MECH420 Lab 2

Ratthamnoon Prakitpong
#63205165

Part A: Characterization of the optical encoder signal

1.

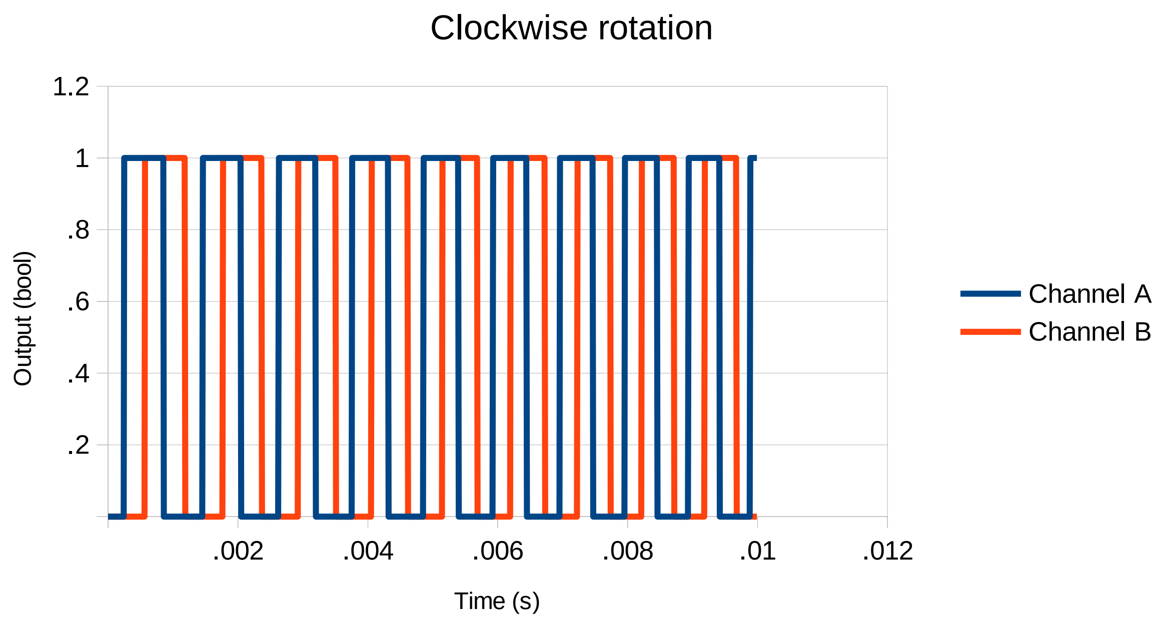


Fig A1: Encoder signal for clockwise rotation

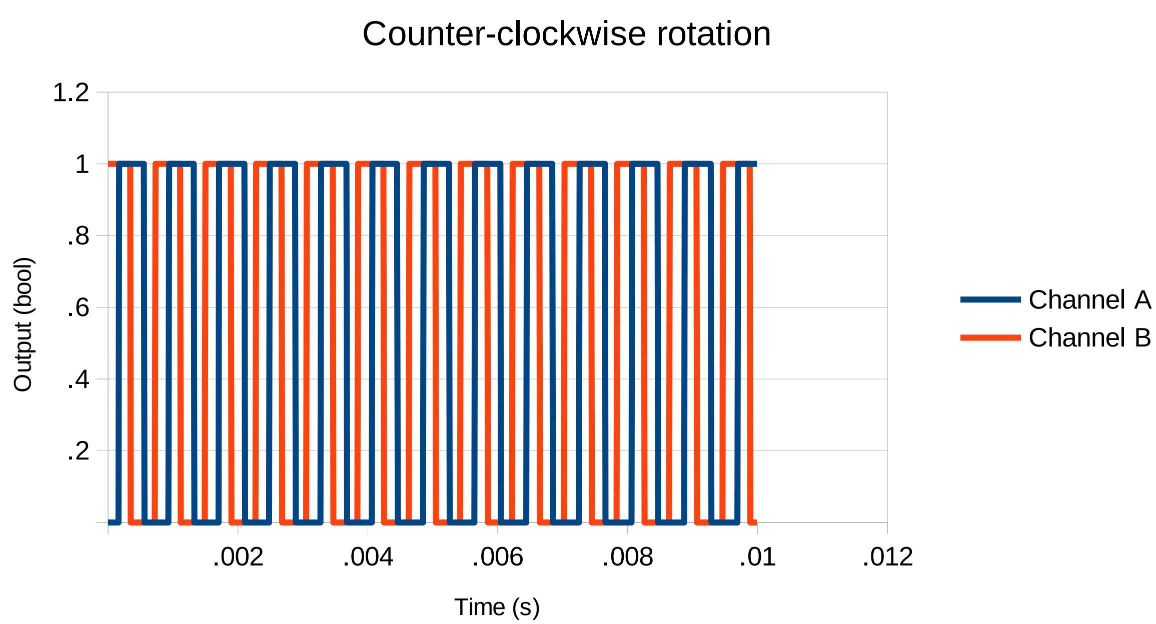


Fig A2: Encoder signal for counter-clockwise rotation

2.
For clockwise rotation, channel A leads by 90 degrees. For counter-clockwise rotation, channel B leads by 90 degrees.

3.

The encoder generates 360 pulses per revolution per channel (N = 360). The encoder generates quadrature signals. Note that we substitute N with 4N into following equations because signals are quadrature.

Pulse counting method:

I counted the pulses, which is n in the equations. The equations for angular speed and its resolution are:

$$\Omega_c = \frac{2\pi n}{NT} \qquad \Delta\Omega_c = \frac{2\pi}{NT}$$

The calculated values are:

c	n	Speed (rad/s)	Resolution (rad/s)
0.5V	39	68.06583333	1.74527777777778
0.7V	42	73.30166667	1.74527777777778
0.9V	43	75.04694444	1.74527777777778

Table A1: Angular speed and resolution, pulse counting method

This method provides low accuracy at low speed, which is the case here. This method's resolution is finer than pulse timing method's resolution.

Pulse timing method:

I counted and calculated average clock count of one pulse in a data set, which is m in the equations. Frequency is 40000 Hz (I see 40 counts in 0.0001s). The equations for angular speed and its resolution are:

$$\Omega_t = \frac{2\pi f}{Nm} \quad \Delta\Omega_t = \frac{2\pi f}{Nm(m+1)} \quad \Delta\Omega_t \approx \frac{N\Omega_t^2}{2\pi f}$$

The calculated values are:

c	m	Speed (rad/s)	Exact resolution (rad/s)	Approximate resolution (rad/s)
0.5V	25.86842105	67.46750311	2.51103341637525	2.60810286685568
0.7V	23.75609756	73.46651837	2.96761305716556	3.0925331139867
0.9V	23.70731707	73.61768404	2.97959037090301	3.10527268078677

Table A2: Angular speed and resolution, pulse timing method

This method is better for low speed. The approximate resolution is closer to exact resolution at lower speed.

Part B: Characterization of the Torque Sensor

1.

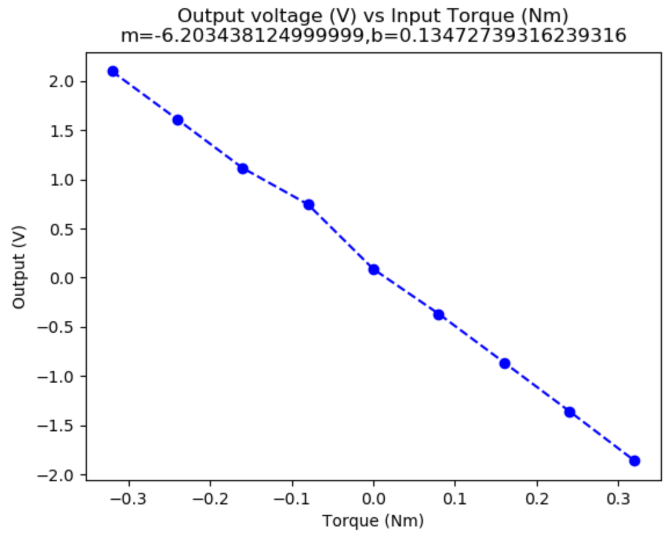


Fig B1: Voltage vs Torque, both cycles

Arbitrarily, I used left weights as negative torque, so the plot resulted in a negative slope.

2.

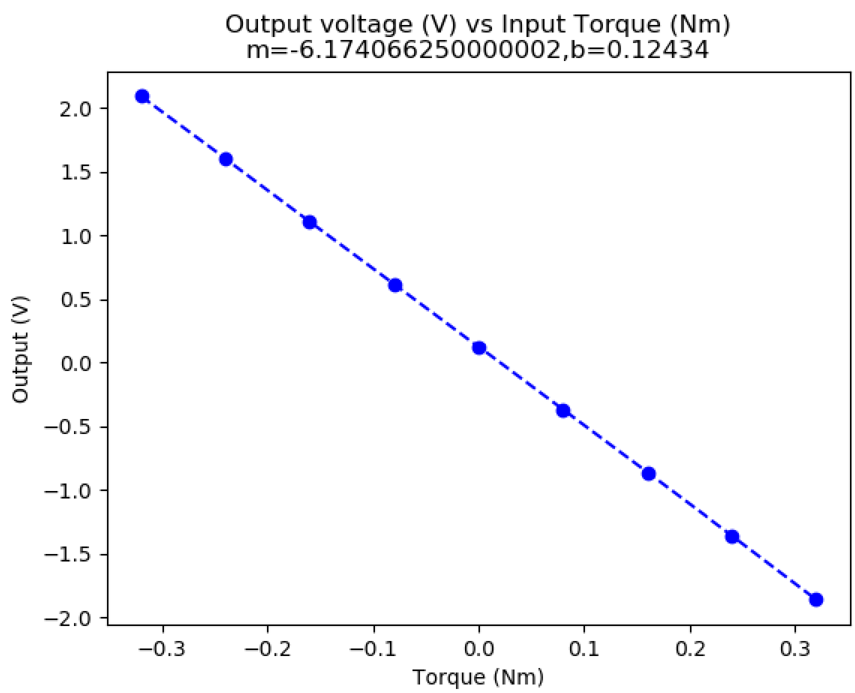


Fig B2: Voltage vs Torque, second load cycles only

Calibration equation is taken from the plot above (numpy.polyfit uses least square method):

$$V(T) = b + mT,\ m = -6.174\ \text{V/Nm},\ b = V0 = 0.124\ \text{V}$$

3.
We get nonlinear error with this equation:

$$\% \text{ Error} = \left| \frac{\text{Theoretical Value} - \text{Experimental Value}}{\text{Theoretical Value}} \right| \text{X } 100$$

Where theoretical value is value from regression, and experimental value is from averaging values.

We get hysteresis error with this equation:

$$H = \left(\frac{I_{UP} - I_D}{I_{MAX}} \right) \text{x } 100\%$$

Where I is the value we’re finding error of, and H is the percent error. Note that we also have to take absolute value of H to get satisfactory percent error.

We use average values for each values in the equations for calculations. Plotted, we get:

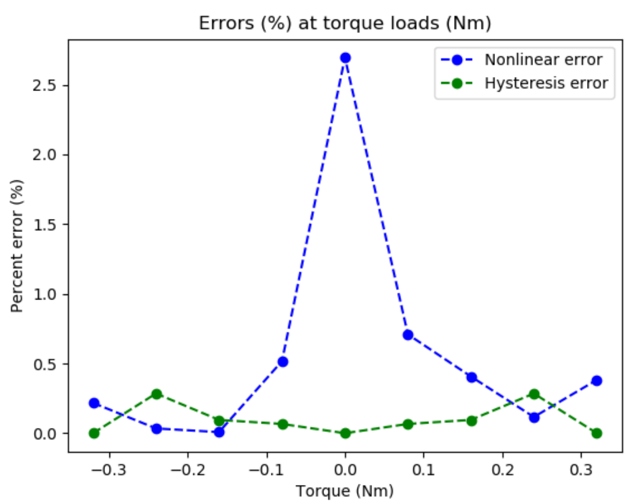


Fig B3: Percent errors at torque loads

4.
From the product datasheet I found online:

SPECIFICATIONS

Nominal Capacity 3kg ~ 250kg
Signal Output at Capacity 2mV/V ± 10%
Linearity Error < 0.020% FSO
Non-Repeatability < 0.010% FSO
Combined Error < 0.025% FSO
Hysteresis < 0.015% FSO
Creep/Zero Return (30 mins) < 0.030% / 0.020% FSO
Zero Balance < 3.000% Capacity
Temperature Effect On Span/10°C < 0.010% FSO
Temperature Effect on Zero/10°C < 0.015% Capacity
Compensated Temperature Range - 10 ~ 40°C
Operating Temperature Range - 30 ~ 70°C
Service Load 100% of Rated Capacity
Safe Load 150% of Rate Capacity
Ultimate Load 300% of Rated Capacity
Excitation +ve RED
Signal +ve GREEN

Input Impedance 425Ω ± 15Ω
Output Impedance 350Ω ± 3Ω
Insulation Impedance >5000 MΩ at 100V DC
Excitation Voltage (Rec) 5 ~ 12V AC/DC
Excitation Voltage (Max) 15V AC/DC
Eccentric Loading (effect/cm) < 0.0085% FSO (3 ~ 35kg), < 0.0074% FSO (50 ~ 250kg)
Deflection at Rated Capacity < 0.4mm
Storage Temperature Range -50 ~ 70°C
Cable Type 4mm, Screened, PVC Sheath 4-core x 0.09mm² (28 AWG)
Cable Length 0.5 Metre (3kg ~ 35kg), 1 Metre (50kg ~ 250kg)
Material Aluminium
Finish Marine Anodised
Excitation -ve BLACK
Signal -ve WHITE

Theoretical calculations are shown in Part 5.2.

Sensitivity:
Theoretical:
Sensitivity = 0.728 mV/Nm

Experimental:
Slope from fig B2 = 6.174 V/Nm
Amplifier = 100
Sensitivity without gain from amplifier = 6.174 V/Nm / 100 = 61.74 mV/Nm

This is very suspect. Reason could be analysis error, or measurement error. For example, the unit for torque was given as Nm. However, the numbers felt suspiciously like they were supposed to be kgm (600 grams * 400 mm = 0.6 kg * 0.4 m = 0.24 kgm != 0.24 kgmm/(ss)). Not accounting for gravity can cause recorded torque values to be smaller than they actually are, hence larger sensitivity.

Offset:
Theoretical:
Offset = 4 mV

Experimental:
Y-intercept from fig B2 = 0.124 V
Amplifier = 100
Offset = 0.124V / 100 = 1.24 mV

It makes sense that we have some offset, but did not hit the maximum of how far off our data could be.

Absolute errors:
Theoretical:
Linearity error = 0.004 mV
Hysteresis error = 0.003 mV

Experimental:
We get these values from plot, multiplying percent error by experimental values.
Average most linearity error = 0.00295 mV
Average hysteresis error = 0.00109 mV

The theoretical value is in range of our theoretical calculations.

Part C: Measurement of the Torque Required to Drive the Conveyor

1.
Given that the belt ran two full cycles and that we want to plot one full cycle, we can just plot half the given data. Average torque and its standard deviation are included in the plot.

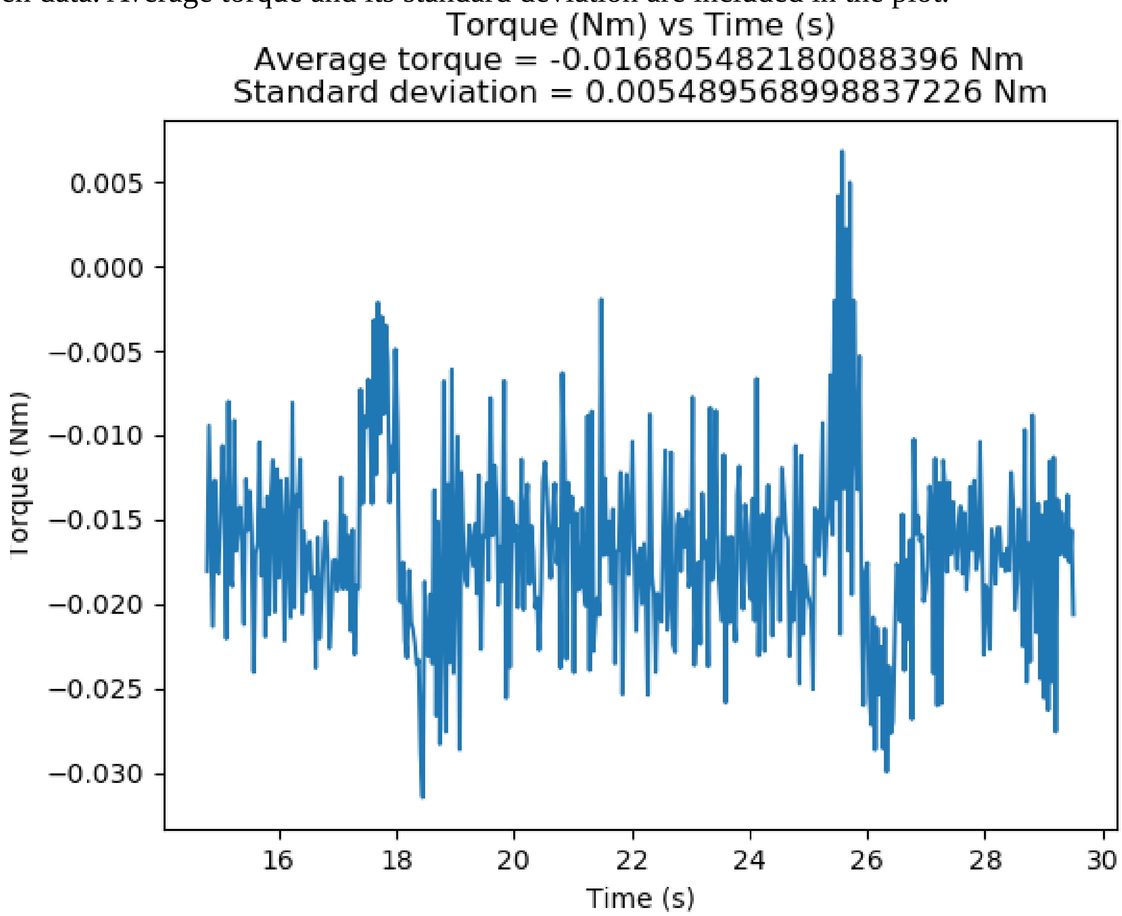


Fig C1: Torque over one full cycle of conveyor belt

5. Additional Exercises

1.

$N = 360$

$f = 400\text{Hz}$

$T = 2.5\text{ms} = \frac{1}{f}$

$\Delta\Omega_c = \Delta\Omega_t$

$\frac{2\pi}{4NT} = \frac{4N\,\Omega_t^2}{2\pi f}$

$\Omega = \sqrt{\frac{4\pi^2 f^2}{16N^2}}$

$= \frac{2\pi f}{4N}$

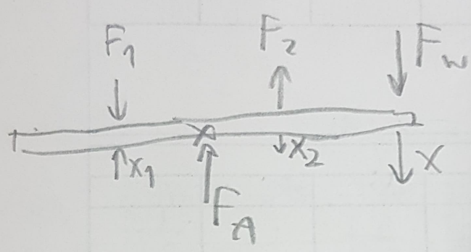
$= \frac{2\pi\, 400\text{ Hz}}{360(4)}$

$= 1.74\text{ rad/s}$

Speed is 1.74 rad/s.

2.

We can think of load cells as springs (resist change in direction).



$\Sigma M_A = 0 = F_W L_W - (F_1 + F_2) L_L$

say $-F_1 = F_2$ because symmetry

$F_1 = \frac{F_W L_W}{2L_L} = \frac{T_{in}}{2L_L}$

$F_2 = \frac{-F_W L_W}{2L_L} = \frac{-T_{in}}{2L_L}$

$V_1 = B_1 F_1 L_L + V_{10}$

$V_2 = B_2 F_2 L_L + V_{20}$

$\Delta V_1 = V_{1\,measured} - V_{1\,theory}$

$\Delta V_2 = V_{2\,measured} - V_{2\,theory}$

$V_{out} = V_1 - V_2$

$= BL_L(F_1 - F_2)$

$= 2BL_L F_1$

$= 2BL_L\left(\frac{T_{in}}{2L_L}\right)$

$S = B = \frac{V_{out}}{F_W L_W}$

$S = \frac{2mV/V\,(10V)}{7kg\,(9.81\,m/s^2).400m}$

$= 0.728\ mV/Nm$

Signal output is $\pm 10\%$

$\Delta V = \Delta V_1 - \Delta V_2$

$\max(\Delta V) = (\Delta V_1 + 10\%) - (\Delta V_2 - 10\%)$

$\Delta V_{max} = 20\% \left(\frac{2mV}{V}\right) 10V$

$= 4mV$

Linearity error $= \frac{2mV}{V}(10V)\,0.02\%$

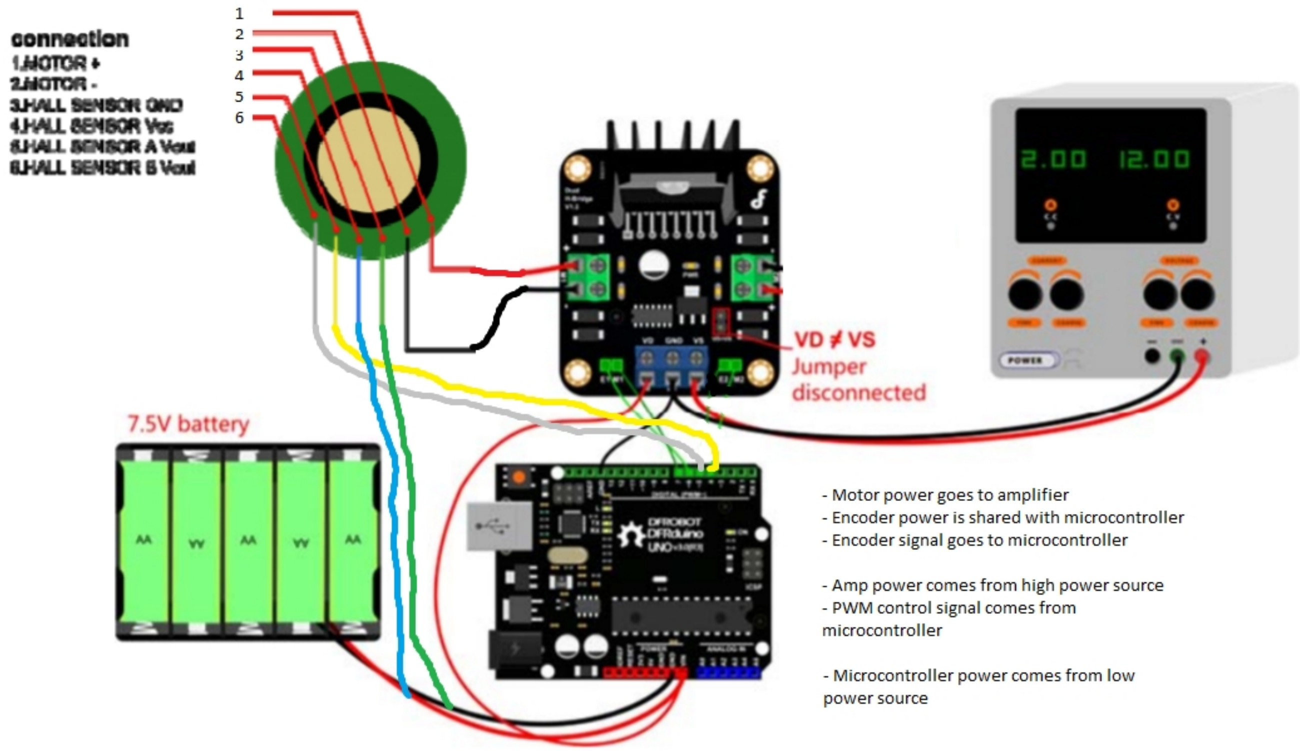
$= 0.004\ mV$

Hysteresis error $= \frac{2mV}{V}(10V)\,0.015\%$

$= 0.003mV$

3.

Part	Datasheet
DC motor with encoder	https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/FIT0186_Web.pdf
Microcontroller	https://www.farnell.com/datasheets/1682209.pdf
PWM Amp	https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DRI0002_Web.pdf



Appendix

e1-pulsecount-get_n.py

```
import os
dir_path = os.path.dirname(os.path.realpath(__file__))
print(dir_path)

files = ['e1_motor_0.5V.csv', 'e1_motor_0.7V.csv', 'e1_motor_0.9V.csv']

for name in files:
    with open(dir_path+'\\'+name) as f:
        counter = 0
        flag = False
        for line in f:
            items = line.split(',')
            if len(items) > 1:
                if items[1] == '1' and flag == False:
                    flag = True
                    counter = counter + 1
                if items[1] == '' and flag == True:
                    flag = False
        print(name)
        print(counter)
```

e1-pulsetiming-get_m.py

```
import os
dir_path = os.path.dirname(os.path.realpath(__file__))
print(dir_path)

files = ['e1_motor_0.5V.csv', 'e1_motor_0.7V.csv', 'e1_motor_0.9V.csv']

for name in files:
    with open(dir_path+'\\'+name) as f:
        counts = []
        counter = 0
        prev = ''
        for line in f:
            items = line.split(',')
            if len(items) > 1:
                if items[1] == '1' and prev == '':
                    counts.append(counter)
                    counter = 0
                counter = counter + 1
                prev = items[1]
        print(name)
        print(sum(counts[1:len(counts)])/(len(counts)-1))
```

e2.py

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

def plotter(files_right_, files_left, savename=None):
    t_08 = [] # underscore == negative == left weights
    t_16 = []
    t_24 = []
    t_32 = []
    t00 = []
    t08 = []
    t16 = []
    t24 = []
    t32 = []
    for name in files_right_:
        with open(dir_path+'\\'+name) as f:
            for line in f:
                items = line.split(',')
                if len(items) > 1:
                    if items[1] == '':
                        t00.append(float(items[2]) - float(items[3]))
                    if items[1] == '.08':
                        t08.append(float(items[2]) - float(items[3]))
                    if items[1] == '.16':
                        t16.append(float(items[2]) - float(items[3]))
                    if items[1] == '.24':
                        t24.append(float(items[2]) - float(items[3]))
                    if items[1] == '.32':
                        t32.append(float(items[2]) - float(items[3]))

    for name in files_left:
        with open(dir_path+'\\'+name) as f:
            for line in f:
                items = line.split(',')
                if len(items) > 1:
                    if items[1] == '':
                        t00.append(float(items[2]) - float(items[3]))
                    if items[1] == '.08':
                        t_08.append(float(items[2]) - float(items[3]))
                    if items[1] == '.16':
                        t_16.append(float(items[2]) - float(items[3]))
                    if items[1] == '.24':
                        t_24.append(float(items[2]) - float(items[3]))
                    if items[1] == '.32':
                        t_32.append(float(items[2]) - float(items[3]))
```

```
    x = [-0.32, -0.24, -0.16, -0.08, 0, 0.08, 0.16, 0.24, 0.32]
    y = [sum(n)/len(n) for n in [t_32, t_24, t_16, t_08, t00, t08, t16, t24, t32]]
    m, b = np.polyfit(x, y, 1)

    if savename:
        plt.clf()
        plt.plot(x, y, 'bo--')
        plt.title('Output voltage (V) vs Input Torque (Nm)\nm=' + str(m) + ',b=' + str(b))
        plt.ylabel('Output (V)')
        plt.xlabel('Torque (Nm)')
        plt.savefig(dir_path + '\\' + savename + '.png')

    return y

def getHysteresisError(filename):
    firstPass = [[], [], []]
    secondPass = [[], [], []]
    firstPassDone = False
    for name in files_right_:
        with open(dir_path+'\\'+name) as f:
            for line in f:
                items = line.split(',')
                if len(items) > 1:
                    ind = 3
                    if items[1] == '.08':
                        ind = 0
                    if items[1] == '.16':
                        ind = 1
                    if items[1] == '.24':
                        ind = 2
                    if items[1] == '.32':
                        firstPassDone = True

                    if ind < 3:
                        val = float(items[2]) - float(items[3])
                        if firstPassDone:
                            secondPass[ind].append(val)
                        else:
                            firstPass[ind].append(val)
    avgFirstPass = [sum(n)/len(n) for n in firstPass]
    avgSecondPass = [sum(n)/len(n) for n in secondPass]
    #print(sum([abs(x) for x in [(f - s) for f, s in zip(avgFirstPass, avgSecondPass)]])/len(avgFirstPass))
# for part B4
    # take the two avg error, times each by 3, sum them, divide by 9 to get total avg
    error = [100 * (f - s) / max(f, s) for f, s in zip(avgFirstPass, avgSecondPass)]
    error = [abs(n) for n in error]
    return error
```

```
# 1
files_right_ = ['e2_torque_0.csv', 'e2_torque_one_by_one_right.csv',
'e2_torque_one_by_one_right_2.csv']
files_left = ['e2_torque_one_by_one_left.csv', 'e2_torque_one_by_one_left_2.csv']
plotter(files_right_, files_left, savename='e2-1')

#2
files_right_ = ['e2_torque_one_by_one_right_2.csv']
files_left = ['e2_torque_one_by_one_left_2.csv']
plotter(files_right_, files_left, savename='e2-2')

#3
files_right_ = ['e2_torque_one_by_one_right_2.csv']
files_left = ['e2_torque_one_by_one_left_2.csv']

x = [-0.32, -0.24, -0.16, -0.08, 0, 0.08, 0.16, 0.24, 0.32]
m = -6.174
b = .124
theory = [b + m*n for n in x]
real = plotter(files_right_, files_left)
print(sum([abs(x) for x in [(t - r) for t, r in zip(theory, real)]])/len(theory)) #for part B4
nonlinearError = [100 * (t - r) / t for t, r in zip(theory, real)]
nonlinearError = [abs(a) for a in nonlinearError]

e_right = getHysteresisError(files_right_)
e_left = getHysteresisError(files_left)
e_left.reverse()
hysteresisError = [0] + e_left + [0] + e_right + [0] # 0s represent no hysteresis error at 0 and .32 Nm,
where they are end points

plt.clf()
plt.plot(x, nonlinearError, 'bo--', label='Nonlinear error')
plt.plot(x, hysteresisError, 'go--', label='Hysteresis error')
plt.title('Errors (%) at torque loads (Nm)')
plt.ylabel('Percent error (%)')
plt.xlabel('Torque (Nm)')
plt.legend()
plt.savefig(dir_path + '\\' + 'e2-3.png')
```

e3.py

```
import os
import matplotlib.pyplot as plt
import numpy as np
dir_path = os.path.dirname(os.path.realpath(__file__))
print(dir_path)
name = 'e3_50RPM.csv'
```

```
def file_len(fname):
    with open(dir_path+'\\'+fname) as f:
        for i, l in enumerate(f):
            pass
    return i + 1

filelen = file_len(name)
t = []
rpm = []
cell0 = []
cell1 = []
with open(dir_path+'\\'+name) as f:
    next(f) # discard first line
    counter = 0 # to flush out initial data
    startSaving = False
    initTime = 0
    for line in f:
        items = line.split(',')

        counter = counter + 1
        if counter > filelen / 2:
            t.append(float(items[0]))
            rpm.append(float(items[1]))
            cell0.append(float(items[2]))
            cell1.append(float(items[3]))

m = -6.174
b = .124
T = [(c0 - c1 - b)/m for c0, c1 in zip(cell0, cell1)]
avgT = np.mean(T)
stdT = np.std(T)

plt.clf()
plt.plot(t, T)
plt.title('Torque (Nm) vs Time (s)\nAverage torque = ' + str(avgT) + ' Nm\nStandard deviation = ' +
str(stdT) + ' Nm')
plt.ylabel('Torque (Nm)')
plt.xlabel('Time (s)')
plt.savefig(dir_path + '\\' + 'e3.png')
```