*#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()