*#code for spectrophotometer GUI…..*

```
from tkinter import *
# from Canvas import*
import numpy as np
import RPi.GPIO as GPIO
import matplotlib

matplotlib.use("TkAgg")
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
# import spectro_hardware_support_new
import math
import smbus
import time
import threading
# import tkMessageBox
from tkinter import messagebox
import openpyxl
from os import listdir
import os
import xlwt

LARGE_FONT = ("Verdana", 12)
SMALL_FONT = ("Verdana", 10)

# GPIO numbering
GPIO.setmode(GPIO.BOARD)
# Close warnings
GPIO.setwarnings(False)


# Function to convert the wavelength into RGB values
def (wavelength):
    # Gamma Correction Limit
    gamma = 0.8

    w = float(wavelength)

    # colour
    if w >= 380 and w <= 440:
        attenuation = 0.3 + 0.7 * (w - 380) / (440 - 380)
        R = ((-(w - 440) / (440 - 380)) * attenuation) ** gamma
```

```
            G = 0.0
            B = (1.0 * attenuation) ** gamma
        elif w >= 440 and w <= 490:
            R = 0.0
            G = ((w - 440) / (490 - 440)) ** gamma
            B = 1.0
        elif w >= 490 and w <= 510:
            R = 0.0
            G = 1.0
            B = (-(w - 510) / (510 - 490)) ** gamma
        elif w >= 510 and w <= 580:
            R = ((w - 510) / (580 - 510)) ** gamma
            G = 1.0
            B = 0.0
        elif w >= 580 and w <= 645:
            R = 1.0
            G = (-(w - 645) / (645 - 580)) ** gamma
            B = 0.0
        elif w >= 645 and w <= 750:
            attenuation = 0.3 + 0.7 * (750 - w) / (750 - 645)
            R = (1.0 * attenuation) ** gamma
            G = 0.0
            B = 0.0
        else:
            R = 0.0
            G = 0.0
            B = 0.0
        R *= 255
        G *= 255
        B *= 255
        R = int(R)
        G = int(G)
        B = int(B)

        return [R, B, G]


class Window(Frame):

    # Define settings upon initialization. Here you can specify
    def __init__(self, master=None):

        self.frame = None
        self.panel = None

        # parameters that you want to send through the Frame
class.
```

```
        Frame.__init__(self, master)

        # reference to the master widget, which is the tk window
        self.master = master

        self.baseline_val = [] * 371
        self.solution_val = [] * 371
        self.absorption = []
        self.wave_len = []

        # Define pins
        self.red = 36
        self.green = 40
        self.blue = 38

        # Define pins as Output
        GPIO.setup(self.red, GPIO.OUT)
        GPIO.setup(self.green, GPIO.OUT)
        GPIO.setup(self.blue, GPIO.OUT)

        # Frequency for PWM
        self.Freq = 100

        # Defining the pins that are going to be used with PWM
        self.RED = GPIO.PWM(self.red, self.Freq)
        self.GREEN = GPIO.PWM(self.green, self.Freq)
        self.BLUE = GPIO.PWM(self.blue, self.Freq)

        self.book = openpyxl.Workbook()

        # with that, we want to then run init_window, which
doesn't yet exist
        self.init_window()

    # Function for the creation of Initial Window of GUI
    def init_window(self):
        # changing the title of our master widget
        self.master.title("GUI")

        # packing the frame
        self.pack(fill=BOTH, expand=1)
        self.title = Label(self, text="SPECTROPHOTOMETER(DIC)",
font="Verdana 10 bold", fg="blue", bg="yellow")
        self.title.place(x=400, y=5)
        self.baseline = Label(self, text="Set Base Line",
font=LARGE_FONT)
        self.baseline.grid(row=0, padx=5, pady=0)
```

```
        self.base_btn = Button(self, text="Start", height=2,
width=4, fg="red", bg="black")
        self.base_btn.configure(command=self.baseline_thread)
        self.base_btn.grid(row=1, column=0, pady=1)

        self.base_timer = StringVar()
        self.base_timer.set("Time\nLeft:")

        self.base_timer_label = Label(self,
textvariable=self.base_timer, font=SMALL_FONT)
        self.base_timer_label.grid(row=1, column=1, pady=1)

        self.solution = Label(self, text="Start Solution Test",
font=LARGE_FONT)
        self.solution.grid(row=2, padx=5, pady=1)

        self.solution_btn = Button(self, text="Start", height=2,
width=4, fg="red")

self.solution_btn.configure(command=self.solution_thread)
        self.solution_btn.grid(row=3, column=0, pady=1)

        self.solution_timer = StringVar()
        self.solution_timer.set("Time\nLeft:")

        self.solution_timer_label = Label(self,
textvariable=self.solution_timer, font=SMALL_FONT)
        self.solution_timer_label.grid(row=3, column=1, pady=1)

        self.result = Label(self, text="Plot Results",
font=LARGE_FONT)
        self.result.grid(row=4, padx=5, pady=1)

        self.result_btn = Button(self, text="Plot", height=2,
width=4, fg="red", bg="black")
        self.result_btn.configure(command=self.graph)
        self.result_btn.grid(row=5, padx=5, pady=1)

        self.reset_exp = Label(self, text="Reset Experiment",
font=LARGE_FONT)
        self.reset_exp.grid(row=6, padx=5, pady=1)

        self.reset_exp_btn = Button(self, text="Reset",
height=2, width=4, fg="red", bg="black")
        self.reset_exp_btn.configure(command=self.reset_fun)
        self.reset_exp_btn.grid(row=7, padx=5, pady=1)
```

```
        self.save_file = Label(self, text="Save Experiment",
font=LARGE_FONT)
        self.save_file.grid(row=8, padx=5, pady=1)

        self.save_file_btn = Button(self, text="Save", height=2,
width=4, fg="red", bg="black")
        self.save_file_btn.configure(command=self.save_fun)
        self.save_file_btn.grid(row=9, padx=5, pady=1)

        self.open_file = Label(self, text="Open Experiment",
font=LARGE_FONT)
        self.open_file.grid(row=10, padx=5, pady=1)

        self.open_file_btn = Button(self, text="Select",
height=2, width=4, fg="red", bg="black")
        self.open_file_btn.configure(command=self.open_fun)
        self.open_file_btn.grid(row=11, padx=5, pady=1)

        self.power_off = Label(self, text="Power Off",
font=LARGE_FONT)
        self.power_off.grid(row=12, padx=5, pady=1)

        self.power_off_btn = Button(self, text="Select",
height=2, width=4, fg="red", bg="black")
        self.power_off_btn.configure(command=self.power_off_fun)
        self.power_off_btn.grid(row=13, padx=5, pady=0)

        f = Figure(figsize=(5.5, 4), dpi=100)
        self.a = f.add_subplot(111)
        self.a.set_title("Wavelength vs Absorption Graph")
        self.a.set_xlabel("Wavelength")
        self.a.set_ylabel("Absorption")
        self.a.set_xlim(xmin=370, xmax=760)
        # a.set_ylim(ymin= ,ymax=)

        self.Canvas1 = Canvas(self)
        self.Canvas1 = FigureCanvasTkAgg(f, self)
        self.Canvas1.draw()
        self.Canvas1.get_tk_widget().place(x=235, y=40)

        toolbarFrame = Frame(self)
        toolbarFrame.place(x=235, y=5)
        # toolbar = NavigationToolbar2Tk(self.Canvas1,
toolbarFrame)

    def baseline_thread(self):
```

```
        thread = threading.Thread(target=self.baseline_fun)

        thread.start()

        t = threading.Thread(target=self.baseline_timer_count)
        t.start()

    def solution_thread(self):
        thread = threading.Thread(target=self.solution_fun)

        thread.start()

        t = threading.Thread(target=self.solution_timer_count)
        t.start()

    def baseline_timer_count(self):
        for i in range(188):
            val = "Time\nLeft:" + str(187 - i)
            self.base_timer.set(val)
            time.sleep(1)

        messagebox.showinfo("Baseline", "Baseline Reading
Complete!")

    def solution_timer_count(self):
        for i in range(188):
            val = "Time\nLeft:" + str(187 - i)
            self.solution_timer.set(val)
            time.sleep(1)

        messagebox.showinfo("Solution", "Solution Reading
Complete!")

    # TSL Sensor Values for BASELINE
    def baseline_fun(self):
        # print("Enter the Wavelength (in nm):")
        # wl=int(input())
        wl = 750
        while (wl != 379):
            [R, G, B] = wav2RGB(wl)
            print("Wavelength :" + str(wl) + " nm --> " + "[R:"
+ str(R) + " G:" + str(G) + " B:" + str(B) + "]")
            self.RED.start((R * 100) / 255)
            self.GREEN.start((G * 100) / 255)
            self.BLUE.start((B * 100) / 255)
            bus = smbus.SMBus(3)
            # TSL2561 address, 0x39(57)
```

```
            # Select control register, 0x00(00) with command
register, 0x80(128)
            #        0x03(03)    Power ON mode
            # bus.write_byte_data(0x39, 0x00 | 0x80, 0x00)
            bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)
            # TSL2561 address, 0x39(57)
            # Select timing register, 0x01(01) with command
register, 0x80(128)
            #        0x02(02)    Nominal integration time =
402ms
            bus.write_byte_data(0x39, 0x01 | 0x80, 0x02)
            time.sleep(0.5)
            # Read data back from 0x0C(12) with command
register, 0x80(128), 2 bytes
            # ch0 LSB, ch0 MSB
            data = bus.read_i2c_block_data(0x39, 0x0C | 0x80, 2)
            # Read data back from 0x0E(14) with command
register, 0x80(128), 2 bytes
            # ch1 LSB, ch1 MSB
            data1 = bus.read_i2c_block_data(0x39, 0x0E | 0x80,
2)
            # Convert the data
            ch0 = data[1] * 256 + data[0]
            ch1 = data1[1] * 256 + data1[0]

            # set baseline
            self.baseline_val.append(ch0 - ch1)
            print("Visible Value :%d lux" % (ch0 - ch1))
            wl = wl - 1
        self.baseline_val.reverse()

    # TSL Sensor Values for SOLUTION
    def solution_fun(self):
        # print("Enter the Wavelength (in nm):")
        # wl=int(input())
        wl = 750
        while (wl != 379):
            [R, G, B] = wav2RGB(wl)
            print("Wavelength :" + str(wl) + " nm --> " + "[R:"
+ str(R) + " G:" + str(G) + " B:" + str(B) + "]")
            self.RED.start((R * 100) / 255)
            self.GREEN.start((G * 100) / 255)
            self.BLUE.start((B * 100) / 255)
            bus = smbus.SMBus(3)
            # TSL2561 address, 0x39(57)
            # Select control register, 0x00(00) with command
register, 0x80(128)
```

```
            #        0x03(03)    Power ON mode
            bus.write_byte_data(0x39, 0x00 | 0x80, 0x00)
            bus.write_byte_data(0x39, 0x00 | 0x80, 0x03)
            # TSL2561 address, 0x39(57)
            # Select timing register, 0x01(01) with command
register, 0x80(128)
            #        0x02(02)    Nominal integration time =
402ms
            bus.write_byte_data(0x39, 0x01 | 0x80, 0x02)
            time.sleep(0.5)
            # Read data back from 0x0C(12) with command
register, 0x80(128), 2 bytes
            # ch0 LSB, ch0 MSB
            data = bus.read_i2c_block_data(0x39, 0x0C | 0x80, 2)
            # Read data back from 0x0E(14) with command
register, 0x80(128), 2 bytes
            # ch1 LSB, ch1 MSB
            data1 = bus.read_i2c_block_data(0x39, 0x0E | 0x80,
2)
            # Convert the data
            ch0 = data[1] * 256 + data[0]
            ch1 = data1[1] * 256 + data1[0]

            # set solution
            self.solution_val.append(ch0 - ch1)
            print("Visible Value :%d lux" % (ch0 - ch1))
            wl = wl - 1
        self.solution_val.reverse()

    def graph(self):
        del self.absorption[:]
        del self.wave_len[:]
        for i in range(len(self.solution_val)):
            self.absorption.append(-
round(math.log10(self.solution_val[i] /
float(self.baseline_val[i])), 3))

        for i in range(380, 751):
            self.wave_len.append(i)

        print(self.wave_len)

        print(len(self.wave_len), len(self.absorption))

        self.a.plot(self.wave_len, self.absorption)

        self.Canvas1.draw()
```

```
        sheet = self.book.worksheets[0]
        sheet.title = 'Sheet 1'

        for i in range(1, 372):
            sheet.cell(row=i, column=1).value = self.wave_len[i
- 1]
            sheet.cell(row=i, column=2).value =
self.baseline_val[i - 1]
            sheet.cell(row=i, column=3).value =
self.solution_val[i - 1]
            sheet.cell(row=i, column=4).value =
self.absorption[i - 1]

        del self.solution_val[:]
        del self.baseline_val[:]
        del self.absorption[:]
        del self.wave_len[:]

    def reset_fun(self):
        self.Canvas1.get_tk_widget().destroy()

        f = Figure(figsize=(5.5, 4), dpi=100)
        self.a = f.add_subplot(111)
        self.a.set_title("Wavelength vs Absorption Graph")
        self.a.set_xlabel("Wavelength")
        self.a.set_ylabel("Absorption")
        self.a.set_xlim(xmin=370, xmax=760)
        # a.set_ylim(ymin= ,ymax=)

        self.Canvas1 = Canvas(self)
        self.Canvas1 = FigureCanvasTkAgg(f, self)
        self.Canvas1.draw()
        self.Canvas1.get_tk_widget().place(x=235, y=40)

        toolbarFrame = Frame(self)
        toolbarFrame.place(x=235, y=5)
        # toolbar = NavigationToolbar2Tk(self.Canvas1,
toolbarFrame)

    def save_fun(self):
        top = self.top = Toplevel(self.master)

        x = (self.master.winfo_screenwidth() / 2) - 400
        y = (self.master.winfo_screenheight() / 2) - 240

        top.geometry('%dx%d+%d+%d' % (180, 100, x, y))
```

```
        top.resizable(False, False)

        self.l = Label(top, text="Enter File Name",
font=LARGE_FONT)
        self.l.grid(row=1, padx=5, pady=3)

        self.e = Entry(top)
        self.e.grid(row=2, padx=5, pady=3)

        self.b = Button(top, text="Submit",
command=self.clean_pop_up)
        self.b.grid(row=3, padx=5, pady=3)

    def clean_pop_up(self):
        path = '/home/pi/Readings/'

        file_name = str(self.e.get())
        self.book.save(path + file_name + '.xlsx')

        self.top.destroy()

    def list_files(self, directory, extension):
        a = []
        for f in listdir(directory):
            if f.endswith('.' + extension):
                a.append(f)

        return a

    def open_fun(self):
        files = self.list_files('/home/pi/Readings/', 'xlsx')
        top = self.top = Toplevel(self.master)

        x = (self.master.winfo_screenwidth() / 2) - 400
        y = (self.master.winfo_screenheight() / 2) - 240

        top.geometry('%dx%d+%d+%d' % (180, 200, x, y))

        top.resizable(False, False)

        self.b = Button(top, text="Open",
command=self.open_file_fun)
        self.b.pack(side=BOTTOM, fill="x")

        self.bt = Button(top, text="Plot",
command=self.plot_file_fun)
```

```
        self.bt.pack(side=BOTTOM, fill="x")

        self.listNodes = Listbox(top)
        self.listNodes.pack(side="left", fill="y")

        self.scrollbar = Scrollbar(top, orient="vertical")
        self.scrollbar.config(command=self.listNodes.yview)
        self.scrollbar.pack(side="right", fill="y")

        self.listNodes.config(yscrollcommand=self.scrollbar.set)

        files.sort()

        for f in files:
            self.listNodes.insert(END, f)

    def plot_file_fun(self):
        name = self.listNodes.get(self.listNodes.curselection())
        print(name)
        del self.solution_val[:]
        del self.baseline_val[:]
        del self.absorption[:]

        wb = openpyxl.load_workbook('/home/pi/Readings/' + name)
        worksheet = wb['Sheet 1']

        for i in range(1, 372):
            self.baseline_val.append(worksheet.cell(row=i,
column=2).value)
            self.solution_val.append(worksheet.cell(row=i,
column=3).value)

        self.graph()

        self.top.destroy()

    def open_file_fun(self):
        name = self.listNodes.get(self.listNodes.curselection())

        del self.solution_val[:]
        del self.baseline_val[:]
        del self.absorption[:]

        wb = openpyxl.load_workbook('/home/pi/Readings/' + name)
        worksheet = wb['Sheet 1']

        for i in range(1, 372):
```

```
            self.wave_len.append(worksheet.cell(row=i,
column=1).value)
            self.baseline_val.append(worksheet.cell(row=i,
column=2).value)
            self.solution_val.append(worksheet.cell(row=i,
column=3).value)
            self.absorption.append(worksheet.cell(row=i,
column=4).value)

        self.top.destroy()

        self.open_file_fun_utility()

    def open_file_fun_utility(self):
        top = self.top = Toplevel(self.master)

        x = (self.master.winfo_screenwidth() / 2) - 400
        y = (self.master.winfo_screenheight() / 2) - 240

        top.geometry('%dx%d+%d+%d' % (540, 200, x, y))

        top.resizable(False, False)

        self.listNodes = Listbox(top, width=65)
        self.listNodes.pack(side="left", fill="y")

        self.scrollbar = Scrollbar(top, orient="vertical")
        self.scrollbar.config(command=self.listNodes.yview)
        self.scrollbar.pack(side="right", fill="y")

        self.listNodes.config(yscrollcommand=self.scrollbar.set)

        self.listNodes.insert(END,
                              "Wavelength     Power
Intensity(Baseline)     Power Intensity(Solution)
Absorption")

        for i in range(len(self.baseline_val)):
            self.listNodes.insert(END, '      ' +
str(self.wave_len[i]) + '                          ' + str(
                self.baseline_val[i]) + '
' + str(
                self.solution_val[i]) + '
' + str(self.absorption[i]))

    def power_off_fun(self):
        os.system('sudo shutdown -h now')
```

```
def main():
    root = Tk()
    # root.geometry('%dx%d+%d+%d' % (800, 480, 0, -30))
    root.geometry("1000x600")
    root.resizable(False, False)
    root.title("DIC")

    canva = Canvas(root, bg="blue", height=150, width=152)
    canva.place(x=100, y=100)
    book = xlwt.Workbook()
    sheet = book.add_sheet('Sheet 2')
    sheet.write(0, 0, 'Wavelength')
    sheet.write(0, 1, 'Red')
    sheet.write(0, 2, 'Green')
    sheet.write(0, 3, 'Blue')
    sheet.write(0, 4, 'Visible1')
    sheet.write(0, 5, 'IR1')
    sheet.write(0, 6, 'Full Spectrum1')

    # creation of an instance
    app = Window(root)
    # mainloop
    root.mainloop()


if __name__ == '__main__':
    main()
```