```python
#! /usr/bin/env python
#  -*- coding: utf-8 -*-
#
# Support module generated by PAGE version 6.2
#  in conjunction with Tcl version 8.6
#    Jun 19, 2022 01:33:03 PM PDT  platform: Windows NT

import sys,os
import serial
import serial.tools.list_ports
import time
from time import sleep
from pynput import keyboard
import threading
import matplotlib
import matplotlib.pyplot as plt
#from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg)
import numpy as np
''' PC specific '''
from pynput import keyboard

''' RPi 4 specific '''
#bcs import RPi.GPIO as GPIO

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

try:
    import ttk
    py3 = False
except ImportError:
    import tkinter.ttk as ttk
    py3 = True

greenbutton = False
jumpFlag = False

state = 0

def init(top, gui, *args, **kwargs):
    global w, top_level, root,listener
    w = gui
    top_level = top
    root = top
    root.protocol("WM_DELETE_WINDOW", on_closing)

    #for keyboard
    listener = keyboard.Listener(on_press=on_press)
    listener.start()

    ''' For RPi4 to dectect button press event '''
#     GPIO.setmode(GPIO.BCM)
#     GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#     GPIO.add_event_detect(23,GPIO.RISING,callback=Button)
#     GPIO.setup(24, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#     GPIO.add_event_detect(24,GPIO.RISING,callback=Button)

    main_app() #start the main app

```

```
#called when the button is pressed
# def Button(event):
#     global greenbutton,redbutton#, state
#     sleep(.005) #debounce 5ms.
#     if GPIO.input(event) != 0:
#         if event == 23:
#             greenbutton = 1
#             #print('green button',greenbutton)
#         if event == 24:
#             redbutton = 1
#             #print('red button',redbutton)

#called if a key is pressed (if listener was 'started'

def on_press(key):
    global button, state
    if key :
        greenbutton =1
        #state = state ^1
        print("key pressed", button,state)

def main_app():
    global ser,offset,a,state,fig,plt,greenbutton,redbutton
    #print('main app')
    ser = open_ser()
    a=App()
    redbutton=0
    greenbutton=0
    state = 0
    offset=calibrate() #make sure no weight is on the scale.
    #for j in range(0,1):
    while True:
        #two commands required to close the plt figure
        plt.close('all')
        plt.close(plt.figure())
        #print('plt.close()')

        window1()
        plot() #plot returns when external event occurs.
        state = 1
        a.xpos = a.xstart
        w=check_weight() #make sure they stand on scale
        #w=True #debug
        if w == True: #proceed if standing on scale
            #state = 1
            window2() #ready, set, go
            plot()
            #print('ready to analyze, Button=',greenbutton)
            err = analyze() #this is executed before previous function is done.
            window3() ##print results
            a.xpos= a.xstart
                ##print(state) #just occurs once
#             else:
#                 greenbutton=0
#                 state=0
                #window1()
        else:#if w == False:
            window4() #tell user to stand on scale
            #greenbutton=0
            #print('stand on scale')
            #state=0
```

```python
        state = 0
        plot() #and start over
        #state=0
        #print(" the end--go to beginning")
        #print(state) #state=3 after normal process


def check_weight(): #make sure someone is standing on the weight
    global state,offset
    calib = 0
    ser.read_all() #clear the serial buffer
    for i in range(0,10):
        try:
            raw=ser.readline()
            raw=raw.decode()
            raw=float(raw)
            calib += raw
            ##print(raw,calib)
        except:
            calib += calib

    calib=calib/10 #get average
    #print('calib',calib,'offset',offset)
    if abs(calib-offset)<6:
        #print("not on scale")
        return False
    return True

def plot():
    global ser,a,state,scount
    global data,datap,i,redbutton,greenbutton,sample_size,offset#,jump#,button,i
    greenbutton=0
    i=0
    w1=0
    w2=0
    same=False #state when to measure consecutive samples after a jump
    scount=0 #number of consecutive values that are close to same value
    data = []
    datap =[]
    ser.read_all()
    #make
    while True:

        if greenbutton == 1: #in range(1,3):
            #print('quite plot()')
            greenbutton =0
            return

        raw=ser.readline()
        try:
            raw=raw.decode()
            ##print(raw)
            raw=float(raw)
            ##print(raw)
        except:
            print("Plot Error")
            raw=offset
            pass

        w1=raw

        if state ==1:
            #print(state,raw-offset)
            if abs(raw-offset)<3: #check if the scale goes close to zero after jumping
                same = True
                #print("same",same)

            if same == True: #now measure consecutive values and exit if true.
                if abs(w1-w2) < 5:
                    scount+=1
                    #print(scount)
                    if scount == 25:
                        print('analyzing...')
                        window5()
                    if scount >= 100:
                        #jump=True
                        #state=0
                        print("finished analyzing")
                        return
                else:
                    scount=0 #reset count since values are not same.
                    print("reset scount",same)#jump=False

        w2=(w2+w1)/2. #update the filter
        point = 450-raw + offset
        a.addPoint(point)
        if i<= a.wwidth:
            data.append(raw)
            #datap.append(point)
        else:
            data=[]
            datap=[]
            i=-1
        i+=1
###
### Calibrate scale. Should have no weight on the scale or error will return
###
def calibrate():
    #take a number of samples and average to find base line
    default_calib = 962#225
    calib = 0
    deviation = 20 #max deviation from "zero" weight value
    ser.read_all() #clear the serial buffer
    for i in range(0,10):
        try:
            raw=ser.readline()
            raw=raw.decode()
            raw=float(raw)
            calib += raw
            ##print(raw,calib)
        except:
            calib += calib

    calib=calib/10 #get average
    #print("Calib =",calib)
    #check if calib is +/- deviation from normal value
    if abs(default_calib-calib) >= deviation:# <= calib <= default_calib+deviation:
        return calib
    else:
        #print("calib error", calib)
        return calib#default_calib

#moving average filter array c of len n
def moving_average(c, n=8):
    ret = np.cumsum(c)
    ret[n:] = ret[n:] - ret[:-n] #normal
    mv=ret/n #return same array size
    mv[0:n-1]=mv[n] #fill end with mv[n]
    return mv

def open_ser():
    baudrate = 115200#57600
    comport = 'COM1'
    ports = list(serial.tools.list_ports.comports())
    for p in ports:
        b=str(p)
        #print(b)
        a=b.find("USB")
        #aa=b.find("AMA")
        if a >=0:
            comport=b[:a+4]
        #if aa >=0:
        #    comport=b[:aa+4]

    #print("USB at ",comport)
    #print('baudrate= ',baudrate)
    #comport = '/dev/ttyUSB0'
    #comport = '/dev/ttyUSB1'
    #comport = '/dev/ttyAMA0'
    ser = serial.Serial()
    ser.baudrate = baudrate
    ser.port = comport
    ser.timeout = 2
    try:
        ser.open()
    except:
        #print("Error.  Incorrect COM port or baud rate")
        sys.exit()
    return ser

def window1():

    w.Label1.configure(text='How High Can You Jump?')
    w.Label2.configure(text='How Long Can You Stay in the Air?')
    w.Label4.configure(text='Stand on the Scale and\rPress the GREEN Button')
    w.Label3.configure(text='')
    return

def window2():
    global greenbutton
    jumpFlag = True
    #print('jumpFlag',jumpFlag)
    td = 1000
    w.Label1.configure(text='* Prepare to Jump *')
    w.Label2.configure(text='')
    w.Label3.configure(text='')
    w.Label4.configure(text='')
#     return
    root.after(td,lambda:w.Label2.configure(text='READY'))
    td=td+1000
    root.after(td,lambda:w.Label3.configure(text='SET'))
    td=td+1000
    root.after(td,lambda:w.Label4.configure(text='JUMP!'))
    td=td+200
    ##print('td',td)
    jumpFlag = False
    #print('jumpFlag',jumpFlag)

    return

def window3():
    global jumpInch,jumpCM,AirTime

    #w.Label1.configure(text='RESULTS')
    w.Label1.configure(text='Air Time = '+str(AirTime)+' Seconds')
    w.Label2.configure(text='Height = '+str(jumpInch)+' Inches')
    w.Label3.configure(text='Height = '+str(jumpCM)+' Centimeters')
    w.Label4.configure(text='Press the GREEN Button\r to Continue')# or wait 10 sec.')
    root.after(10000,lambda:foo())

def window4():
    w.Label1.configure(text='')
    w.Label3.configure(text='')
    w.Label4.configure(text='Press the GREEN Button\r to Continue')
    w.Label2.configure(text='Please Stand on the Scale')

def window5():
    w.Label1.configure(text='')
    w.Label3.configure(text='Analyzing...')
    w.Label4.configure(text='Please Stand Still')
    w.Label2.configure(text='')


def foo():
    global state, greenbutton
    #print("exit from Window3")
    state=0
    greenbutton=0



def analyze():
    global i,dataNP,offset,data, jumpInch,jumpCM,AirTime
    sample_rate=50.
    sample_size = i

    m0 = np.argmin(data) #find min value in array. ToDo fix so first min not always picked
    #print(m0)
    m0=int(m0-.005*m0) #go back 0.5% of y axis.

    #print("m0", m0)
    thresh = data[m0]
    m1=m0
    #print('m0,m1 =',m0,m1,data[m0]-offset,data[m1]-offset,offset, thresh)
    while data[m1] <= thresh: #walk through min values to find endpoint
        ##print('m1',m1,i)
        m1 += 1
        if m1 >= i:
            m1=i-1
            break


    m1 -= 1

    #m1=int(m1+.005*m1) #move 5% beyond endpoint
    #m1=m1-1
    #print('m0,m1 =',m0,m1,data[m0]-offset,data[m1]-offset,offset)
    #print(len(data))
    t=(m1-m0)/2
    t=t/sample_rate
    #print('time =', t)
    h = (t**2)*9.81/2

    #print('Jump Height =',round(h,3), 'meters ', round(h*100,3),
    #      'cm', round(h*39.37,2),'inches')
    #print('Air Time =', round(t*2,3),'seconds')
    jumpInch=round(h*39.37,2)
    jumpCM = round(h*100,2)
    AirTime = round(t*2,2)
    #check to make sure data makes sense
    if jumpInch >24 :
        #print("bad data")
        return -1
    '''Plot meaningfull data using matplotlib '''

    dataNP=np.array(data,dtype=float)-offset
    dataNP=moving_average(dataNP,1)
    plt.rcParams['toolbar'] = 'None'
    plt.ion()
    fig, ax = plt.subplots()
    fig.canvas.manager.window.move(220,220)
    ax.set_frame_on(False)
    xaxis = np.arange(i,i+sample_size)/50
    ax.plot(xaxis,dataNP)

    ax.plot(xaxis[m0],dataNP[m0],'ro') #plot the two minima values
    ax.plot(xaxis[m1],dataNP[m1],'ro')

    ax.set(xlabel='time (s)', ylabel='newtons', title='Your "Flight" Profile')
    ax.grid()
    plt.tight_layout()

    plt.show()
    plt.pause(0.1)
    return 0


def on_closing():

#     listener.stop()
    #print("system exit")
    top_level.destroy()

'''
def destroy_window():
    # Function which closes the window.
    global top_level
    top_level.destroy()
    top_level = None
'''

class App:
    wwidth = 800
    xstart=0
    samplerate=50
    def __init__(self):
        self.xpos=self.xstart#0 #change to 75 but runs out of range in addPoint
        self.line1avg=0
        self.c = tk.Canvas(w.Frame1, width=self.wwidth, height=512) #place canvas in Frame1
        #self.c.tk.call('tk','scaling',2.5)
```

```python
        self.c.pack()
        self.white()

    def __del__(self):
        print("removed")

    def pause(self):
        #root.forget(self.c.withdraw())
        pass

    def white(self):
        #print("white")
        self.lines=[]
        self.lastpos=0
        self.c.create_rectangle(0, 0, self.wwidth, 500, fill="black")

        for y in range(0,10):#(-50,512,50): #draw Y labels
            y=y*50
            #print("y=", y)
            self.c.create_line(self.xstart, y, self.wwidth, y, fill="#999999",dash=(4, 4)) #create y grid, was #333333
            #if y<425:
            #    self.c.create_text(5, 450-y, fill="#ffffff", text=str(y//2), anchor="w") #create y labels, was #999999

        for x in range(self.xstart,self.wwidth,self.samplerate): #x axis labels
            self.c.create_line(x, 0, x, 512, fill="#999999",dash=(4, 4)) #create x grid
            #self.c.create_text(x-25, 500-10, fill="#ffffff", text=str((x-75)/100)+"s", anchor="w")

        self.lineRedraw=self.c.create_line(0, self.wwidth, 0, 0, fill="red",width=2) #define the scroll line

        #self.lines1text=self.c.create_text(self.wwidth-3, 10, fill="#00FF00", text=str("9 DoF"), anchor="e")
        for x in range(self.wwidth):
            self.lines.append(self.c.create_line(x, 0, x, 0, fill="#00FF00",width=3))
            #pass
        self.xpos=self.xstart

    def addPoint(self,val):
        self.c.coords(self.lines[self.xpos],(self.xpos-1,self.lastpos,self.xpos,val))
        self.c.coords(self.lineRedraw,(self.xpos+1,0,self.xpos+1,self.wwidth)) #draw the vertical line
        self.lastpos= val
        self.xpos+=1 #sets span

        if self.xpos>=self.wwidth:
            ##print("blah")
            self.xpos=self.xstart#0

        root.update()

if __name__ == '__main__':
    import weigh_page
    weigh_page.vp_start_gui()
```