
plt.title('Tachometer voltage (V) vs Velocity (mm/s)\nm=' + str(m) + ',b=' + str(b))
plt.ylabel('Voltage (V)')
plt.xlabel('Velocity (mm/s)')
plt.savefig(dir_path + '\\' + 'partB' + '.png')
```

partC1.py:

```
import os
import matplotlib.pyplot as plt
import numpy as np
dir_path = os.path.dirname(os.path.realpath(__file__))



def tachVToVel(v):
    return (v - 0.0024645)/0.00426465

def accelVToA(v):
    return (v - 2.65)*9810*.96

names = ['lab3_C_2.csv', 'lab3_C_1.csv']
for name in names:
    t = []
    tachA = []
    accelA = []
    with open(dir_path+'\\'+name) as f:
        next(f) # flush header line
        init = next(f)
        items = init.split(',')
        prevTach = tachVToVel(float(items[4]))
        for line in f:
            items = line.split(',')
            if len(items) > 1:
                t.append(float(items[0]))
                curTach = 0
                if items[4].rstrip(): # some values are skipped for no reason
                    curTach = tachVToVel(float(items[4]))
                tachA.append((curTach - prevTach)/0.005)
                prevTach = curTach
                accelA.append(accelVToA(float(items[3])))

    plt.clf()
    plt.plot(t, tachA, 'b', label='From tachometer', alpha=0.5)
    plt.plot(t, accelA, 'g', label='From accelerometer', alpha=0.5)
    plt.title('Acceleration (mm/s^2) over time (s)), data from ' + name)
    plt.ylabel('Acceleration (mm/s^2)')
    plt.xlabel('Time (s)')
    plt.legend()
    plt.savefig(dir_path + '\\' + 'partC1_' + name + '.png')
```

partC2.py:

```
import os
import matplotlib.pyplot as plt
import numpy as np
from scipy import integrate
dir_path = os.path.dirname(os.path.realpath(__file__))


def tachVToVel(v):
    return (v - 0.0024645)/0.00426465

def accelVToA(v):
    return (v - 2.65)*9810

def potVToX(v):
    return (v - 10.306)/(-.0344)

names = ['lab3_C_2.csv', 'lab3_C_1.csv']
for name in names:
    t = []
    potX = []
    tachV = []
    accelA = []
    with open(dir_path+'\\'+name) as f:
        next(f) # flush header line
        for line in f:
            items = line.split(',')
            if len(items) > 1:
                t.append(float(items[0]))
                potX.append(potVToX(float(items[2])))
                curTach = 0
                if items[4].rstrip(): # some values are skipped for no reason
                    curTach = tachVToVel(float(items[4]))
                tachV.append(curTach)
                accelA.append(accelVToA(float(items[3])))

    tachX = integrate.cumtrapz(tachV, t)
    tachX = [x + potX[0] for x in tachX]

    accelV = integrate.cumtrapz(accelA, t)
    #accelV = [x + tachV[0] for x in accelV]
    accelV = [(accelV[x] + tachV[0])-6100*x/len(accelV) for x in range(len(accelV))] # rescaled
    accelX = integrate.cumtrapz(accelV, t[1:])
```

```
#accelX = [x + potX[0] for x in accelX]
accelX = [(accelX[x] + potX[0])-800*x/len(accelX) for x in range(len(accelX))] # rescaled

plt.clf()
plt.plot(t, potX, 'r', label='From potentiometer', alpha=0.5)
plt.plot(t[1:], tachX, 'b', label='From tachometer', alpha=0.5)
plt.plot(t[2:], accelX, 'g', label='From accelerometer', alpha=0.5)
plt.title('Position (mm) over time (s)), data from ' + name)
plt.ylabel('Position (mm)')
plt.xlabel('Time (s)')
plt.legend()
plt.savefig(dir_path + '\\' + 'partC2_rescaled_' + name + '.png')
```