Investigating Torricelli's law using a pressure sensor

Jamie Lee Somers, 19330931
B.Sc in Applied Physics.

Wednesday 14th April, 2021

1 Introduction:

In 1643, after succeeding Galileo Galilei as professor of mathematics at the Florentine Academy[1]. Evangelista Torricelli, among other experiments, discovered a law now known as Torricelli's Law. The crux of this law is that water jets our of a small hole at the bottom of a container filled to a particular height h proportional to the speed it would be free falling following the equation $v = \sqrt{2gh}$ where h is the aforementioned height of the liquid in the container. This jetting is known as the efflux speed.

Usually when people discuss Torricelli's Law they are referring strictly to water flowing through a small hole in a container, however Torricelli's law is not necessarily limited to just water. When attempting to verify Torricelli's law the only limitations for the law is that there is minimal air resistance, the liquids viscosity is extremely low and the hole at the bottom of the container is far smaller proportional to the hole opening at the top of the container[2].

In this experiment more sophisticated equipment than Evangelista Torricelli would have had available to him at the time of discovery was used, in an effort to verify or disprove Torricelli's law. Discussed below is the findings of four separate liquids, whether or not they fall within the limitations set out by Torricelli's law and if the results line up with what Torricelli's law predicts.

Using an Arduino, a pressure sensor and an ultrasonic sensor, large amounts of data points can be obtained which measure the height of the liquid in a container at any given time. Liquids which behave according to Torricelli's law when graphed will display as a quadratic equation where the decrease in height gradually becomes slower and slower until there is a long tail leading to a height of zero. This is because the jet or the velocity of the water leaving the hole will decrease as the height of water above the hole decreases.

Shown below is two graphs, one graph represents height vs time for water which gives an ideal representation of Torricelli's law on a graph. The second graph was obtained using honey. Honey is an incredibly viscous liquid which as explained above, does not obey Torricelli's law.

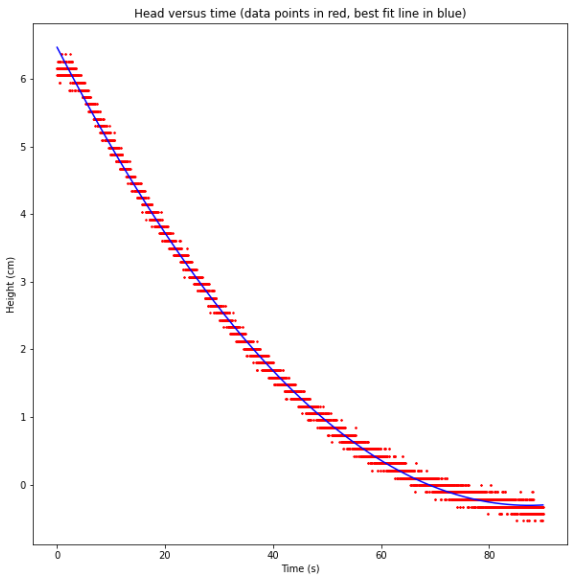

Figure 1: Height versus time (water)

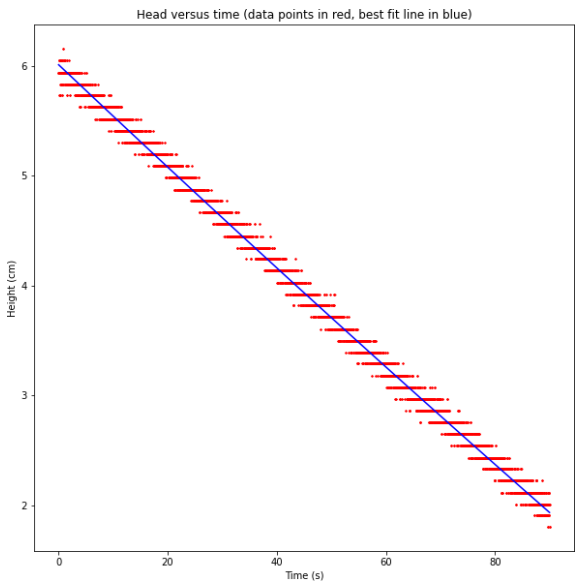

Figure 2: Height versus time (honey)

2 Project Objectives:

- Set up Arduino system which can measure the change in liquid head over time.
- Import the Arduino code into Python using PyFirmata to take advantage of modules such as numpy and matplotlib
- Verify that Torricelli's law holds true for water.
- Explore Torricelli's law using alternate liquids and viscosities.
- Build a GUI which improves the systems real world usability for applications such as automatic drainage mentioned in the project proposal.
- Find the quadratic equation for each liquid using the least squared fitting python code introduced previously.

3 Experimental Setup and Methods:

The general experimental setup used to investigate Torricelli's law involves a container with two holes, one large hole at the top to allow liquid to be stored inside and one smaller hole at the base of the container which is filled with a tube that has a stopper which can prevent the draining of liquid.

There also needs to be some method of measuring the height of the liquid still in the container. An Arduino was utilised for this purpose as it is modular and has many different types of sensors which could be added to the system. The two sensors involves were the MPX5010GP Pressure sensor and the HC-SR04 ultrasonic sensor.

The pressure sensor can calculate the height of the liquid by taking the pressure in the room and comparing it to the pressure where the sensor is located below the surface of the water. This difference in pressures generates a voltage which the Arduino can print to the screen, using a formula involving the voltage reading (V), the initial voltage when the container is drained (V0) and a beta value which converts the voltage reading to a comprehensible height in cm (β):

$$h = \frac{V - V0}{\beta} \tag{1}$$

When the container is fully drained the voltage reading isn't necessarily zero. We can see from the equation that our V0 value depends on whatever the voltage reading obtained is whenever the h value is 0. The β value is found by dividing the voltage difference by whatever the measured height of the liquid is at a given time.

For the purposes of this experiment, the values were $\beta = 0.045$ and $V0 = 0.22$

The ultrasonic sensor determines the height of the liquid using sonar with high accuracy. The sensor emits an ultrasonic wave which travels to the surface of the water, once it hits the surface the wave bounces back as an 'echo'. The distance is calculated based on the amount of time it takes the wave to hit the surface and the speed of sound.

The height of the liquid in the container h, is found by subtracting the distance from the sensor to the surface of the water when the container is full from the reading. This way when the container is full, the reading is zero, and as the water starts to drain the distance being shown is how much of the water has been drained so far.

To calculate the distance using an ultrasonic sensor an external guide was used which had a comprehensive explanation for the technique involved[3]. The calculation was the speed of sound in cm/μs, (0.034) divided by two with the distance between the sensor and the surface of the water (2.5 cm) subtracted as a function of time.

$$s = \frac{0.034 \cdot t}{2} - 2.5 \tag{2}$$

As the time taken for the sound wave to reach the sensor increases, the top part of the fraction gets larger and the distance value grows bigger which is the height the H value has lost since the measurements began.

Some minor adjustments were made to the drain tube, the para-film was switched out in favour of hot glue which as well as reducing the number of leaks occurring from the container also increased the rigidity of the drain tube itself and was able stay perfectly horizontal during the readings without the need of a support.

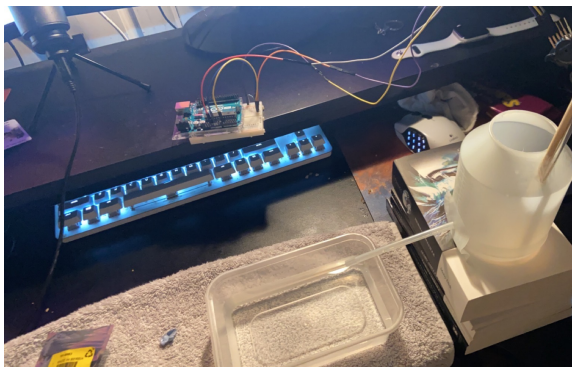

Figure 3: Apparatus with para-film & unsupported drain pipe

Figure 4: Apparatus with hot glue & supported drain tube

4 Data Presentation and Analysis

Water:

The experiment was carried out using water three times, all three times the data remained consistent with the A, B and C values for water being A $= 8.86 \times 10^{-4} \pm 8.68 \times 10^{-7}$, B $= -1.55 \times 10^{-1} \pm 8.17 \times 10^{-5}$ and C $= 6.46 \pm 1.6 \times 10^{-3}$. This produces a quadratic which displays the exact characteristics we would expect to see from a liquid obeying Torricelli's law (Fig. 5). We also measured the efflux value for water which showed an initial flow rate of 118 cm/s^{-1} which decreased linearly over time (Fig. 6). This is exactly the behaviour we expect to see.

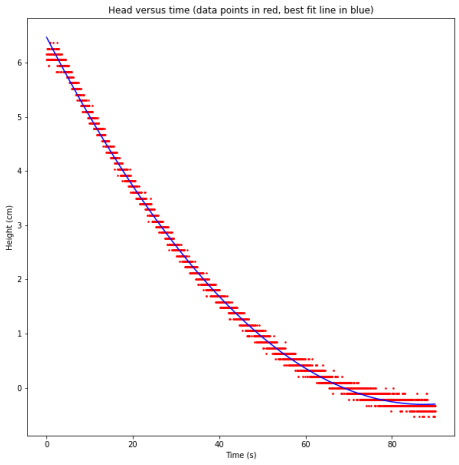

Figure 5: Height versus time (water)

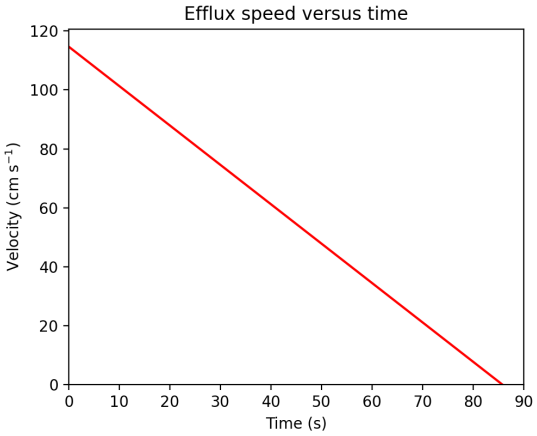

Figure 6: Efflux speed versus time (water)

Washing-up liquid:

The results for washing-up liquid were very similar to the values seen for water. The A,B and C values were A $= 9.41 \times 10^{-4} \pm 7.78 \times 10^{-7}$, B $= -1.59 \times 10^{-1} \pm 7.37 \times 10^{-5}$ and C $= 6.37 \pm 1.46 \times 10^{-3}$ The efflux value decreased in a linear pattern, however the washing-up liquid took less time to drain than the water.

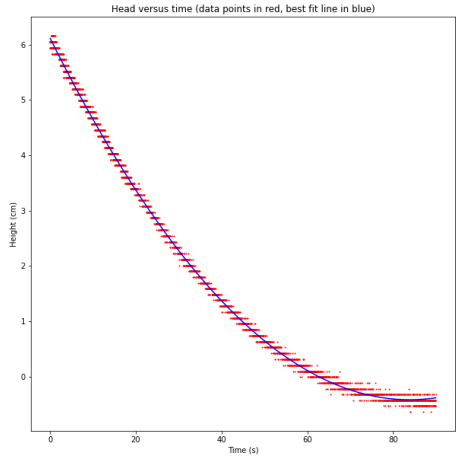

Figure 7: Height versus time (washing-up)

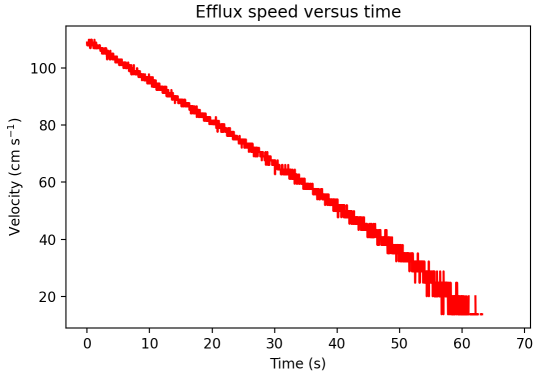

Figure 8: Efflux speed versus time (washingup)

Carbonated Soda:

This is the most interesting of the liquids tested. In the first run the carbonated liquid behaved nothing like a typical liquid obeying Torricelli's Law but by the third run it was obeying it perfectly, this is reflected in both the A,B and C values for each run as well as the height vs time graphs (Figure 9, 10 and 11)

Run 1: $A = 2.76 \times 10^{-4} \pm 2.01 \times 10^{-6}$, $B = -5.99 \times 10^{-2} \pm 1.98 \times 10^{-4}$ and $C = 6.28 \pm 4.15 \times 10^{-3}$

Run 2: $A = 6.04 \times 10^{-4} \pm 1.57 \times 10^{-6}$, $B = -1.09 \times 10^{-1} \pm 1.48 \times 10^{-4}$ and $C = 6.39 \pm 2.80 \times 10^{-3}$

Run 3: $A = 7.54 \times 10^{-4} \pm 1.39 \times 10^{-6}$, $B = -1.33 \times 10^{-1} \pm 1.21 \times 10^{-4}$ and $C = 6.46 \pm 1.88 \times 10^{-3}$

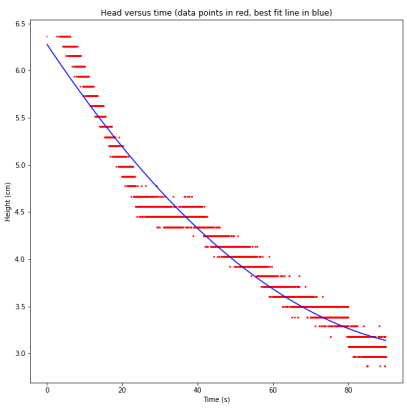

Figure 9: h vs t (Run 1)

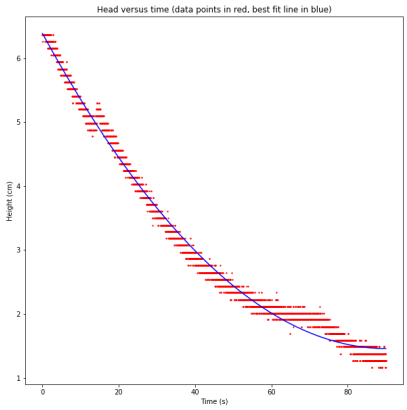

Figure 10: h vs t (Run 2)

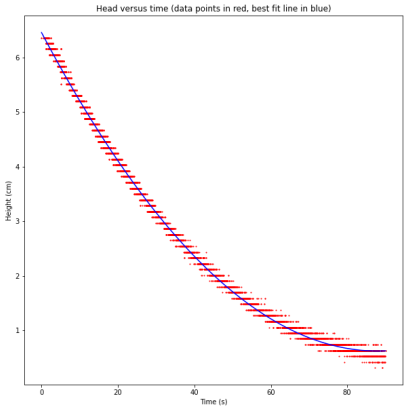

Figure 11: h vs t (Run 3)

It appears as though something has changed during each run which somehow makes the liquid more accurate to what Torricelli's Law predicts.

Honey:

Honey is an extremely viscous liquid, this means that unlike other liquids honey has a high resistance to molecules sliding over each other and flowing like a typical liquid. This is due to the fact that honey is chemically made up of sugars such as glucose and fructose which are bonded using Hydrogen bonds[4]. The results for honey are as follows: $A = 1.90 \times 10^{-5} \pm 7.99 \times 10^{-7}$, $B = -4.70 \times 10^{-2} \pm 7.61 \times 10^{-5}$ and $C = 6.01 \pm 1.51 \times 10^{-3}$

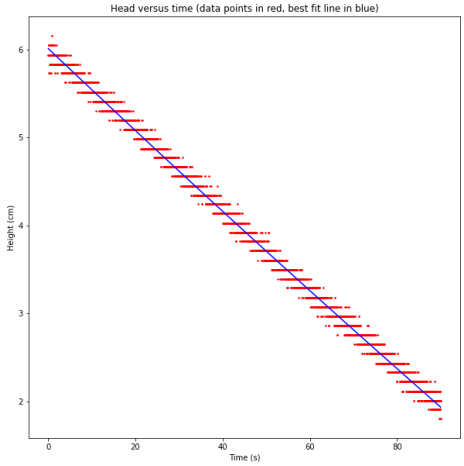

Figure 12: Height versus time (honey)

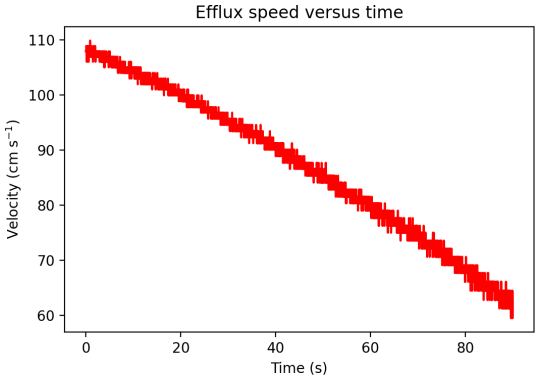

Figure 13: Efflux speed versus time (honey)

5 Discussion of Results

Thanks to equation 6 in the "Investigating Torricelli law PhysEd 2018" paper[5], it was possible to work out the values for A, B and C mathematically and compare the answer to the answers obtained using the least squared fitting method. All three runs of the water and washing-up liquid tests fell within the uncertainty indicated in the results section, however only the third run of the carbonated soda run fell within its calculated uncertainty and none of the honey runs matched up with their calculated A,B and C values.

A possible hypothesis as to why the washing-up liquid water drains faster than regular water is that the suds in the mixture help spread out the water molecules. This gives the water molecules the opposite effect of honey and its hydrogen bonds, reducing the resistance to the molecules sliding over each other.

A possible hypothesis as to why the carbonated soda behaved more and more inline with Torricelli's Law as the experiment was ran more times is that, carbonated soda is an example of a homogeneous mixture of liquid and gas (carbon dioxide). Once the soda can is opened the carbon dioxide begins to escape and the soda becomes increasingly "flat". Torricelli's Law is specifically stated as a law that applies to non-viscous liquids, it is assumed that a liquid and gas mixture will not obey Torricelli's law until the gas has been extracted from the liquid.

According to Figure 13, the efflux velocity for Honey was extremely high compared to the other liquids tested. Its important to remember that Torricelli's law is strictly limited to describing non-viscous liquids, this graph is a perfect example of why Torricelli's law does not effectively predict the characteristics of viscous liquids. The method of calculating the efflux velocity is by assuming that a liquid pouring out of a container with a higher 'h' value is going to flow faster than a liquid with a smaller 'h' value. This was not true in the case of honey, the reason why the 'h' value for honey was so high and remained that way was because they honey was actually flowing extremely slowly and didn't even drain completely in the 90 seconds allocated for measurements. It is then assumed that this graphs velocity predictions are completely inaccurate.

6 Conclusions and Potential Improvements

Torricelli's law held true for Water:
$A = 8.86 \times 10^{-4} \pm 8.68 \times 10^{-7}$, $B = -1.55 \times 10^{-1} \pm 8.17 \times 10^{-5}$ and $C = 6.46 \pm 1.6 \times 10^{-3}$,
Washing-up liquid:
$A = 9.41 \times 10^{-4} \pm 7.78 \times 10^{-7}$, $B = -1.59 \times 10^{-1} \pm 7.37 \times 10^{-5}$ and $C = 6.37 \pm 1.46 \times 10^{-3}$
and Run 3 of Carbonated Soda:
$A = 7.54 \times 10^{-4} \pm 1.39 \times 10^{-6}$, $B = -1.33 \times 10^{-1} \pm 1.21 \times 10^{-4}$ and $C = 6.46 \pm 1.88 \times 10^{-3}$.
All other results fall outside of the parameters of what Torricelli's Law predicts, therefore, Torricelli's Law remains verifiably true.

Potential improvements which could be made to the experiment in future:

Now knowing that certain liquids such as Carbonated Soda have a time constraint around noticing changes in its behaviour, I would have worked faster to do more runs in a shorter amount of time. This would either back up or falsify the hypothesis that the liquids strange behaviour is based on its chemical composition. Once the effect was observed more cans of carbonated soda were used directly after being opened and once they had gone flat to test the hypothesis. These measurements remained consistent with the ones included in this report.

Placing the entire apparatus, particularly the drain tube stopper on a shock absorbent mount would help reduce the unreliably nature of the ultrasonic sensor. During multiple test runs, removing the drain tube stopper with anything but the lightest touch caused the liquid inside the container to oscillate at the surface which effected the sensors ability to accurately detect the height of the liquid. This irregularity in the readings diminishes as the water settles back to equilibrium at the surface of the container. This effect was minimised by placing two towels below the container and also coding in more time to release the drain tube before measurements are recorded to ensure there was no need to act quickly.

Ideally a method of producing Efflux velocity graphs would have been included within the GUI, however time constraints meant that this was no longer a possibility. In future the GUI should be designed to produce all of the plots necessary for the project report without the need for external code. Instead an external python programme was made with the explicit intent to produce efflux plots and save them to a file.

References:

[1] Britannica, The Editors of Encyclopaedia. "Evangelista Torricelli".Encyclopedia Britannica, 21 Oct. 2020, https://www.britannica.com/biography/Evangelista-Torricelli

[2] Princeton University, Johann Otto. "Torricelli's Law for Large Holes". 15 Sep. 2018, https://www.physics.princeton.edu/~mcdonald/examples/leaky_tank.pdf

[3] HowToMechatronics, Dejan. "Ultrasonic Sensor HC-SR04 and Arduino Tut.".26 Jul. 2015, https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

[4] Moore, Justin Shorb, Xavier Prat-Resina, Tim Wendorff, Ed Vitz, John W., and Adam Hahn. "Viscosity." Chemical Education Digital Library, 5 Nov. 2020, https://chem.libretexts.org/@go/page/49661.

[5] Atkin, Keith. "Investigating the Torricelli law using a pressure sensor with the Arduino and MakerPlot." Physics Education, Aug. 2018, https://doi.org/10.1088/1361-6552/aad680

Appendix:

Torricelli's Law, Using Arduino pressure sensor, Python code and PyFirmata

```
# -*- coding: utf-8 -*-
"""
Created on Sat Mar 27 19:24:49 2021

@author: JamieSomers
References:
General UI Template from: https://pythonprogramming.net/change-show-new-frame-
    tkinter/
Creating a file reader button: https://stackoverflow.com/questions/16798937/creating
    -a-browse-button-with-tkinter
How to open a file based on filepath: https://stackoverflow.com/questions/41443600/
    get-name-attribute-of-io-bufferedreader
How to open file explorer: #https://stackoverflow.com/questions/281888/open-explorer
    -on-a-file
Positioning multiple buttons: #https://stackoverflow.com/questions/51631105/how-to-
    position-several-widgets-side-by-side-on-one-line-with-tkinter
Text box info: https://www.geeksforgeeks.org/python-tkinter-text-widget/
3D plot: https://matplotlib.org/2.0.2/mpl_toolkits/mplot3d/tutorial.html
PyFirmata: https://pypi.org/project/pyFirmata/
Change variable value based on user input: https://stackoverflow.com/questions
    /24911805/change-the-value-of-a-variable-with-a-button-tkinter
"""


import numpy as np #Imports the library known as numpy and changes its name to np
import matplotlib #Imports matplotlib
matplotlib.use("TkAgg") #Allows matplotlib to be used inside tkinter
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
    NavigationToolbar2Tk #Allows for graphs to be displayed on the UI
from matplotlib.figure import Figure #Allows matplotlib to use figures
import matplotlib.pyplot as plt  #Imports MatPlotLib which is a comprehensive
    library for creating static, animated and interactive visualizations in Python
    like creating plots, and changes its name to plt
from matplotlib import pyplot

from mpl_toolkits.mplot3d import Axes3D

import scipy as sp #Imports scipy as sp
from scipy import optimize #allows us to use sp.optimize which is important for the
    least squared fitting window

import tkinter as tk #Imports a UI building tool called tkinter as tk
from tkinter import ttk #allows us to use tk themed widgets
from tkinter import * #Imports every exposed object in Tkinter into namespace
from tkinter.ttk import * #Allows us to import things like labels and buttons
from tkinter import filedialog #Allows us to open the file explorer window on a
    windows machine inside our UI
```

```
import csv #Used to create and edit our .csv files
import time #Allows us to tell the code to sleep for a second (required to read
    input values from Arduino)
import ctypes, os #Allows us to use the internal QPC clock for our milis() function
import threading #Threading functions is required for this code as the UI will
    become unresponsive when gathering measurements otherwise

import subprocess #Subprocess allows us to open the file explorer window on a
    windows machine and locate a specific file

try:
    from pyfirmata import Arduino, util #Try using PyFirmata to import Arduino
except:
    import pip
    pip.main(['install','pyfirmata']) #If PyFirmata is not installed, install it
    from pyfirmata import Arduino, util #imports Arduino and util from pyfirmata


LARGE_FONT= ("Verdana", 12) #This is the font style and size that is used within the
     UI

settings = {} #This is a dictionary that stores the settings menu until it is
    written to a file
analysis_data = {} #This is a dictionary that stores A,B,C guesses for least squared
     fitting until it is written to a file

# Mahalanobis distance variables
    ----------------------------------------------------------------------------------------------------


distances = [] #This list stores the mahalanobis distances so I can determine which
    is lowest
Vector_database = np.array([[0.0008864714983003059,
    -0.15494893946363458,6.464166964037915],[0.0008705378877068795,
    -0.15518994683160206,6.721225367081532],#two water vectors
                            [0.0009187352876899168,
                                -0.15955350486484704,6.537786655059696],[0.0009413558375836622,
                                 -0.1598663366460094,6.374893163429263],#two washing
                                -up liquid vectors
                            [0.0009278046644047776,
                                -0.05991894711853976,6.280040401911863],[0.0006035277020619921,
                                 -0.10905845490923652,6.385419084924521],#two
                                carbonated vectors
                            [1.8960545786506496e-5,
                                -0.04700306935841803,6.011695946893561],[0.0004910783655506937,
                                 -0.1137749459745347,6.610190819819566]])#two honey
                                vectors

Water_database = Vector_database[0:2] #Slices the water vectors from the vector
    database
```

```
Washingup_database = Vector_database[2:4] #Slices the washingup liquid vectors from
    the vector database
Carbonated_database = Vector_database[4:6] #Slices the carbonated soda vectors from
    the vector database
Honey_database = Vector_database[6:8] #Slices the honey vectors from the vector
    database

Covariance_matrix_water = np.array([[((Vector_database[0][0]-(Vector_database[0][0]+
    Vector_database[1][0])/2)**2+(Vector_database[1][0]-(Vector_database[0][0]+
    Vector_database[1][0])/2)**2), (Vector_database[0][0]-((Vector_database[0][0]+
    Vector_database[1][0])/2))*(Vector_database[0][1] - ((Vector_database[0][1]+
    Vector_database[1][1])/2)) + (Vector_database[1][0]-((Vector_database[0][0]+
    Vector_database[1][0])/2))*(Vector_database[1][1] - ((Vector_database[0][1]+
    Vector_database[1][1])/2)), (Vector_database[0][0]-((Vector_database[0][0]+
    Vector_database[1][0])/2))*(Vector_database[0][2] - ((Vector_database[0][2]+
    Vector_database[1][2])/2)) + (Vector_database[1][0]-((Vector_database[0][0]+
    Vector_database[1][0])/2))*(Vector_database[1][2] - ((Vector_database[0][2]+
    Vector_database[1][2])/2))],
                                    [(Vector_database[0][0]-((Vector_database[0][0]+
                                       Vector_database[1][0])/2))*(Vector_database
                                       [0][1] - ((Vector_database[0][1]+
                                       Vector_database[1][1])/2)) + (
                                       Vector_database[1][0]-((Vector_database
                                       [0][0]+Vector_database[1][0])/2))*(
                                       Vector_database[1][1] - ((Vector_database
                                       [0][1]+Vector_database[1][1])/2)), ((
                                       Vector_database[0][1]-(Vector_database
                                       [0][1]+Vector_database[1][1])/2)**2+(
                                       Vector_database[1][1]-(Vector_database
                                       [0][1]+Vector_database[1][1])/2)**2), (
                                       Vector_database[0][1]-((Vector_database
                                       [0][1]+Vector_database[1][1])/2))*(
                                       Vector_database[0][2] - ((Vector_database
                                       [0][2]+Vector_database[1][2])/2)) + (
                                       Vector_database[1][1]-((Vector_database
                                       [0][1]+Vector_database[1][1])/2))*(
                                       Vector_database[1][2] - ((Vector_database
                                       [0][2]+Vector_database[1][2])/2))],
                                    [(Vector_database[0][0]-((Vector_database[0][0]+
                                       Vector_database[1][0])/2))*(Vector_database
                                       [0][2] - ((Vector_database[0][2]+
                                       Vector_database[1][2])/2)) + (
                                       Vector_database[1][0]-((Vector_database
                                       [0][0]+Vector_database[1][0])/2))*(
                                       Vector_database[1][2] - ((Vector_database
                                       [0][2]+Vector_database[1][2])/2)), (
                                       Vector_database[0][1]-((Vector_database
                                       [0][1]+Vector_database[1][1])/2))*(
                                       Vector_database[0][2] - ((Vector_database
                                       [0][2]+Vector_database[1][2])/2)) + (
                                       Vector_database[1][1]-((Vector_database
```

```
                                        [0][1]+Vector_database[1][1])/2))*(
                                        Vector_database[1][2] - ((Vector_database
                                        [0][2]+Vector_database[1][2])/2)), ((
                                        Vector_database[0][2]-(Vector_database
                                        [0][2]+Vector_database[1][2])/2)**2+(
                                        Vector_database[1][2]-(Vector_database
                                        [0][2]+Vector_database[1][2])/2)**2)  ]])

#Inverse covariance matrix is required for the Mahalanobis distance equation
iv_water = (Covariance_matrix_water)**(-1)

Covariance_matrix_washingup = np.array([[((Vector_database[2][0]-(Vector_database
    [2][0]+Vector_database[3][0])/2)**2+(Vector_database[3][0]-(Vector_database
    [2][0]+Vector_database[3][0])/2)**2), (Vector_database[2][0]-((Vector_database
    [2][0]+Vector_database[3][0])/2))*(Vector_database[2][1] - ((Vector_database
    [2][1]+Vector_database[3][1])/2)) + (Vector_database[3][0]-((Vector_database
    [2][0]+Vector_database[3][0])/2))*(Vector_database[3][1] - ((Vector_database
    [2][1]+Vector_database[3][1])/2)), (Vector_database[2][0]-((Vector_database
    [2][0]+Vector_database[3][0])/2))*(Vector_database[2][2] - ((Vector_database
    [2][2]+Vector_database[3][2])/2)) + (Vector_database[3][0]-((Vector_database
    [2][0]+Vector_database[3][0])/2))*(Vector_database[3][2] - ((Vector_database
    [2][2]+Vector_database[3][2])/2))],
                                    [(Vector_database[2][0]-((Vector_database[2][0]+
                                        Vector_database[3][0])/2))*(Vector_database
                                        [2][1] - ((Vector_database[2][1]+
                                        Vector_database[3][1])/2)) + (
                                        Vector_database[3][0]-((Vector_database
                                        [2][0]+Vector_database[3][0])/2))*(
                                        Vector_database[3][1] - ((Vector_database
                                        [2][1]+Vector_database[3][1])/2)), ((
                                        Vector_database[2][1]-(Vector_database
                                        [2][1]+Vector_database[3][1])/2)**2+(
                                        Vector_database[3][1]-(Vector_database
                                        [2][1]+Vector_database[3][1])/2)**2), (
                                        Vector_database[2][1]-((Vector_database
                                        [2][1]+Vector_database[3][1])/2))*(
                                        Vector_database[2][2] - ((Vector_database
                                        [2][2]+Vector_database[3][2])/2)) + (
                                        Vector_database[3][1]-((Vector_database
                                        [2][1]+Vector_database[3][1])/2))*(
                                        Vector_database[3][2] - ((Vector_database
                                        [2][2]+Vector_database[3][2])/2))],
                                    [(Vector_database[2][0]-((Vector_database[2][0]+
                                        Vector_database[3][0])/2))*(Vector_database
                                        [2][2] - ((Vector_database[2][2]+
                                        Vector_database[3][2])/2)) + (
                                        Vector_database[3][0]-((Vector_database
                                        [2][0]+Vector_database[3][0])/2))*(
                                        Vector_database[3][2] - ((Vector_database
                                        [2][2]+Vector_database[3][2])/2)), (
                                        Vector_database[2][1]-((Vector_database
```

```
                                        [2][1]+Vector_database[3][1])/2))*(
                                        Vector_database[2][2] - ((Vector_database
                                        [2][2]+Vector_database[3][2])/2)) + (
                                        Vector_database[3][1]-((Vector_database
                                        [2][1]+Vector_database[3][1])/2))*(
                                        Vector_database[3][2] - ((Vector_database
                                        [2][2]+Vector_database[3][2])/2)), ((
                                        Vector_database[2][2]-(Vector_database
                                        [2][2]+Vector_database[3][2])/2)**2+(
                                        Vector_database[3][2]-(Vector_database
                                        [2][2]+Vector_database[3][2])/2)**2)  ]])

iv_washingup = (Covariance_matrix_washingup)**(-1)

Covariance_matrix_carbonated = np.array([[((Vector_database[4][0]-(Vector_database
    [4][0]+Vector_database[5][0])/2)**2+(Vector_database[5][0]-(Vector_database
    [4][0]+Vector_database[5][0])/2)**2), (Vector_database[4][0]-((Vector_database
    [4][0]+Vector_database[5][0])/2))*(Vector_database[4][1] - ((Vector_database
    [4][1]+Vector_database[5][1])/2)) + (Vector_database[5][0]-((Vector_database
    [4][0]+Vector_database[5][0])/2))*(Vector_database[5][1] - ((Vector_database
    [4][1]+Vector_database[5][1])/2)), (Vector_database[4][0]-((Vector_database
    [4][0]+Vector_database[5][0])/2))*(Vector_database[4][2] - ((Vector_database
    [4][2]+Vector_database[5][2])/2)) + (Vector_database[5][0]-((Vector_database
    [4][0]+Vector_database[5][0])/2))*(Vector_database[5][2] - ((Vector_database
    [4][2]+Vector_database[5][2])/2))],
                                    [(Vector_database[4][0]-((Vector_database[4][0]+
                                        Vector_database[5][0])/2))*(Vector_database
                                        [4][1] - ((Vector_database[4][1]+
                                        Vector_database[5][1])/2)) + (
                                        Vector_database[5][0]-((Vector_database
                                        [4][0]+Vector_database[5][0])/2))*(
                                        Vector_database[5][1] - ((Vector_database
                                        [4][1]+Vector_database[5][1])/2)), ((
                                        Vector_database[4][1]-(Vector_database
                                        [4][1]+Vector_database[5][1])/2)**2+(
                                        Vector_database[5][1]-(Vector_database
                                        [4][1]+Vector_database[5][1])/2)**2), (
                                        Vector_database[4][1]-((Vector_database
                                        [4][1]+Vector_database[5][1])/2))*(
                                        Vector_database[4][2] - ((Vector_database
                                        [4][2]+Vector_database[5][2])/2)) + (
                                        Vector_database[5][1]-((Vector_database
                                        [4][1]+Vector_database[5][1])/2))*(
                                        Vector_database[5][2] - ((Vector_database
                                        [4][2]+Vector_database[5][2])/2))],
                                    [(Vector_database[4][0]-((Vector_database[4][0]+
                                        Vector_database[5][0])/2))*(Vector_database
                                        [4][2] - ((Vector_database[4][2]+
                                        Vector_database[5][2])/2)) + (
                                        Vector_database[5][0]-((Vector_database
                                        [4][0]+Vector_database[5][0])/2))*(
```

```
                                        Vector_database[5][2] - ((Vector_database
                                        [4][2]+Vector_database[5][2])/2)), (
                                        Vector_database[4][1]-((Vector_database
                                        [4][1]+Vector_database[5][1])/2))*(
                                        Vector_database[4][2] - ((Vector_database
                                        [4][2]+Vector_database[5][2])/2)) + (
                                        Vector_database[5][1]-((Vector_database
                                        [4][1]+Vector_database[5][1])/2))*(
                                        Vector_database[5][2] - ((Vector_database
                                        [4][2]+Vector_database[5][2])/2)), ((
                                        Vector_database[4][2]-(Vector_database
                                        [4][2]+Vector_database[5][2])/2)**2+(
                                        Vector_database[5][2]-(Vector_database
                                        [4][2]+Vector_database[5][2])/2)**2)  ]])

iv_carbonated = (Covariance_matrix_carbonated)**(-1)

Covariance_matrix_honey = np.array([[((Vector_database[6][0]-(Vector_database[6][0]+
    Vector_database[7][0])/2)**2+(Vector_database[7][0]-(Vector_database[6][0]+
    Vector_database[7][0])/2)**2), (Vector_database[6][0]-((Vector_database[6][0]+
    Vector_database[7][0])/2))*(Vector_database[6][1] - ((Vector_database[6][1]+
    Vector_database[7][1])/2)) + (Vector_database[7][0]-((Vector_database[6][0]+
    Vector_database[7][0])/2))*(Vector_database[7][1] - ((Vector_database[6][1]+
    Vector_database[7][1])/2)), (Vector_database[6][0]-((Vector_database[6][0]+
    Vector_database[7][0])/2))*(Vector_database[6][2] - ((Vector_database[6][2]+
    Vector_database[7][2])/2)) + (Vector_database[7][0]-((Vector_database[6][0]+
    Vector_database[7][0])/2))*(Vector_database[7][2] - ((Vector_database[6][2]+
    Vector_database[7][2])/2))],
                                    [(Vector_database[6][0]-((Vector_database[6][0]+
                                        Vector_database[7][0])/2))*(Vector_database
                                        [6][1] - ((Vector_database[6][1]+
                                        Vector_database[7][1])/2)) + (
                                        Vector_database[7][0]-((Vector_database
                                        [6][0]+Vector_database[7][0])/2))*(
                                        Vector_database[7][1] - ((Vector_database
                                        [6][1]+Vector_database[7][1])/2)), ((
                                        Vector_database[6][1]-(Vector_database
                                        [6][1]+Vector_database[7][1])/2)**2+(
                                        Vector_database[7][1]-(Vector_database
                                        [6][1]+Vector_database[7][1])/2)**2), (
                                        Vector_database[6][1]-((Vector_database
                                        [6][1]+Vector_database[7][1])/2))*(
                                        Vector_database[6][2] - ((Vector_database
                                        [6][2]+Vector_database[7][2])/2)) + (
                                        Vector_database[7][1]-((Vector_database
                                        [6][1]+Vector_database[7][1])/2))*(
                                        Vector_database[7][2] - ((Vector_database
                                        [6][2]+Vector_database[7][2])/2))],
                                    [(Vector_database[6][0]-((Vector_database[6][0]+
                                        Vector_database[7][0])/2))*(Vector_database
                                        [6][2] - ((Vector_database[6][2]+
```

```
                                        Vector_database[7][2])/2)) + (
                                        Vector_database[7][0]-((Vector_database
                                        [6][0]+Vector_database[7][0])/2))*(
                                        Vector_database[7][2] - ((Vector_database
                                        [6][2]+Vector_database[7][2])/2)), (
                                        Vector_database[6][1]-((Vector_database
                                        [6][1]+Vector_database[7][1])/2))*(
                                        Vector_database[6][2] - ((Vector_database
                                        [6][2]+Vector_database[7][2])/2)) + (
                                        Vector_database[7][1]-((Vector_database
                                        [6][1]+Vector_database[7][1])/2))*(
                                        Vector_database[7][2] - ((Vector_database
                                        [6][2]+Vector_database[7][2])/2)), ((
                                        Vector_database[6][2]-(Vector_database
                                        [6][2]+Vector_database[7][2])/2)**2+(
                                        Vector_database[7][2]-(Vector_database
                                        [6][2]+Vector_database[7][2])/2)**2)  ]])

iv_honey = (Covariance_matrix_honey)**(-1)

#The mean of the vector values is required for the Mahalanobis distance equation
Mean_matrix_water = np.array([[((Vector_database[0][0]+Vector_database[1][0])/2),((
    Vector_database[0][1]+Vector_database[1][1])/2),((Vector_database[0][2]+
    Vector_database[1][2])/2)]])
Mean_matrix_washingup = np.array([[((Vector_database[2][0]+Vector_database[3][0])/2)
    ,((Vector_database[2][1]+Vector_database[3][1])/2),((Vector_database[2][2]+
    Vector_database[3][2])/2)]])
Mean_matrix_carbonated = np.array([[((Vector_database[4][0]+Vector_database[5][0])
    /2),((Vector_database[4][1]+Vector_database[5][1])/2),((Vector_database[4][2]+
    Vector_database[5][2])/2)]])
Mean_matrix_honey = np.array([[((Vector_database[6][0]+Vector_database[7][0])/2),((
    Vector_database[6][1]+Vector_database[7][1])/2),((Vector_database[6][2]+
    Vector_database[7][2])/2)]])


#
    ----------------------------------------------------------------------------------------------------


# Reads in previously saved settings from a settings text file
    ----------------------------------------------------------------------------------------------------

with open('settings.txt') as fileinput:
    line0 = fileinput.readlines()
    settings['COM'] = line0[0].strip()
    settings['beta'] = line0[1].strip()
    settings['V0'] = line0[2].strip()
    settings['H'] = line0[3].strip()
    settings['G'] = line0[4].strip()
```

```
#Reads in previously saved guess values for least squared fitting
    ----------------------------------------------------------------------------------------------------


with open('analysis_data.txt') as fileinput:
    line0 = fileinput.readlines()
    analysis_data['A0'] = line0[0].strip()
    analysis_data['B0'] = line0[1].strip()
    analysis_data['C0'] = line0[2].strip()

#
    ----------------------------------------------------------------------------------------------------



# Main window of UI
    ----------------------------------------------------------------------------------------------------

class GUI(tk.Tk):

    def __init__(self, *args, **kwargs): #defines the variable self to mean the main
         window of UI
        ------------------------------------------------------------

        tk.Tk.__init__(self, *args, **kwargs)

        tk.Tk.iconbitmap(self, default="icon.ico") # Sets the icon of the UI
        tk.Tk.wm_title(self, "Torcelli's Law Sensor") # Sets the title of the main
            UI window


        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand = True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, PageOne, PageTwo, PageThree, PageFour, PageFive): #
            Defines all of the pages present in the UI

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
```

```
        frame = self.frames[cont]
        frame.tkraise()

        self.geometry('500x650+650+180') #Defines the size of the main UI window

#
    ----------------------------------------------------------------------------------------------------


# Main home page of UI
    ----------------------------------------------------------------------------------------------------


class StartPage(tk.Frame):


    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        label = tk.Label(self,text="Torcelli's Law", font=LARGE_FONT)
        label.pack(pady=10,padx=10)
        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=70,padx=10)
        label = tk.Label(self, text="Menu", font=LARGE_FONT)

        button = ttk.Button(self, text="Settings",
                            command=lambda: controller.show_frame(PageOne)) #
                                Settings menu button opens Setting menu
        button.pack()

        button2 = ttk.Button(self, text="Run",
                            command=lambda: controller.show_frame(PageTwo)) # Run
                                button opens Run page
        button2.pack()#

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=70,padx=10) #Empty label for formatting purposes


        button3 = ttk.Button(self, text="Liquid Detector",
                            command=lambda: controller.show_frame(PageFive)) #Liquid
                                 detector button opens Liquid detector page
        button3.pack()

#
    ----------------------------------------------------------------------------------------------------


# Settings Menu Page
    ----------------------------------------------------------------------------------------------------
```

```
class PageOne(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)




        label = tk.Label(self, text="Settings␣Menu", font=LARGE_FONT) #Title of page
             "Settings Menu"
        label.pack(pady=35,padx=10)

        L1 = Label(self, text="COM␣Port:", font=LARGE_FONT) #Names the COM Port
            setting
        L1.pack()
        COM = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        COM.insert(END, settings['COM']) #Puts the saved previous value into the
            entry box
        entryString = COM.get()
        COM.pack()

        label = tk.Label(self, text="", font=LARGE_FONT) #Empty label for formatting
             purposes
        label.pack(padx=10)

        L2 = Label(self, text="Beta␣($\beta$)␣:", font=LARGE_FONT) #Names the Beta
            setting
        L2.pack()
        beta = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        beta.insert(END, settings['beta']) #Puts the saved previous value into the
            entry box
        entryString = beta.get()
        beta.pack()

        label = tk.Label(self, text="", font=LARGE_FONT) #Empty label for formatting
             purposes
        label.pack(padx=10)

        L2 = Label(self, text="V0␣:", font=LARGE_FONT) #Names the V0 setting
        L2.pack()
        V0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        V0.insert(END, settings['V0']) #Puts the saved previous value into the entry
             box
        entryString = V0.get()
        V0.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        L2 = Label(self, text="H␣:", font=LARGE_FONT)
```

```
        L2.pack()
        H = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        H.insert(END, settings['H']) #Puts the saved previous value into the entry
            box
        entryString = H.get()
        H.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        L2 = Label(self, text="G␣:", font=LARGE_FONT)
        L2.pack()
        G = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        G.insert(END, settings['G']) #Puts the saved previous value into the entry
            box
        G1 = G.get()
        settings['G'] = G1
        G.pack()

        label = tk.Label(self, text="", font=LARGE_FONT) #Empty label for formatting
             purposes
        label.pack(pady=5,padx=10)


        def makeSomething(name, variable): #A function which allows you to add
            inputted values to a previously defined dictionary or list
            settings[name] = variable.get()

        def Save(filename): #A function which writes all of the settings to a text
            file
            x = open(filename, "w")
            x.writelines(str(settings['COM']))
            x.writelines("\n")
            x.writelines(str(settings['beta']))
            x.writelines("\n")
            x.writelines(str(settings['V0']))
            x.writelines("\n")
            x.writelines(str(settings['H']))
            x.writelines("\n")
            x.writelines(str(settings['G']))
            x.close()

        button3 = ttk.Button(self, text = 'Accept', command=lambda:[makeSomething('
            COM', COM), makeSomething('beta', beta), makeSomething('V0', V0),
            makeSomething('H', H), makeSomething('G', G), Save("settings.txt")])#
            Button which will save the values

        button3.pack()


        label = tk.Label(self, text="", font=LARGE_FONT)
```

```
        label.pack(pady=0,padx=10)

        button2 = ttk.Button(self, text="Run",
                             command=lambda: controller.show_frame(PageTwo)) #Button
                                 which takes you to the experiment run page
        button2.pack()


        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=10,padx=10)


        button1 = ttk.Button(self, text="Back␣to␣Menu",
                             command=lambda: controller.show_frame(StartPage)) #
                                 Button which will take you back to the main menu
        button1.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

#
    -----------------------------------------------------------------------------------------------


# Experiment Run page
    -----------------------------------------------------------------------------------------------


class PageTwo(tk.Frame):


    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Run␣Experiment", font=LARGE_FONT)
        label.pack(pady=10,padx=10)


        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        progress=Progressbar(self,orient=HORIZONTAL,length=100,mode='determinate')

        def millis(): #Function needed to measure the time taken for the experiment
            to run correctly
            "return␣a␣timestamp␣in␣milliseconds␣(ms)"
            tics = ctypes.c_int64()
            freq = ctypes.c_int64()

            #get ticks on the internal ~2MHz QPC clock
            ctypes.windll.Kernel32.QueryPerformanceCounter(ctypes.byref(tics))
            #get the actual freq. of the internal ~2MHz QPC clock
```

```
        ctypes.windll.Kernel32.QueryPerformanceFrequency(ctypes.byref(freq))

        t_ms = tics.value*1e3/freq.value
        return t_ms

    def FileExplorer(): #Function which allows the file explorer to open on a
        specific file location
        subprocess.Popen(r'explorer /select,"C:\Users\JamieSomers\Desktop\GUI\
            Run_Results.csv"')

    def flush(): #Flushes the excel file so the code reads the newly saved
        values
        f = open('Run_Results.csv','r')
        f.flush()

    def bar(): #Function which defines the progress bar
        import time
        green = 0
        Ti = 0
        begin = True
        time.sleep(6)
        while green < 90000:
            if begin == True:
                Ti = millis();
                begin = False;
            green = millis() - Ti
            progress['value'] = green/900
            self.update_idletasks()
            time.sleep(1)

    progress.pack()

    def RUN(): # Main function of the experiment which reads all of the values
        from the arduino, displays them in the UI and saves them to a csv file
        board = Arduino("COM{}".format(settings["COM"])) #Identifies the COM
            Port which the Arduino is connected to based on the COM port set in
            the settings menu
        iterator = util.Iterator(board) #To read analog ports we need to start
            an iterator thread
        iterator.start() #This means the board will stop sending data to serial
        a0 = board.get_pin('a:0:i') #This identifies the pin the pressure sensor
             is connected to as an input and a variable we can use in our code
        beta = float(settings["beta"]) #This takes the beta value defined in the
             settings menu
        V0 = float(settings["V0"]) #" "
        H = float(settings["H"])#" "
        G = float(settings["G"])#" "
        t = 0 #This sets the time t = 0
        T0 = 0 #This sets the time T0 = 0
        first = True #This statement is required for the millis() at the
            beginning of the experiment to be recorded
```

```
            a = []  #This is a list that will store the time and height values for
                the experiment
            f = open("Run_Results.csv", "w") #Opens the Run_Results excel file
            f.truncate() #Deletes any previous values in the excel file so it is
                free to use
            f.close() #Closes the excel file
            time.sleep(1.0) #Tells the program to sleep for 1 second, this is
                required when using PyFirmata otherwise the Arduino will be
                unresponsive
            while t < 90000: #This is the limit to how long the experiment should
                run for in miliseconds, it is set at 90 seconds
                N = 1000*(a0.read()) #Defines the interger N, since we are using
                    python instead of C++ we are required to multiply the sensor
                    data by 1,000 to get the same value
                V = G*N #Defines the Voltage V as being G multiplied by N
                h = (V-V0)/beta #Defines the height as being V - V0 divided by beta
               #h = H - ((0.034*t)/2 - 2)  #height value for ultrasonic sensor
                if h <= H: #If the height of the liquid is less than or equal to the
                     height of the container
                    if first == True: #This will first be a boolean that equates to
                        true before staying false for the duration of the experiment
                        T0 = millis(); #This takes the time the experiment was
                            started
                        first = False; #This stops the if statement from running
                            again and deleting the start time
                    t = millis() - T0 #This measures the time as being the current
                        time on the machines clock minus the start time
                    a += [float(t), float(h)]; #This saves both the t and h values
                        in the a list defined above
                    textbox.insert(END,[t/1000, h,]) #This displays both the t value
                         in seconds and the h value on the GUI
                    textbox.insert(END, '\n') #This causes the next set of data to
                        be displayed on the line below
                    textbox.see('end') #This causes the UI to scroll as new values
                        are added
                    with open("Run_Results.csv", "a+", newline='') as file: #Opens
                        the results csv file
                        writer = csv.writer(file, delimiter=',') #Tells the program
                            to seperate the values with a comma (comma-seperated
                            values)
                        writer.writerow([t/1000, h]) #Writes the time in seconds and
                             the height in cm's to the file


                else:
                    print("")
                    print("!RSET") #If the height of the liquid h is greated than
                        the inputted height of the container H than the experiment
                        should not begin
            board.exit() #It is required to exit the board otherwise Anaconda will
                need to be restarted for the COM port to be cleared
```

```
        w = threading.Thread(target=bar) #It is required to run the progress bar
            through a Daemon thread to avoid it becoming unresponsive
        w.setDaemon(True)
        t = threading.Thread(target=RUN) #It is required to run the experiment
            measurement through a Daemon thread to avoid the UI becoming
            unresponsive
        t.setDaemon(True)

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=10,padx=10)


        button7 = ttk.Button(self, text="Start",
                             command=lambda:[t.start(),w.start()]) #This button
                                 starts the readings and progress bar at the same
                                 time
        button7.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(pady=5,padx=10)

        textbox = Text(self, bd=2, height=15, width=40)
        textbox.pack(padx=85)

        button2 = ttk.Button(self, text="Open␣CSV␣File␣",
                             command=lambda: FileExplorer()).place(x=85, y=470) #This
                                  button opens the file explorer where the results
                                 are saved

        button3 = ttk.Button(self, text="␣Graph␣",
                             command=lambda:[controller.show_frame(PageThree), flush
                                 ()]).place(x=217.5, y=470) #This button takes you to
                                  the graph page and flushes the excel values

        button4 = ttk.Button(self, text="Analyze␣Data",
                             command=lambda: controller.show_frame(PageFour)).place(x
                                 =335, y=470) #This button takes you to the least
                                 squared fitting page

        button1 = ttk.Button(self, text="Back␣to␣Menu",
                             command=lambda: controller.show_frame(StartPage)).place(
                                 x=214, y=550) #This button takes you back to the
                                 menu

#
    ----------------------------------------------------------------------------------------
```

```
# Graph Generation page
    ----------------------------------------------------------------------------------------------------


class PageThree(tk.Frame):


    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Graph", font=LARGE_FONT)
        label.pack(pady=10,padx=10)
        PageThree.update(self)

        def Graph(): #Defines the graph function which will create a matplotlib
            scatterplot in the UI
            PageThree.update(self)
            self.update()
            f = open('Run_Results.csv','r') #Opens the results just recorded in the
                Experiment Run page
            my_file = f
            temp_data=[] #A new list to store temp data
            for line in my_file: #This is the beginning of the function 'for' which
                tells the program what to do FOR every line in the opened file
                my_file created above.
                t, h = line.split(',') #This tells the program to seperate the file
                    on each line into a stringe with two components an x components,
                     they will be split wherever there is a comma in the string.
                temp_data += [float(t), float(h)] #Saves the t and h values measured
                     in a temp list

            my_file.close()#Closes the csv file

            data_set = np.array(temp_data) #places the temp data in a numpy array

            t_data = data_set[0::2] #This is using whats known as 'Slice Notation',
                specifically this line of code tells the program to take the first
                element in the list before stepping 1 and taking what would be the
                third element, the reason its skipping one is to ignore the h-values
                 as this is the variable for t_data.
            h_data = data_set[1::2] #This is more Slice notation telling the program
                 to start at the second element in the list (1, since python starts
                counting at 0) and step 1 element every time as this is the variable
                 for h_data.

            f = Figure(figsize=(5,5), dpi=100) #Creates a figure window with our
                defined dimensions and dpi
            a = f.add_subplot(111) #Adds a subplot to the figure window
            a.scatter(t_data,h_data, s=0.5, color="red") #Creates a scatterplot with
                 our t and h values
```

```
            canvas = FigureCanvasTkAgg(f, self) #The canvas allows us to have the
                figure show in our UI
            canvas.draw() #This shows the figure in the UI
            canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

            toolbar = NavigationToolbar2Tk(canvas, self) #This allows us to have a
                tool bar which we can use to zoom in and out aswell as save the
                scatterplot
            toolbar.update()
            canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        generate = ttk.Button(self, text="Generate",
                              command=lambda: Graph()) #Creates a generate button
                                  which will generate the scatter plot
        generate.pack()


        button1 = ttk.Button(self, text="Back",
                              command=lambda: controller.show_frame(PageTwo)) #Creates
                                   a back button which allows us to go back to the
                                  Experiment Run page
        button1.pack()

#
    ------------------------------------------------------------------------------------------


#Least Squares Fitting page
    ------------------------------------------------------------------------------------------


class PageFour(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Least␣Squares␣Fitting", font=LARGE_FONT)
        label.pack(pady=10,padx=10)
        PageThree.update(self)
        self.update()

        filename = tk.Label(self, text="Empty").place(x=65, y=200) #Sets the
            filename as Empty before the user selects a file

        A0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        A0.insert(END, analysis_data['A0']) #Inserts the saved guess value for A
        A0.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        B0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
```

```
B0.insert(END, analysis_data['B0']) #Inserts the saved guess value for B
B0.pack()

label = tk.Label(self, text="", font=LARGE_FONT)
label.pack(padx=10)

C0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
C0.insert(END, analysis_data['C0']) #Inserts the saved value for C
C0.pack()

label = tk.Label(self, text="", font=LARGE_FONT)
label.pack(padx=10)

textbox = Text(self, bd=2, height=15, width=45)
textbox.place(x=65, y=240) #Creates a text box


def filefinder(): #Defines a function which allows the user to select a file
     and run least squared fitting on it
    textbox = Text(self, bd=2, height=15, width=45).place(x=65, y=240) #
        Defines the textbox within the scope of the function
    file = filedialog.askopenfile(parent=self,mode='rb',title='Choose␣a␣file
        ') #creates a button which allows the user to select a file
    if file:
        data = open(os.path.basename(file.name)) #Opens the file based on
            the file path name
        filename = tk.Label(self, text="␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
            ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣").place(x=65, y=200)
            #Creates a blank name on top of the filename so multiple file
            names dont overlap
        filename = tk.Label(self, text=os.path.basename(file.name)).place(x
            =65, y=200) #Replaces the Empty filename with the actual file
            name
        excel_temp = [] #Creates a list for temp excel values
        for line in data: #This is the beginning of the function 'for' which
             tells the program what to do FOR every line in the opened file
            'data' created above.
            t, h = line.split(',') #This tells the program to seperate the
                file on each line into a stringe with two components, a t
                component and a h component, they will be split wherever
                there is a comma in the string.
            excel_temp += [float(t), float(h)] #Writes the t and h values to
                 the temp excel list
        file.close() #closes the file
        data_set = np.array(excel_temp) #puts the temp excel values into a
            numpy array
        t_data = data_set[0::2] #This slices all of the t values
        h_data = data_set[1::2] #This slices all of the h values
        N=len(t_data) #the number of t values
        t_best = np.linspace(t_data[0],t_data[N - 1],num=300) #Creates 300
            values in between the t values gotten 0 to 90 seconds, so that
```

```
        our line of best fit is smoother
    inpA0 = A0.get() #Gets the A guess inputted
    inpB0 = B0.get() #Gets the B guess inputted
    inpC0 = C0.get() #Gets the C guess inputted
    floatA0=float(inpA0) #Changes the value inputed for A into a
        floating point number. This changes intergers into usable values
         with a decimal.
    floatB0=float(inpB0) #Changes the value inputed for B into a
        floating point number. This changes intergers into usable values
         with a decimal.
    floatC0=float(inpC0) #Changes the value inputed for C into a float
        point number. This changes intergers into usable values with a
        decimal.
    init_guess = np.array([floatA0,floatB0,floatC0]) #Creates a numpy
        array for the A, B and C guess
    def lin_fit(par): #This line is the beginning of a new function
        denoted by the def keyword, and the function name which is
        defined as lin_fit with the argument (par).
        A=par[0] #This line sets up the variable 'A' to be hard coded as
             0 in the function parameters.
        B=par[1] #This line sets up the variable 'B' to be hard coded as
             1 in the function parameters.
        C=par[2] #This line sets up the variable 'C' to be hard coded as
             2 in the function parameters.
        squared\_diff = ((A*((t_data)**2) + B*(t_data) + C)-h_data)**2 #
            This equation is given the name 'squared$\_$diff' its the
            squared difference of a quadratic equation
        sum_squares = np.sum(squared_diff) #Since the calculation above
            uses many different values for the t_data and h_data, this
            line uses the NumPy SUM function to add up all of the
            answers and gives this answer the variable name sum_squares
        return sum_squares
    best_fit = sp.optimize.fmin(lin_fit, init_guess, maxiter=None,
        maxfun=None, full_output=0, disp=0, retall=0, callback=None) #
        This creates the line of best fit using sp.omtimize
    textbox = Text(self, bd=2, height=15, width=46)
    textbox.place(x=65, y=240)

    textbox.see('end') #Allows us to scroll to the newest value in the
        textbox
    t = t_data #defines t and the t_data
    y1=(floatA0*((t_data)**2) + floatB0*(t_data) + floatC0) #Creates a
        y1 value based on our guesses and our t values
    y2=(best_fit[0]*((t_best)**2) + best_fit[1]*(t_best) + best_fit[2])
        #Creates a y2 value based on the best fit lines calculated

    def line(tdata,A,B,C): #This line is the beginning of a new function
         denoted by the def keyword, and the function name which is
        defined as line with the arguments (tdata,A,B and C).
        return A*((t_data)**2) + B*(t_data) + C #This is the return
            statement, it returns the value produced by the equation
```

```
                    when the values for tdata,A,B and C are all inputted.

            best_fit, cov = sp.optimize.curve_fit(line, t_data, h_data,[floatA0,
                floatB0,floatC0]) #Creates two variables best_fit and cov, which
                 both equate to an optimized curve fit function where our
                equation is f, our x values are x_data, our y values are y_data
                and uses our initial guesses A0, B0 and C0. these variables both
                 produce arrays
            fit_err = np.sqrt(np.diag(cov)) #Creates a variable named fit_err
                which equates to the square root of the diagonal array produced
                by the cov variable defined above.

            textbox.insert(END,"Best␣fit␣value␣of␣A:") #Writes the line to the
                textbox
            textbox.insert(END,"\n") #Writes a new line to the textbox
            textbox.insert(END,[best_fit[0],"$\pm$", fit_err[0]]) #Writes the
                best fit value and the error
            textbox.insert(END,"\n")
            textbox.insert(END,"\n")
            textbox.insert(END,"Best␣fit␣value␣of␣B:")
            textbox.insert(END,"\n")
            textbox.insert(END,[best_fit[1],"$\pm$", fit_err[1]])
            textbox.insert(END,"\n")
            textbox.insert(END,"\n")
            textbox.insert(END,"Best␣fit␣value␣of␣C:")
            textbox.insert(END,"\n")
            textbox.insert(END,[best_fit[2],"$\pm$", fit_err[2]])
            fig = plt.figure(figsize=(10,10)) #Creates a plt figure named fig
                with our dimensions defined
            plt.scatter(t_data, h_data, s=1, color="red") #Scatter plot made
                using our t and h values found and selects the data point size
                and color
            plt.plot(t_best, y2, color="blue", linewidth=1.5)#line of best fit
                using the 300 data points between 0 and 90 seconds and the best
                fit values for y. The color of the line of best fit is blue and
                the line width is 1.5
            plt.xlabel('Time␣(s)') #label the x-axis with the word "Time (s)"
            plt.ylabel('Height␣(cm)') #label the y-axis with the word "Height (
                cm)"
            plt.title('Head␣versus␣time␣(data␣points␣in␣red,␣best␣fit␣line␣in␣
                blue)') #label the plot title Head versus time (data points in
                red, best fit line in blue)



    button1 = ttk.Button(self, text="Select␣File",
                         command=lambda:filefinder()).place(x=354, y=200) #Button
                              to select a file

    button1 = ttk.Button(self, text="Back",
```

```
                            command=lambda: controller.show_frame(PageTwo)).place(x
                                =214, y=550) #Button to go back to the Run
                                Experiment Page

#
    ----------------------------------------------------------------------------------------------------


# Liquid Detector page
    ----------------------------------------------------------------------------------------------------


class PageFive(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Liquid␣Detector␣/␣Indentifier", font=LARGE_FONT
            )
        label.pack(pady=10,padx=10)
        PageThree.update(self)
        self.update()

        filename = tk.Label(self, text="Empty").place(x=65, y=200) #Sets the
            filename as Empty before the user selects a file

        A0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        A0.insert(END, analysis_data['A0']) #Inputs the guess for A0
        A0.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        B0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        B0.insert(END, analysis_data['B0']) #Inputs the guess for B0
        B0.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        C0 = tk.Entry(self,bd=1, show=None, font=LARGE_FONT)
        C0.insert(END, analysis_data['C0']) #Inputs the guess for C0
        C0.pack()

        label = tk.Label(self, text="", font=LARGE_FONT)
        label.pack(padx=10)

        def openNewWindow(liquidname): #Defines the function which opens a new
            window with the liquid name in it

            # Toplevel object which will
            # be treated as a new window
```

```
            newWindow = Toplevel(self)

            # sets the title of the
            # Toplevel widget
            newWindow.title("Liquid␣Identified") #Names the window 'Liquid
                Identified'

            # sets the geometry of toplevel
            newWindow.geometry("255x20+770+500") #Sets the window dimensions

            # A Label widget to show in toplevel
            Label(newWindow,
                  text = liquidname).pack()


        fig = Figure(dpi=90)
        a = Axes3D(fig) #Creates a figure with 3D dimensions
        a.scatter(Water_database[0][0], Water_database[0][1], Water_database[0][2],
            color='blue') #Generates the first water vector in blue
        a.scatter(Water_database[1][0], Water_database[1][1], Water_database[1][2],
            color='blue') #Generates the second water vector in blue
        a.scatter(Washingup_database[0][0], Washingup_database[0][1],
            Washingup_database[0][2], color='green')#Generates the first washingup
            liquid vector in green
        a.scatter(Washingup_database[1][0], Washingup_database[1][1],
            Washingup_database[1][2], color='green')#Generates the second washingup
            liquid vector in green
        a.scatter(Carbonated_database[0][0], Carbonated_database[0][1],
            Carbonated_database[0][2], color='red')#Generates the first carbonated
            soda vector in red
        a.scatter(Carbonated_database[1][0], Carbonated_database[1][1],
            Carbonated_database[1][2] , color='red')#Generates the second carbonated
             soda vector in red
        a.scatter(Honey_database[0][0], Honey_database[0][1], Honey_database[0][2],
            color='orange')#Generates the first honey vector in orange
        a.scatter(Honey_database[1][0], Honey_database[1][1], Honey_database[1][2],
            color='orange')#Generates the second honey vector in orange


        canvas = FigureCanvasTkAgg(fig, self)
        canvas.draw() #Creates the 3D plot
        canvas.get_tk_widget().pack(side=tk.BOTTOM)

        toolbar = NavigationToolbar2Tk(canvas, self) #Creates the same toolbar we
            used in the graph sections
        toolbar.update()
        canvas._tkcanvas.pack(side=tk.TOP)

        def LiquidDetector(): #Defines the function which will detect the unknown
            liquid
```

```
            file = filedialog.askopenfile(parent=self,mode='rb',title='Choose␣a␣file
                ') #Lets the user choose a file to input as an unknown liquid
            if file:
                data = open(os.path.basename(file.name)) #opens the file based on
                    the path name
                filename = tk.Label(self, text="␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
                    ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣").place(x=65, y=175)
                filename = tk.Label(self, text=os.path.basename(file.name)).place(x
                    =65, y=175)
                excel_temp = []
                for line in data: #This is the beginning of the function 'for' which
                     tells the program what to do FOR every line in the opened file
                    my_file created above.
                    x, y = line.split(',') #This tells the program to seperate the
                        file on each line into a stringe with two components an x
                        components, they will be split wherever there is a comma in
                        the string.
                    excel_temp += [float(x), float(y)]
                file.close()
                data_set = np.array(excel_temp)
                x_data = data_set[0::2]
                y_data = data_set[1::2]
                inpA0 = A0.get()
                inpB0 = B0.get()
                inpC0 = C0.get()
                floatA0=float(inpA0)
                floatB0=float(inpB0)
                floatC0=float(inpC0)
                init_guess = np.array([floatA0,floatB0,floatC0])
                def lin_fit(par):
                    A=par[0]
                    B=par[1]
                    C=par[2]
                    squared_diff = ((A*((x_data)**2) + B*(x_data) + C)-y_data)**2
                    sum_squares = np.sum(squared_diff)
                    return sum_squares
                best_fit = sp.optimize.fmin(lin_fit, init_guess, maxiter=None,
                    maxfun=None, full_output=0, disp=0, retall=0, callback=None)

                x = x_data

                def line(xdata,A,B,C):
                    return A*((x_data)**2) + B*(x_data) + C

                best_fit, cov = sp.optimize.curve_fit(line, x_data, y_data,[floatA0,
                    floatB0,floatC0])

                temp_array = np.array([[best_fit[0], best_fit[1], best_fit[2]]]) #
                    Places the unknown liquids A, B and C values in a numpy array
```

```
            distances.append(np.sqrt(((temp_array - Mean_matrix_water).transpose
                () * iv_water * (temp_array - Mean_matrix_water))[0][0])) #Finds
                 the mahalanobis distance from the water vectors
            distances.append(np.sqrt(((temp_array - Mean_matrix_washingup).
                transpose() * iv_washingup * (temp_array - Mean_matrix_washingup
                ))[0][0])) #Finds the mahalanobis distance from the washingup
                liquid vectors
            distances.append(np.sqrt(((temp_array - Mean_matrix_carbonated).
                transpose() * iv_carbonated * (temp_array -
                Mean_matrix_carbonated))[0][0])) #Finds the mahalanobis distance
                 from the carbonated soda vectors
            distances.append(np.sqrt(((temp_array - Mean_matrix_honey).transpose
                () * iv_honey * (temp_array - Mean_matrix_honey))[0][0])) #Finds
                 the mahalanobis distance from the honey vectors
            smallest_value = distances[0] #makes the smallest value the first
                distance
            for i in range(len(distances)): #for i in the range of the number of
                 distances
                if distances[i] <= smallest_value: #if distance i is smaller or
                    equal to the smallest value
                    smallest_value = distances[i] #make that distance the new
                        smallest value
                else:
                    smallest_value = smallest_value #keep the smallest value the
                         same
            index = distances.index(smallest_value) #find what index in the list
                 the smallest value is
            if index == 0: #if its the first distance, the liquid is identified
                as water
                openNewWindow('Water')
            elif index == 1: #if its the second distance, the liquid is
                identified as washing-up liquid
                openNewWindow('Washing-up␣liquid')
            elif index == 2: #if its the third distance, the liquid is
                identified as carbonated soda
                openNewWindow('Carbonated␣Soda')
            else: #if its none of the above distances, its the fourth distance
                and the liquid is identified as honey
                openNewWindow('Honey')
            distances.clear() #This clears the distances list so the function
                can be run again
            a.scatter(temp_array[0][0], temp_array[0][1], temp_array[0][2],
                color='black') #generates the unknown liquid vector in black
            canvas.draw() #generates a new 3d plot

        filebutton = ttk.Button(self, text="Select␣File",
                            command=lambda:LiquidDetector()).place(x=354, y=170) #
                                Creates the button which runs the liquid detector
                                once a file is selected

        backbutton = ttk.Button(self, text="Back",
```

```
                            command=lambda: controller.show_frame(StartPage)).place(
                                x=214, y=570) #Button to go back to the start page

#
    --------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


app = GUI()
app.mainloop()
```

Python code used to plot efflux velocity

```
 # -*- coding: utf-8 -*-
"""
Created on Wed Apr  7 12:42:58 2021

@author: JamieSomers
"""

import numpy as np
import matplotlib.pyplot as plt


filename = input('Data␣file␣name␣(including␣extension):␣') #Type in the name of the
    csv file
data = open(filename) #Opens csv file for use
excel_temp = [] #temp list
v = [] #list for v values
for line in data: #This is the beginning of the function 'for' which tells the
    program what to do FOR every line in the opened file my_file created above.
    t, y = line.split(',') #This tells the program to seperate the file on each line
         into a stringe with two components t and y components, they will be split
        wherever there is a comma in the string.
    excel_temp += [float(t), float(y)] #saves the t and y values as floats in the
        temp list
data.close() #close the csv file after use

t_data = excel_temp[0::2] #This is using whats known as 'Slice Notation',
    specifically this line of code tells the program to take the first element in
    the list before stepping 1 and taking what would be the third element, the
    reason its skipping one is to ignore the y-values as this is the variable for
    t_data.
y_data = excel_temp[1::2]
for i in range(len(t_data)):
    v.append(np.sqrt(2*981*y_data[i])) #This is the equation for efflux velocity,
        the square root of 2 g h, g is in cm/s^1

plt.plot(t_data, v, color="red") #Scatterplot using t-values, v values, and changes
    the color to red.
plt.xlabel('Time␣(s)') #label the x-axis with the word "'Time (s)')"
```

```
plt.ylabel('Velocity␣(cm␣s$^{-1}$)') #label the y-axis with the word "Velocity (cm s
    ^-1)"
plt.title('Efflux␣speed␣versus␣time') #label the plot title "Efflux speed versus
    time"
plt.savefig('Efflux1.png', dpi=200)#This line of code saves the plot as a .png in
    higher resolution than Spyder will allow
plt.show()
```