Aashrit's SerialPlotter v1

INSTRUCTION MANUAL

Installation

SerialPlotter is only compatible with Windows computers.

No installation is required: just copy the entire folder labelled "SerialPlotter" to the desired directory and create a shortcut of the executable from \SerialPlotter\dist\SerialPlotter.exe

All required software and libraries are included in the folder. The source code is appended to the end of this document and the Python file is accessible at \SerialPlotter\SerialPlotter.py

The program was built using the Spyder 5.0.5 IDE with

Python 3.8.8	64-bit
pySerial 3.4	to get serial data
Matplotlib 3.2.2	to create the charts
PyInstaller 3.6	to build the executable

Use

Setting up the Arduino:

SerialPlotter reads the output of the **Serial.print()** command from the Arduino when using the following syntax within **void loop() {**

For the desired variables X and Y that need to be plotted:

```
//All the other code that gives the two outputs, then
Serial.print(X);      //There must be at least one data point
Serial.print(" ");    //Ensure this is a space, do not use commas
Serial.println(Y);    //This has to be "println"
```

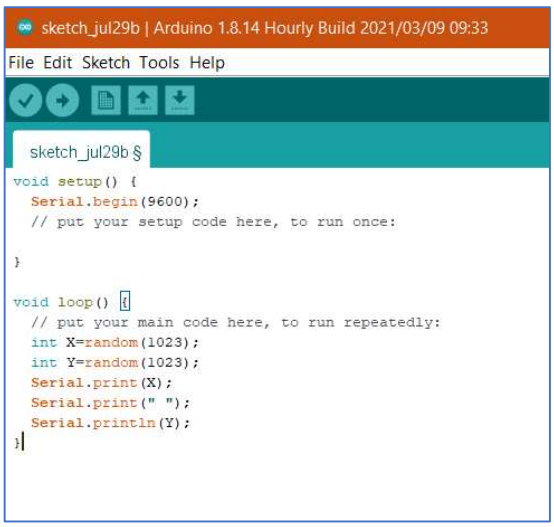

Figure 1: Example Arduino code

While any variable name can be used, ensure that the data type is numerical, preferably **int**, **float** or **double**. Also ensure that this is the only data being printed to the serial port from the Arduino. Since only one connection to the serial port is permitted at a time, you will not be able to interact with the Arduino via the computer while SerialPlotter is running, so there is no reason to have any other information printed to the Serial.

The first data point needs to be printed. Don't print "Y" if not required. In that case, the code should read as:

```
Serial.println(X);          //The "println" cannot be replaced with print and "\n"
```

In Arduino IDE, the baud rate can be set in **void setup() {** using the command **Serial.begin(*baudrate*);**

While most devices use the standard 9600 baud, if required, the rate can be changed in SerialPlotter using the "**Additional Options**" as covered below.

Using SerialPlotter:

On launching SerialPlotter, a very stylish logo will be displayed. Following this, it will begin to request inputs. Type "y" (without quotes) and press Enter to say "Yes" to each question. Type "n", followed by Enter, or simply press Enter without typing anything to say "No". Where data needs to be inputted, do so and then press Enter.

Enable live graph of data? (y/n):

This will enable the graphs for the serial data. Since plotting the graphs is the slowest process in the program, disable the graphs if small time increments are required. The formats of the graphs are shown below.

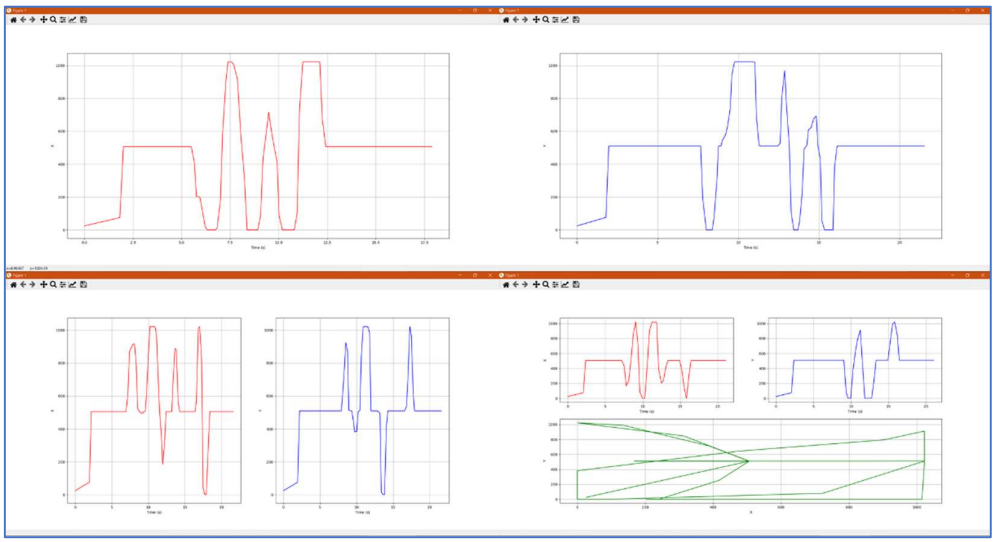

Figure 2: Various chart layouts available: X versus time (top left), Y versus time (top right), both X and Y (bottom left), X, Y and X versus Y. Any graph or combination of graphs can be used.

Enable X vs Time graph? (y/n):

This will enable the graph of the first variable on the y-axis and the computer timer on the x-axis.

Enable Y vs Time graph? (y/n):

This will enable the graph of the second variable on the y-axis and the computer timer on the x-axis.

Enable X vs Y graph? (y/n):
This will enable the graph of the second variable on the y-axis against the first variable on the x-axis.

View additional options? (y/n):
This will allow the tuning of the other settings related to running SerialPlotter and getting usable data. If "y" was chosen, the following prompts will be presented.

Graph refresh rate (min=0.010, def=0.025):
This controls the rate at which the plots will be refreshed. This also represents the gap between two readings used between the plot points, independent of the time increments of the ASP_*.csv file. By default, the graph refreshes every 0.025 seconds, or 25 milliseconds. The minimum possible gap is of 10 milliseconds. If the value entered is not accepted or left blank, the following message will be displayed after all entries:
Graph refresh rate set to <*saved value*> seconds.

Maximum readings on each axis (def=200):
This controls the maximum number of readings maintained on the axes of the graph. The plot scrolls automatically to display the latest readings. The default is 200 readings, which roughly corresponds to one minute of data when all three graphs are enabled, and lesser for fewer charts. If the value entered is not accepted or left blank, the following message will be displayed after all entries:
Axes lengths set to <*saved value*> readings each.

Input Arduino name, or skip with Enter:
If the device name is not the standard "USB Serial Device" (as in the case of non-genuine Arduinos), enter the name here. The device name can be found by searching for "Device Manager" and checking under "Ports (COM & LPT)" for the name that shows up when the Arduino is plugged in, as displayed below. Note: The name of the device is not the same as that shown in the Arduino IDE. An incorrect or missing entry will make SerialPlotter use the saved name (by default "USB Serial Device") to connect to the Arduino. If the saved name doesn't work, then it will resort to the default.

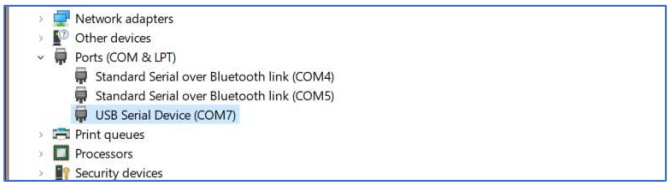

Figure 3: The name of device as shown in the Device Manager

Input the Arduino baud rate (def=9600):
The baud rate of the Arduino can be entered here so that SerialPlotter can communicate with the device. If the value entered is not accepted or left blank, the following message will be displayed after all entries:
Baud rate set to <*saved value*> baud.

Save the accepted values future use? (y/n) or 'd' to reset to defaults:
This will save the values entered in the "**Additional Options**" for future instances of SerialPlotter. This only saves accepted values, i.e., numbers for refresh rate, readings and baud rate. Since there is not way to check the Arduino name at this stage, the name is saved without doing so. If "d" (without quotes) is entered here instead, then the values are rewritten to their defaults, and these are used in the next steps.

At various steps of the process, various messages will be presented on the screen. Each one indicates the completion of a particular step and could help with the troubleshooting process (next section). If everything goes well, the following messages should be presented:

Graphs disabled, recording to CSV only. (If the user disabled graphs or failed to select any)
OR
Plotting <*selected graphs*>.

Connecting to Arduino…
Using saved device named <*saved value*>
Arduino connected via <*COM Port*> (On successful connection)
Reading from COM Port.

==========
CLOSE CHART WHEN DONE AND
PRESS ENTER TO SAVE AND TERMINATE
==========

Recording has started. Do not open CSV file. (If any data was received in the last 30 seconds and is in the correct format)

Recording ended, saving readings… (Once Enter is pressed to terminate)
Readings saved to desktop.

==========
THIS WINDOW CAN NOW BE CLOSED
==========

<*credits*>

Once the readings have completed, the data can be found on the computer desktop with the name "ASP_<*date*>_<*time*>" (Format: YYYYMMDD_HHMMSS).

Common Issues and Troubleshooting

1. "**No data received yet, please wait or press Enter to abort.**"
    a. The most common cause is that the Arduino device is not printing any information to the serial, either because the print function is not called in the (at all, or often enough) or the correct print command was not used: **Serial.println(*variable name*);**. Also ensure that the **Serial.begin(*baud rate*)** command has been entered into the Arduino's **void setup() {** function. Confirm for data using the the Arduino IDE's Serial Monitor.
    b. Another reason could be that the Arduino is yet to transmit information and the is taking time to setup and start transmitting data. SerialPlotter prints this message every 30 seconds that it doesn't receive information. Check this in the Arduino IDE's Serial Monitor for the delay between opening it and starting to print data or any other pauses in between.
    c. If you also received the message "**Multiple Arduinos found - using the first.**", then you have more than one Arduino device connected to the computer and the one printing the required data is on a larger number COM port. Abort and either disconnect the other device or manually change the required device's COM Port to a lower number in Device Manager > Ports (COM & LPT) and restart SerialPlotter.

2. "**Arduino not found.**"
    a. Check the USB connections and that the lights on the Arduino board are on. Then check that the Arduino is connected in the Device Manager.
    b. The name of the Arduino device varies between manufacturers. Check the name of your device in the Device Manager and manually enter that (even partially) in the relevant prompt in Additional Options. Save the name when prompted to avoid having to do so repeatedly.
    c. If you also received the message "**Saved device not found. Connecting to "USB Serial Device".**" check that the spelling is correct. Open the device properties in Device Manager and go to the tab Details and check the information given in Device description. Enter this (even partially) as the name of the device in **Additional Options**.
3. "**COM Port already open, close other program(s) and restart SerialPlotter.**"
    a. This one is usually due to another instance of SerialPlotter or, the Serial Monitor or Serial Plotter from the Arduino IDE being open. Close those program(s) and restart.
    b. If issue persists, disconnect all Arduino devices and reconnect them before relaunching SerialPlotter. Keep all other programs accessing the COM Ports disconnected.
4. "**Error encountered, recording aborted.**"
    a. This is most likely due to the presence of text in the serial data. Check the Arduino code for any **Serial.print("<*anything*>")**. The inverted commas indicate that the information within is a string. Ensure that only the variables are being printed to the Serial according to the proper syntax covered at the start.
    b. This could also be because of an incorrect baud rate. Ensure that the baud rate in SerialPlotter matches that of the Arduino device. This can be seen in the **Serial.begin(*baud rate*)** command in the Arduino IDE and can be set in the **Additional Options** section of SerialPlotter.
    c. Another issue is that the computer is not allowing SerialPlotter to create or access the current ASP_*.csv file required. This could happen if you try accessing the file while SerialPlotter is still writing to it or if the software does not have the required privileges. In general, wait for the program to terminate before accessing the file. If the issue persists, right-click the shortcut or executable and select "Run as administrator". To make this permanent, right-click the shortcut or executable and click on "Properties", then go to the tab "Compatibility" and check "Run this program as an administrator"; then press "OK". Windows will ask for confirmation every time you launch the program – just click on "Yes".
    d. This is a general error that can be triggered by almost anything, so if the above solutions don't work, get in touch.
5. Y versus time graph is stuck at 0, or the X versus Y graph is flat.
    a. The data for the second variable is not being sent to Serial. If this was intentional, disable all graphs except the first.
    b. If both values are being printed, ensure that the print command in the Arduino IDE is "**Serial.println(Y)**" and not just "**Serial.print(Y)**". Also open the Serial Monitor and ensure that the values are being separated by a space (" ") and not a comma or anything other symbol.
6. One graph has both values (added), the other is 0.
    a. See previous point.
7. Graph and data quality issues

a. Within SerialPlotter, the number of graphs, axes lengths and graph refresh rates affect the quality of the data collected. Lower the first two and increase the third to improve performance.
b. If the graph is lagging too far behind the current reading, try increasing the refresh rate of within **Additional Options**.
c. The size of the variables being read can also affect the performance, as it takes longer to take readings. This could also lead to data truncation but is not the only cause. Also see point 8.
d. Complex code in the Arduino, or a less powerful device like an Arduino Nano could also affect the rate at which data is transmitted to the serial. Insert timers in your Arduino code and check them to see the delays in the program.
e. The processing power of the computer also affects the speed at which the readings are converted, stored and plotted. Even with all graphs disabled and with small variables (<10 bits each), a delay of 10 milliseconds can be expected between readings.
f. Every time the graph is called, a portion of the serial data is scrapped. While this ensures that the graph's lag doesn't increase, it does reduce the quality of the stored data, especially with lower refresh rates.

8. Data is being truncated.
   a. The size of the variables is too large. Also see point 7.c.
   b. Data type is not float (like long). SerialPlotter converts all variables to floats before storing and plotting them, so large types will be truncated. Use int, float or double in the Arduino IDE.
   c. The Arduino is clearing the serial buffer with a **Serial.flush()** command. If threaded with the main code (like when using **millis()**) and not timed properly, this could truncate the readings being printed to the serial. Check the Arduino IDE Serial Monitor for truncation as well.
9. A graph is empty
   a. You sure? There is no way this could happen since SerialPlotter checks the data before plotting the graphs. C'est impossible! Get in touch if it does.

Contact Information

My name is Aashrit Gautam and I wrote this program to help with some research projects I am undertaking at the University of Wollongong. Since I am a novice coder, this program is not very sophisticated, and there may be issues with it that I may not have considered. I have only tried it on Windows 10 64-bit computers and it may not be compatible with other devices. Any feedback would be greatly appreciated, but do refer to the Troubleshooting section and check your Arduino code before contacting me at agautam@uow.edu.au.

Source Code

```
import os
import sys
import serial
import serial.tools.list_ports
import warnings
import matplotlib.pyplot as plt
import csv
import numpy as np
from timeit import default_timer as timer
import threading as th
from datetime import datetime
import pickle
import time

chart_selector_list = []
add_info = []

#Establish the path to SavedValues.pk: the saved file that contains the settings
if getattr(sys, 'frozen', False):
    application_path = sys._MEIPASS
else:
    application_path = os.path.dirname(os.path.abspath(__file__))

savefilename=os.path.join(application_path, "SerialPlotterSavedValues.pk")

#Access saved file or create it if it doesn't exist
try:
    with open(savefilename, 'rb') as fi:
        savedata = pickle.load(fi)
except:
    saved_3 = [0.025, 9600, 'USB Serial Device', 200]
    with open(savefilename, 'wb') as fi:
        pickle.dump(saved_3,fi)
    savedata = saved_3

#Reset saved file to default if it is empty for any reason, IDK
if len(savedata) != 4:
    saved_3 = [0.025, 9600, 'USB Serial Device', 200]
    with open(savefilename, 'wb') as fi:
        pickle.dump(saved_3,fi)
    savedata = saved_3

#User inputs for selecting which charts to print
def chart_selector():
    global draw_plot
    print ("\n     TYPE y AND PRESS Enter FOR Yes, OR JUST PRESS Enter TO SKIP:")
    draw_plot = input("\n       Enable live graph of data? (y/n): ")
    draw_tx = 0
    draw_ty = 0
    draw_xy = 0
    add1 = 'n'
    if draw_plot.lower() in ['y', 'yes']:
        print("\n       For serial data recieved as \"X Y\":")
        draw_ij = input("       Enable X vs Time graph?    (y/n): ")
        draw_ik = input("       Enable Y vs Time graph?    (y/n): ")
        draw_jk = input("       Enable X vs Y graph?       (y/n): ")
        print("\n")
        if draw_ij.lower() in ['y','yes']:  # These values were calculated to allow
            draw_tx = 1                     # simplification of the chart selection
        if draw_ik.lower() in ['y','yes']:  # process as ANY combination of these
            draw_ty = 2                     # digits has a unique total.
        if draw_jk.lower() in ['y','yes']:
            draw_xy = 4
```

```
    add1    = input("       View additional options?   (y/n): ") #To enable additional settings and options
    return [draw_plot,draw_tx,draw_ty,draw_xy,add1]

#Parsing the input into a list
chart_selector_list = chart_selector()

#User inputs for further settings and options
def additional_options(enable):
    global chart_selector_list
    global savedata
    enable = chart_selector_list[4]
    if enable.lower() in ['y','yes']:
        print("\n     Additional Options (press Enter to skip):")
        print("\n       Increase(>0.1 s) refresh rate if graphs or data are messy.")
        refresh   = input("       Graph refresh rate   (min=0.010, def=0.025): ")
        print("\n       High number of graph readings causes the plot to lag.\n       Enter 0 for endless mode.")
        no_reads  = input("       Maximum readings on each axis     (def=200):")
        print("\n       If required, enter the Arduino name as displayed in \"Device Manager > Ports (COM & LPT)\" on a Windows PC.\n
The default is \"USB Serial Device\".")
        cust_ard  = input("       Input Arduino name, or skip with Enter     : ")
        print("\n       The baud rate is set in the Arduino IDE within\n       \"void setup {\" using \"Serial.begin(9600);\" ")
        cust_baud = input("       Input the Arduino baud rate     (def=9600): ")
        print ("\n")
        if len(cust_ard) == 0:
            cust_ard = str(savedata[2])
    else:
        cust_ard = str(savedata[2])
    try:
        refresher = float(refresh)
    except:
        refresher = savedata[0]
        print("\n     Graph refresh rate set to", savedata[0], "seconds.")
    try:
        bauds = float(cust_baud)
    except:
        bauds = savedata[1]
        print("\n     Baud rate set to", savedata[1], "baud.")
    try:
        readss = float(no_reads)
    except:
        readss = savedata[3]
        print("\n     Axes lengths set to", savedata[3], "readings each.")
    if enable.lower() in ['y','yes']:
        pickled = input("\n       Save accepted values for future use? (y/n) or \'d\' to reset to defaults: ")
        if pickled.lower() in ['y','yes']:
            s2 = refresher
            s3 = bauds
            s4 = readss
            saved_2 = [s2,s3,str(cust_ard),s4]
            with open(savefilename, 'wb') as fi:
                pickle.dump(saved_2,fi)
            print("\n       Manual entries saved.")
        elif pickled.lower() in ['d','default', 'def']:
            refresher, bauds, cust_ard, readss = 0.025, 0.025, 9600, 'USB Serial Device', 200 #Resetting to default
            saved_3 = [0.025, 9600, 'USB Serial Device', 200]
            with open(savefilename, 'wb') as fi:
                pickle.dump(saved_3,fi)
            print("       All values reset to default.")
    return [refresher, bauds, cust_ard, readss]

#To parse the additional information
add_info = additional_options('n') #The 'n' ensures that it only runs if requested

#To calculate plot size
reads2 = 1-add_info[3]
reads3 = [0,reads2]
```

```
reason_read = min(reads3)
refresh_list = [0.01, add_info[0]]
reasonable_refresher = max(refresh_list)
#To create the plot figures based on the case number
case = chart_selector_list[1] + chart_selector_list[2] + chart_selector_list[3]
if chart_selector_list[0].lower() in ['y', 'yes']:
    fig = plt.figure()
    if case == 7:
        ax = plt.subplot(221)
        bx = plt.subplot(222)
        cx = plt.subplot(2,2,(3,4))
        print ("\n     Plotting X, Y and XY.")
    elif case == 6:
        bx = plt.subplot(121)
        cx = plt.subplot(122)
        print ("\n     Plotting Y and XY")
    elif case == 5:
        ax = plt.subplot(121)
        cx = plt.subplot(122)
        print ("\n     Plotting X and XY.")
    elif case == 4:
        cx = plt.subplot(111)
        print ("\n     Plotting XY only.")
    elif case == 3:
        ax = plt.subplot(121)
        bx = plt.subplot(122)
        print ("\n     Plotting X and Y.")
    elif case == 1:
        ax = plt.subplot(111)
        print ("\n     Plotting X only.")
    elif case == 2:
        bx = plt.subplot(111)
        print ("\n     Plotting Y only.")
    else:
        print("\n\n     Graphs were enabled, but none were selected. Disabling...")
        chart_selector_list[0] = 'n'
        print("     Graphs disabled, recording to CSV only.")
else:
    print("\n     Graphs disabled, recording to CSV only.")

#To detect the COM port for the Arduino and open it
print ("\n     Connecting to Arduino...")
try:
    BMMT_Port = [
        p.device
        for p in serial.tools.list_ports.comports()
        if add_info[2] in p.description
    ]
    if not BMMT_Port:
        print("     Using saved device named", savedata[2])
        raise IOError()
    if len(BMMT_Port) > 1:
        warnings.warn('\n     Multiple Arduinos found - using the first.')
except:
    print("\n     Saved device not found. Connecting to \"USB Serial Device\".")
    BMMT_Port = [
        p.device
        for p in serial.tools.list_ports.comports()
        if 'USB Serial Device' in p.description
    ]
    if not BMMT_Port:
        raise IOError("\n     Arduino not found.")
    if len(BMMT_Port) > 1:
        warnings.warn('\nMultiple Arduinos found - using the first.')

BMMT_Port_Str = str(BMMT_Port[0])
```

```
try:
   ser = serial.Serial(BMMT_Port_Str,add_info[1],timeout=30)
   print("      Arduino connected via", BMMT_Port_Str)
   #print("\n      Receiving serial data...")
except:
   ser.close()
   try:
      ser = serial.Serial(BMMT_Port_Str,add_info[1],timeout=30)
   except:
      print("\n      COM Port already open, close other program(s) and restart SerialPlotter.")

#Declaring variables as per the accompanying description
keep_going = True          #To interrupt the recording
cntr = 0                   #To refresh the serial lists in order to keep things manageable
time_1 = timer()           #To establish the starting time
i_list = []                #Time list
i_plot = []                #Time plotting list drawn at lower refresh rate
k_list = []                #Load list taken from the device
k_plot = []                #Load plotting list
j_list = []                #Distance list
j_plot = []                #Distance plotting list
header = ['Time (s)', 'X', 'Y']  #Top row of CSV file
suffix = datetime.now().strftime("%Y%m%d_%H%M%S")   #To add to the end of the CSV file's name
filename = 'ASP_'+suffix+'.csv'                 #Naming of the CSV file
filepath = os.path.join(os.path.join(os.environ['USERPROFILE']), 'Desktop', filename) #To find the desktop directory and save CSV
fail = 0

plt.ion()

#To create the CSV and write the header row
with open(filepath,'a', newline = '\n') as f:
      writer = csv.writer(f)
      writer.writerow(header)

#The figure making function
def show_plot(time1,dist1,load1):
   global i_plot
   global j_plot
   global k_plot
   global case
   global reason_read

   i_plot.append(time1[-1])
   j_plot.append(dist1[-1])
   k_plot.append(load1[-1])

   if case in [1,3,4,7]: #Cases 1,3,4,7
      ax.cla()
      ax.grid(True)
      ax.set_xlabel('Time (s)')
      ax.set_ylabel('X')
      #j_plot.append(dist1[-1])
      ax.plot(i_plot,j_plot,'r')

   if case in [2,3,6,7]: #Cases 2,3,6,7
      bx.cla()
      bx.grid(True)
      bx.set_ylabel('Y')
      bx.set_xlabel('Time (s)')
      #k_plot.append(load1[-1])
      bx.plot(i_plot,k_plot,'b')

   if case in [4,5,6,7]:
      cx.cla()
      cx.grid(True)
      cx.set_xlabel('X')
```

```
        cx.set_ylabel('Y')
        cx.plot(j_plot,k_plot,'g')

    plt.pause(0.00001)
    if reason_read !=0:
        i_plot = i_plot[reason_read:] #To limit the size of the graph
        j_plot = j_plot[reason_read:] #It's either this
        k_plot = k_plot[reason_read:] #or slow plotting

#The timer to update the figure
def plot_gap(time2,dist2,load2):
    global add_info
    global reasonable_refresher
    th.Timer(reasonable_refresher,plot_gap)
    ser.reset_input_buffer()  #Clears out the serial data so that only the latest info is plotted
    ser.read_until().decode('utf-8') #To read any info mangled by the flush so that read_data() doesn't
    show_plot(time2,dist2,load2)

#To terminate the program
def key_capture_thread():
    global keep_going
    input()
    keep_going = False

#To transfer all list data to the file at a particular number of readings
def write_csv (time3, dist3, load3):
    i_array = np.array(time3)
    j_array = np.array(dist3)
    k_array = np.array(load3)
    i_vert = i_array.reshape(-1,1)
    j_vert = j_array.reshape(-1,1)
    k_vert = k_array.reshape(-1,1)
    final_array = np.concatenate((i_vert,j_vert,k_vert),axis=1)
    with open(filepath,'a', newline = '\n') as f:
        writer = csv.writer(f)
        writer.writerows(final_array)
    #ser.reset_input_buffer()  #Clears out the serial data so that only the latest info is plotted

#To read the serial data and parse it into usable information
def read_data():
    th.Thread(target=key_capture_thread, args=(), name='key_capture_thread', daemon=True).start()
    while keep_going:
        global cntr
        global i_list
        global j_list
        global k_list
        global add_info
        time_2 = timer() - time_1
        # if chart_selector_list[0].lower() in ['y', 'yes']:
        #     ser.reset_input_buffer()
        #     s = ser.read_until().decode('utf-8')
        s = ser.read_until().decode('utf-8')
        # s = ser.readline().decode('utf-8')
        s_list=s.split()
        # if len(s_list)!=2:
        #     s = ser.readline().decode('utf-8')
        #     s_list=s.split()
        if len(s_list)!=0:
            if cntr == 0:
                print("      Recording has started. Do not open CSV file.")
                cntr = 1
            j = float(s_list[0])
            if len(s_list) == 1:  #Write 0 to the third column if no input is available
                k = 0
            else:
                k = float(s_list[1])
```

```
            #k = float(s_list[1])
            i = round(float(time_2),3)
            i_list.append(i)
            j_list.append(j)
            k_list.append(k)
            cntr=cntr+1  #To check the number of reading loops executed to clear memory (Line 366)
            if chart_selector_list[0].lower() in ['y', 'yes']:  #Enables the graph if user said yes
               plot_gap(i_list,j_list,k_list)
            if cntr == 10:  #To refresh the serial lists to save memory
               write_csv(i_list,j_list,k_list)
               i_list = []    #Empties all
               j_list = []    #the lists
               k_list = []    #for speed
               cntr = 1  #Resets the counter
         elif len(s_list)==0:
            print("     No data received yet, please wait or press Enter to abort.")

#Starts the reading and plotting
print ("     Reading from COM Port.")
print("\n\n=============================================================================================================")
if chart_selector_list[0].lower() in ['y', 'yes']: #Enables the chart closing reminder
   print("                          CLOSE CHART WHEN DONE AND")
print("                        PRESS ENTER TO SAVE AND TERMINATE")
print("=============================================================================================================\n")
ser.reset_input_buffer()  #Clears all existing data in the serial for fresh readings
ser.reset_output_buffer()
try:
   read_data()  #To call the thread that enables the reading function
   print("\n     Recording ended, saving readings...")
   write_csv(i_list,j_list,k_list)
   print("     Readings saved to desktop.")
except:
   ser.close() #In case of error, closes port
   print("\n     Error encountered, recording aborted.\n     Check Instructions > Common Issues and Troubleshooting.")

print("\n\n=============================================================================================================")
print("                          THIS WINDOW CAN NOW BE CLOSED")
print("=============================================================================================================\n")
#read_data()  #To call the reading loop (disabled for threading)
ser.close()  #Closes the serial when reading ends

print("\n     Made by Aashrit Gautam for free and open source use.\n\n     Python file included in folder, feel free to tinker and send\n
improvements to agautam@uow.edu.au")
time.sleep(5)
```