**Source Code (Asha N)**

Highlighted portion in the below script(Server code) is the implemented by Asha N

import tkinter as tk

from tkinter import Frame, Button, Canvas

import matplotlib

matplotlib.use('TkAgg')

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from matplotlib.figure import Figure

from matplotlib.lines import Line2D

import numpy as np

from math import pi

import math

import pickle

import pylab

from scipy.signal import butter, lfilter

from scipy.signal import freqz

from scipy.ndimage import gaussian\_filter1d

import socket

import time

import datetime as dt

import random

from collections import deque

numberconvert = 0

HEADERSIZE =100

numberlist = []

temp=[]

List\_type=[]

list\_dc=[]

type\_of\_measurement = ""

range1=0

list1=[]

input\_volt=0

#attenuation\_factor=0.014

fs=1000

k=0

def convert\_into\_eight\_bits(value):

    MSB = (value >> 8) & 0xff

    LSB = value & 0xff

    return MSB,LSB

def butter\_bandpass(lowcut, highcut, fs, order=5):

    nyq = 0.5 \* fs

    low = lowcut / nyq

    high = highcut / nyq

    b, a = butter(order, [low, high], btype='band')

    return b, a

def butter\_bandpass\_filter(data, lowcut, highcut, fs, order=5):

    b, a = butter\_bandpass(lowcut, highcut, fs, order=order)

    y = lfilter(b, a, data)

    return y

def butter\_lowpass(lowcut, fs, order=5):

    nyq = 0.5 \* fs

    low = lowcut / nyq

    b, a = butter(order, [low], btype='low')

    return b, a

def butter\_lowpass\_filter(data, lowcut, fs, order=5):

    b, a = butter\_lowpass(lowcut, fs, order = order)

    y = lfilter(b, a, data)

    return y

def plot ():

    input\_volt=int(input\_volt\_amplitude.get())#input from amplitude

    type\_of\_measurement = measurement\_choices.get()#type of measurement

    range1 = choices.get()

    print("Type of measurement : ",type\_of\_measurement)

    print("Range is : ",range1)

    numberlist.append(type\_of\_measurement)

    numberlist.append(str(range1))

    numberlist.append(str(input\_volt))

    if type\_of\_measurement == "Voltage - AC":

        f=int(signal.get())

        lowcut = 40.0

        highcut = 70.0

        o = 3

        if input\_volt<range1:

            print("Signal Frequency is ",f)

            print("Input Voltage(Amplitude) is ",input\_volt)

            print("Sampling Frequency is ",fs)

            amp = 1.414\*input\_volt

            T = 1/f

            Ts = 1/fs

            harmonic\_amp1=0.05\*amp

            harmonic\_amp2=0.05\*amp

            #attenuation\_factor=(amp+harmonic\_amp1+harmonic\_amp2)/2.5

            attenuation\_factor= (range1 + 0.5\*range1) /2.5

            #attenuation\_factor\_value\_label['text'] = attenuation\_factor

            numberlist.append(str(attenuation\_factor))

            attenuation.set(attenuation\_factor)

            continous  = True

            fig = plt.Figure()

            x = np.arange(fs)

            yout= [ ((amp\*np.sin(2\*np.pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(6\*pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(12\*pi\*f \* (i/fs)))) for i in x ]

            yo= [ ((amp\*np.sin(2\*np.pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(6\*pi\*f \* (i/fs)))+(0.05\*amp\*np.sin(12\*pi\*f \* (i/fs))))\*(1/attenuation\_factor) for i in x ]

            adc=[((yo[i]/5)\*65536)+(32768) for i in x]

            for i in x:

                op1 = 32768 + butter\_bandpass\_filter(adc, lowcut, highcut, fs, order=o)

                List\_type = op1.tolist()

            for i in List\_type:

                numberlist.append(convert\_into\_eight\_bits(math.ceil(i)))

            class aScope(object):

                def \_\_init\_\_(self, ax, maxt=2\*T, dt=Ts):

                    self.ax = ax

                    self.dt = dt

                    self.maxt = maxt

                    self.tdata = [0]

                    self.ydata = [0]

                    self.line = Line2D(self.tdata, self.ydata)

                    self.ax.add\_line(self.line)

                    self.ax.set\_ylim(-350,350)

                    self.ax.set\_xlim(0, self.maxt)

                def aupdate(self, y):

                    lastt = self.tdata[-1]

                    if continous :

                        if lastt > self.tdata[0] + self.maxt:

                            self.ax.set\_xlim(lastt-self.maxt, lastt)

                    t = self.tdata[-1] + self.dt

                    self.tdata.append(t)

                    self.ydata.append(y)

                    self.line.set\_data(self.tdata, self.ydata)

                    return self.line,

            def sineEmitter():

                for i in x:

                    yield math.ceil(yout[i])

            fig = plt.Figure(figsize=(6,4))

            canvas = FigureCanvasTkAgg(fig, master=root)

            canvas.get\_tk\_widget().place(x=0, y=190)

            ax1 = fig.add\_subplot(111)

            ax1.set\_xlabel("Time")

            ax1.set\_ylabel("Amplitude")

            #line, = ax1.plot(x, np.sin(x))

            scope = aScope(ax1)

            # pass a generator in "sineEmitter" to produce data for the update func

            plot.ani = animation.FuncAnimation(fig, scope.aupdate, sineEmitter, interval=10,blit=True)

        else:

            print("Signal Frequency is ",f)

            print("Input Voltage(Amplitude) is ",input\_volt)

            print("Sampling Frequency is ",fs)

            #Parameters

            amp = 1.414\*input\_volt       #(Amplitude)

            T = 1/f

            Ts = 1/fs

            harmonic\_amp1=0.05\*amp

            harmonic\_amp2=0.05\*amp

            #attenuation\_factor=(amp+harmonic\_amp1+harmonic\_amp2)/2.5

            attenuation\_factor= (range1 + 0.5\*range1) /2.5

            numberlist.append(str(attenuation\_factor))

            attenuation.set(attenuation\_factor)

            continous  = True

            fig = plt.Figure()

            x = np.arange(fs)

            yout= [ 0 for i in x ]

            yo= [ 0 for i in x ]

            adc=[((yo[i]/5)\*65536)+(32768) for i in x]

            for i in x:

                op1 = 32768 + butter\_bandpass\_filter(adc, lowcut, highcut, fs, order=o)

                List\_type = op1.tolist()

            for i in List\_type:

                numberlist.append(convert\_into\_eight\_bits(math.ceil(i)))

            class aScope(object):

                def \_\_init\_\_(self, ax, maxt=2\*T, dt=Ts):

                    self.ax = ax

                    self.dt = dt

                    self.maxt = maxt

                    self.tdata = [0]

                    self.ydata = [0]

                    self.line = Line2D(self.tdata, self.ydata)

                    self.ax.add\_line(self.line)

                    self.ax.set\_ylim(-350,350)

                    self.ax.set\_xlim(0, self.maxt)

                def aupdate(self, y):

                    lastt = self.tdata[-1]

                    if continous :

                        if lastt > self.tdata[0] + self.maxt:

                            self.ax.set\_xlim(lastt-self.maxt, lastt)

                    t = self.tdata[-1] + self.dt

                    self.tdata.append(t)

                    self.ydata.append(y)

                    self.line.set\_data(self.tdata, self.ydata)

                    return self.line,

            def sineEmitter():

                for i in x:

                    yield math.ceil(yout[i])

            fig = plt.Figure(figsize=(6,4))

            canvas = FigureCanvasTkAgg(fig, master=root)

            canvas.get\_tk\_widget().place(x=0, y=190)

            ax1 = fig.add\_subplot(111)

            ax1.set\_xlabel("Time")

            ax1.set\_ylabel("Amplitude")

            #line, = ax1.plot(x, np.sin(x))

            scope = aScope(ax1)

            plot.ani = animation.FuncAnimation(fig, scope.aupdate, sineEmitter, interval=10,blit=True)

    elif type\_of\_measurement == "Voltage - DC":

        lowcut = 25

        o = 3

        N = 1000

        Ts=1/fs

        if input\_volt<range1:

            print("Input voltage(Amplitude) is ",input\_volt)

            print("Sampling Frequency is ",fs)

            input\_volt=int(input\_volt\_amplitude.get())#----------input from amplitude

            #ip = 100 + ((3\*np.random.randn(N)))

            attenuation\_factor = range1/5

            attenuation.set(attenuation\_factor)

            numberlist.append(attenuation\_factor)

            yout = (input\_volt + ((0.01\*input\_volt\*np.random.randn(N))))

            ip = (input\_volt + ((0.01\*input\_volt\*np.random.randn(N))))\*5/range1

            adc=((ip/5)\*65536)

            x = []

            N2=N/10

            y = []

            it=0

            i=0

            while(it < N):

                op = butter\_lowpass\_filter(adc,lowcut,fs,order = o)

                op1 = gaussian\_filter1d(op, 12)

                it = it + 1

                List\_type = op1.tolist()

            #print(type(List\_type))

            List\_type = op1.tolist()

            for i in List\_type:

                numberlist.append(convert\_into\_eight\_bits(math.ceil(i)))

            fig = plt.Figure(figsize=(6,4))

            canvas = FigureCanvasTkAgg(fig, master=root)

            canvas.get\_tk\_widget().place(x=0, y=190)

            ax1 = fig.add\_subplot(111)

            ax1.set\_xlabel("Time")

            ax1.set\_ylabel("Amplitude")

            ax1.plot(yout)

            plt.show()

            canvas.draw()

        else:

            #error messege

            tk.messagebox.showerror("Error", "Exceeds the range")

            tk.messagebox.showinfo("Hint","The Input Voltage value is always lesser than range value.")

    return numberlist

def reset():

    signal.set(50)

    input\_volt\_amplitude.set(220)

    sam\_fre.set(1000)

    measurement\_choices.set("Voltage - AC")

    choices.set(230)

    #attenuation.set(0.014)

    signal\_freq\_entry['state'] = 'normal'

def on\_option\_change(event):

    selected = measurement\_choices.get()

    if selected == "Voltage - AC":

        #input\_range = {110,230,440,500}

        rangeL = tk.OptionMenu(root,choices, \*ac\_range)

        rangeL.config(bg = "LightYellow2")

        rangeL.configure(width=15)

        rangeL.place(x=250, y=35)

        choices.set(230)

        signal\_freq\_entry['state'] = 'normal'

        signal.set(0)

        input\_volt\_amplitude.set(0)

    elif selected == "Voltage - DC":

        #input\_range = {10,24,48}

        rangeL = tk.OptionMenu(root,choices, \*dc\_range)

        rangeL.config(bg = "LightYellow2")

        rangeL.configure(width=15)

        rangeL.place(x=250, y=35)

        choices.set(10)

        signal\_freq\_entry['state'] = 'disabled'

        signal.set(0)

        input\_volt\_amplitude.set(0)

#Execution starts here

root = tk.Tk()

root.title("Configurator")

title\_bar = Frame(root, bg='brown', relief='raised', bd=2)

title\_bar.pack()

root.geometry("600x700")

root.configure(bg='#FFFFFF')

#Variable data types

measurement\_choices = tk.StringVar()

ac\_choices = tk.IntVar()

dc\_choices = tk.IntVar()

choices=tk.IntVar()

#Assigning values

measurement\_type = {"Voltage - AC", "Voltage - DC"}

measurement\_type=sorted(measurement\_type)

#Label - Type of Measurement

type\_label=tk.Label(root,text="Type of Measurement",bg='White',fg='black',font='Helvetica 8 bold')

type\_label.place(x=50, y=10)

measurement\_choices.set("Voltage - AC")

#Option Menu to select Measurement type

measur\_type = tk.OptionMenu(root,measurement\_choices, \*measurement\_type, command=on\_option\_change)

measur\_type.config(bg = "LightYellow2")

measur\_type.configure(width=15)

measur\_type.place(x=50, y=35)

range\_label=tk.Label(root,text="Range",bg='White',fg='black',font='Helvetica 8 bold')

range\_label.place(x=250, y=10)

ac\_range ={110,230,440,500}

ac\_range=sorted(ac\_range)

dc\_range = {10,24,48}

dc\_range=sorted(dc\_range)

selected = measurement\_choices.get()

if selected ==  "Voltage - AC":

    rangeL = tk.OptionMenu(root,choices, \*ac\_range)

    rangeL.config(bg = "LightYellow2")

    rangeL.configure(width=15)

    rangeL.place(x=250, y=35)

    choices.set(230)

if selected == "Voltage - DC":

    rangeL = tk.OptionMenu(root,choices, \*dc\_range)

    rangeL.config(bg = "LightYellow2")

    rangeL.configure(width=15)

    rangeL.place(x=250, y=35)

    choices.set(230)

#Unit

unit\_range\_label = tk.Label(text="Unit",bg='White',fg='black',font='Helvetica 8 bold')

unit\_range\_label.place(x=465, y=10)

unit\_range\_label1 = tk.Label(text="Volts",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15)

unit\_range\_label1.place(x=430, y=40)

#Signal Frequency

signal\_freq\_label = tk.Label(text="Signal Frequency",bg='White',fg='black',font='Helvetica 8 bold')

signal\_freq\_label.place(x=290, y=100)

signal = tk.DoubleVar()

signal\_freq\_entry = tk.Entry(root, width = 15, textvariable = signal)

signal\_freq\_entry.place(x=290, y=130)

#Amplitude

amplitude\_label = tk.Label(text="Input Voltage",bg='White',fg='black',font='Helvetica 8 bold')

amplitude\_label.place(x=160, y=100)

input\_volt\_amplitude = tk.DoubleVar()

amplitude\_entry = tk.Entry(root, width = 15, textvariable = input\_volt\_amplitude)

amplitude\_entry.place(x=160, y=130)

#Sampling Frequency

sampling\_freq\_label = tk.Label(text="Sampling Frequency",bg='White',fg='black',font='Helvetica 8 bold')

sampling\_freq\_label.place(x=430, y=100)

sam\_fre = tk.DoubleVar()

sampling\_freq\_entry = tk.Label(text="12",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = sam\_fre)

sampling\_freq\_entry.place(x=430, y=130)

sam\_fre.set(1000)

#Attenuation Factor

attenuation\_factor\_label = tk.Label(text="Attenuation Factor",bg='White',fg='black',font='Helvetica 8 bold',width=15)

attenuation\_factor\_label.place(x=30, y=100)

attenuation = tk.DoubleVar()

attenuation\_factor\_value\_label = tk.Label(text="Attenuation Value",bg='DarkSeaGreen3',fg='black',font='Helvetica 8 bold',width=15, textvariable = attenuation)

attenuation\_factor\_value\_label.place(x=30, y=130)

#attenuation.set(0.014)

#Draw button

graph\_button = tk.Button(root, text="Execute",bg='brown',fg='white', command=plot)

graph\_button.place(x=45, y=160)

#Reset button

reset\_button = tk.Button(root,text="Reset", bg='brown',fg='white', command=reset)

reset\_button.place(x=250, y=160)

root.mainloop()

HOST = '127.0.0.1'

PORT = 5032

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

    s.bind((HOST, PORT))

    s.listen()

    conn, addr = s.accept()

    with conn:

        print('Connected by', addr)

        while True:

            for i in numberlist :

                k=k+1

                for j in i :

                    if not i:

                        break

                    data=str(numberlist)

                    data=data.encode()

                    conn.sendall(data)

                    if k == 100:

                        time.sleep(0.1)

                        k=0

conn.close()