GUI motor control

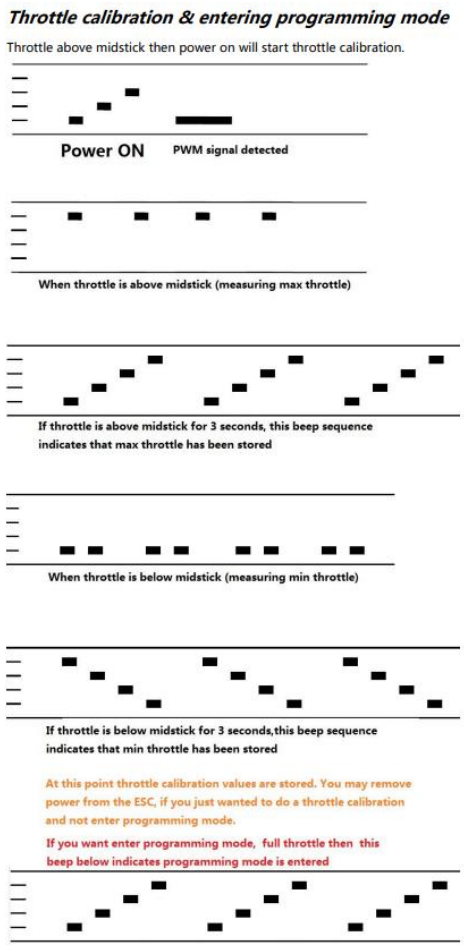
Objective

During my internship, I was tasked with developing a Python GUI to control a DC motor in two modes. In Mode 1, users can define a minimum and maximum velocity percentage

along with a specific step. The motor starts turning at the minimum percentage velocity, gradually increasing until it reaches the maximum percentage. Subsequently, the velocity decreases back to the minimum percentage. This cycle repeats with a user-defined spacing between repetitions. In Mode 2, the motor continuously turns at a constant speed percentage.

Calibration

Electronic speed controllers are responsible for spinning the motors at the speed requested by the user. Most ESCs need to be calibrated so that they know the minimum and maximum pwm values that the microcontroller will send. ESC calibration will vary based on what brand of ESC you are using, so always refer to the documentation for the brand of ESC you are using for specific information (such as tones). During the calibration procedure you are going to hear certain tones so that you know that the values that you have sent were saved successfully. Calibration needs to be done only one time, after that we can just safely send the values that we want to the motor.



As shown in the manual, in order to calibrate the motor, we need to send the maximum value for more than 3 and then we need to send the minimum throttle for more than 3 seconds. At each stage we are going to hear different sounds from the ESC so that we know that the procedure is going well

Arduino Code

```
#include <Servo.h>

const unsigned int MAX_MESSAGE_LENGTH = 12;
Servo ser;
void setup() {
// put your setup code here, to run once:
Serial.begin(9600);
```

```
ser.attach(3); //Attach the ESC to digital port 3 (PWM port required)
delay (15);
ser.write(0);

/* callibration
ser.write(140); delay (5000); ser.write(150); delay(5000);ser.write(10);
delay(5000);ser.write(120); delay (5000);
*/


}

void loop () {
while (Serial.available() > 0)
 {

   //Create a place to hold the incoming message
   static char message [MAX_MESSAGE_LENGTH];
   static unsigned int message_pos = 0;

   //Read the next available byte in the serial receive buffer


  char inByte = Serial.read();

   //Message coming in (check not terminating character) and guard for over
message size
   if (inByte != '\n' && (message_pos < MAX_MESSAGE_LENGTH - 1) && inByte!='%')
   {
     //Add the incoming byte to our message
     message[message_pos] = inByte;
     message_pos++;
   }
   //Full message received...
   else
   {
     //Add null character to string
    message[message_pos] = '\0';
    //Print the message (or do other things)
    //Serial.println(message);
    //Or convert to integer and print
    int number = atoi(message);
    ser.write(number);
    //Reset for the next message
    message_pos = 0;
```

```
   }

 }
```

This Arduino code sets up a Servo motor and listens for commands via the Serial communication. It reads messages from the Serial input, which are expected to be integers representing the desired position for the Servo motor. The messages are terminated by a newline character ('\n'), and the program interprets the received message, converts it to an integer using **`atoi`**, and then sends the integer value to control the Servo motor position using the **`ser.write`** function.

Python Motor Gui

```
class SerialControlGUI(QWidget):
def __init__(self):
        super().__init__()
        self.serial = serial.Serial('COM3', 9600, timeout=10)  #
Adjust the timeout value as needed
        self.initUI()
        self.stop_flag = False  # Flag to control the stop action
        self.sequence_thread = None
        self.current_mode = None  # Variable to store the current mode

    def initUI(self):
        self.layout = QVBoxLayout()

        mode_layout = QHBoxLayout()

        self.mode1_label = QLabel("Mode 1")
        self.mode1_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode1_label.mousePressEvent = self.select_mode1
        mode_layout.addWidget(self.mode1_label)

        self.mode2_label = QLabel("Mode 2")
        self.mode2_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode2_label.mousePressEvent = self.select_mode2
        mode_layout.addWidget(self.mode2_label)
```

```
self.layout.addLayout(mode_layout)

self.percentage_label = QLabel("Percentage:")
self.percentage_input = QLineEdit()
self.layout.addWidget(self.percentage_label)
self.layout.addWidget(self.percentage_input)

self.period_label = QLabel("Period:")
self.period_input = QLineEdit()
self.layout.addWidget(self.period_label)
self.layout.addWidget(self.period_input)

self.space_label = QLabel("Space:")
self.space_input = QLineEdit()
self.layout.addWidget(self.space_label)
self.layout.addWidget(self.space_input)

# Change the name of the Mode 2 input label to "Percentage"
self.percentage_label_mode2 = QLabel("Percentage:")
self.percentage_input_mode2 = QLineEdit()
self.layout.addWidget(self.percentage_label_mode2)
self.layout.addWidget(self.percentage_input_mode2)
self.percentage_label_mode2.hide()
self.percentage_input_mode2.hide()
```

Led Strip Code

After creating the Gui code i was asked to modify the code so that we can add a led strip in the system. The led strip was supposed to change color when the speed of the motor would change. Unfortunately, due to some unexpexted error that i did not

have the time to solve, when the led strip changed color, the motor would stop and start again. So finally, the led strip had two colors: green when the motor was turning and red when the motor stopped. The following scripts are the final scripts that were used

Arduino Code

```
#include <Servo.h>

#include <Adafruit_NeoPixel.h>


int LED_NUM = 320;

static unsigned int message_pos = 0;

const unsigned int MAX_MESSAGE_LENGTH = 12;

static char message [MAX_MESSAGE_LENGTH];

Servo ser;

bool leds_on = false;

Adafruit_NeoPixel pixels = Adafruit_NeoPixel(LED_NUM, 2, NEO_GRB +
NEO_KHZ800);


void setup () {

  Serial.begin(9600);

  ser.attach(3);

  delay (15);

  ser.write(0);

  pixels.begin();

}
```

```
void loop () {
  while (Serial.available() > 0) {

    char inByte = Serial.read();

    if (inByte != '\n' && (message_pos < MAX_MESSAGE_LENGTH - 1) && inByte != '%')
{
      message[message_pos] = inByte;
      message_pos++;
    } else {
      message[message_pos] = '\0';
      int number = atoi(message);

      if(number==-1&&!leds_on){
        for (int i = 0; i < LED_NUM; i++) {
        pixels.setPixelColor(i, pixels.Color(0, 255, 0));
        pixels.show();
        leds_on = true;
        }
      }
      else{
      if (number! =0) {
         ser.write(number);
```

```
    }
    else if(number==0){
      for (int i = 0; i < LED_NUM; i++){
      pixels.setPixelColor(i, pixels.Color(255, 0, 0));
      pixels.show();
      }
      ser.write(number);
      leds_on = false;
     }
    }
    message_pos = 0;
  }
 }
}
```

This Arduino code reads numeric input from the Serial port, controls a servo motor and Neo Pixel LEDs. It turns on all LEDs with a green color when -1 is received, sets the servo position for other numeric inputs, and turns off LEDs when 0 is received. The code continuously listens for input and responds accordingly, allowing dynamic control over the servo and the led strip

Python Code

(PyQt5)

```
import sys
import serial
import time
import threading
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit,
QPushButton, QVBoxLayout, QHBoxLayoutclass SerialControlGUI(QWidget):
```

```
    def __init__(self):
        super().__init__()
        self.serial = serial.Serial('COM3', 9600, timeout=10)  #
Adjust the timeout value as needed
        self.initUI()
        self.stop_flag = False  # Flag to control the stop action
        self.sequence_thread = None
       self.current_mode = None  # Variable to store the current mode
        self.select_mode1(None)


    def initUI(self):

        self.layout = QVBoxLayout()
        mode_layout = QHBoxLayout()
        self.mode1_label = QLabel("Mode 1")
        self.mode1_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode1_label.mousePressEvent = self.select_mode1
        mode_layout.addWidget(self.mode1_label)

        self.mode2_label = QLabel("Mode 2")
        self.mode2_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode2_label.mousePressEvent = self.select_mode2
        mode_layout.addWidget(self.mode2_label)

        self.layout.addLayout(mode_layout)

        self.Min_Power_label = QLabel("Min Power:")
        self.Min_Power_input = QLineEdit()
        self.layout.addWidget(self.Min_Power_label)
        self.layout.addWidget(self.Min_Power_input)

        self.Max_Power_label = QLabel("Max Power:")
        self.Max_Power_input = QLineEdit()
        self.layout.addWidget(self.Max_Power_label)
        self.layout.addWidget(self.Max_Power_input)

        self.Power_Step_label = QLabel("Power Step:")
        self.Power_Step_input = QLineEdit()
        self.layout.addWidget(self.Power_Step_label)
```

```
        self.layout.addWidget(self.Power_Step_input)

        self.Space_Step_label = QLabel("Time space Step:")
        self.Space_Step_input = QLineEdit()
        self.layout.addWidget(self.Space_Step_label)
        self.layout.addWidget(self.Space_Step_input)

        # Change the name of the Mode 2 input label to "Percentage"
        self.Power_label_mode2 = QLabel("Power:")
        self.Power_input_mode2 = QLineEdit()
        self.layout.addWidget(self.Power_label_mode2)
        self.layout.addWidget(self.Power_input_mode2)
        self.Power_label_mode2.hide()
        self.Power_input_mode2.hide()

        self.start_button = QPushButton("Start")
        self.start_button.clicked.connect(self.start_sequence_thread)
        self.layout.addWidget(self.start_button)

        self.stop_button = QPushButton("Stop")
        self.stop_button.clicked.connect(self.stop_sequence_thread)
        self.layout.addWidget(self.stop_button)

        self.setLayout(self.layout)
        self.setWindowTitle('Serial Control GUI')
        self.show()

    def select_mode1(self, event):
        print("Mode 1 selected")
        self.current_mode = 'Mode 1'
        self.show_mode1()

    def select_mode2(self, event):
        print("Mode 2 selected")
        self.current_mode = 'Mode 2'
        self.show_mode2()

    def show_mode1(self):
        self.Min_Power_label.show()
        self.Min_Power_input.show()
        self.Max_Power_label.show()
        self.Max_Power_input.show()
```

```
        self.Power_Step_label.show()
        self.Power_Step_input.show()
        self.Space_Step_label.show()
        self.Space_Step_input.show()
        self.Power_label_mode2.hide()
        self.Power_input_mode2.hide()

    def show_mode2(self):
        self.Min_Power_label.hide()
        self.Min_Power_input.hide()
        self.Max_Power_label.hide()
        self.Max_Power_input.hide()
        self.Power_Step_label.hide()
        self.Power_Step_input.hide()
        self.Space_Step_label.hide()
        self.Space_Step_input.hide()
        self.Power_input_mode2.show()
        self.Power_label_mode2.show()
    def start_sequence_thread(self):
        if self.current_mode == 'Mode 1':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode1)
        elif self.current_mode == 'Mode 2':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode2)

        if self.sequence_thread:
            self.sequence_thread.start()

    def start_sequence_mode1(self):
        try:
            min_power = int(self.Min_Power_input.text())
            max_power = int(self.Max_Power_input.text())
            power_step = int(self.Power_Step_input.text())
            space_step = int(self.Space_Step_input.text())
           # number_of_periods = 5
############################################################   SOS
change number of periods here!!!!!!!!!!!!!!!!!!!
            max_per = (max_power/ 100)
            min_per = (min_power/100)
```

```
            min_val = int(14 + min_per * (60 - 14))
            max_val = int(14 + max_per * (60 - 14))
            val = min_val
            percentage = min_power
            iterations = int((max_power-min_power)/power_step)
            #self.command("-1")
            time.sleep(3)
            while (True):
                for i in range(iterations):
                    if self.stop_flag:
                        self.command("0")  # Send '0' to ensure a
graceful stop
                        return
                    print(str(percentage)+"%")
                    val = int(14 + (percentage/100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    val += power_step
                    percentage += power_step
                    time.sleep(space_step)

                for j in range(iterations):
                    if self.stop_flag:
                        self.command("0")  # Send '0' to ensure a
graceful stop
                        return
                    print(str(percentage) + "%")
                    val = int(14 + (percentage / 100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    percentage -= power_step
                    time.sleep(space_step)
                time.sleep(3)

        except ValueError:
            print("Please enter valid integer values for Percentage,
Period, and Space")

    def start_sequence_mode2(self):
        try:
            #self.command("-1")
            percentage = int(self.Power_input_mode2.text())  # Using
```

```
the correct input field for Mode 2
            per = (percentage / 100)
            val = int(14 + per * (60 - 14))
            self.command(str(val))
            # Implement Mode 2 sequence using mode2_specific_value
            # Perform tasks related to Mode 2 using the
'mode2_specific_value'
            pass
        except ValueError:
            print("Please enter valid values for Mode 2")

    def stop_sequence_thread(self):
        self.stop_flag = True
        self.command("0")
        time.sleep(0.1)  # Add a short delay to allow the sequence
thread to process the stop request
        if self.sequence_thread:
            self.sequence_thread.join()  # Wait for the sequence
thread to complete

    def command(self, value):
        self.serial.write(str.encode(value + "%"))
        time.sleep(0.1)

def main():
    app = QApplication(sys.argv)
    ex = SerialControlGUI()
    sys.exit(app.exec_())

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

In this code the user gives the minimum speed percentage(min_power), the maximum percentage(max_power), the step(power_step) and the space between the periods(space_val). With these values we can calculate the exact number of iterations that we want.

The minimum value that we can send to the ESC is 14 and after 16 the speed of the motor remains almost the same.

Python Code

(Pyside6)

```
import sys
import serial
import time
import threading
from PySide6.QtWidgets import QApplication, QWidget, QLabel,
QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout


class SerialControlGUI(QWidget):
def __init__(self):
        super().__init__()
        self.serial = serial.Serial('COM3', 9600, timeout=10)  #
Adjust the timeout value as needed
        self.initUI()
        self.stop_flag = False  # Flag to control the stop action
        self.sequence_thread = None
        self.current_mode = None  # Variable to store the current mode
        self.select_mode1(None)  # Call select_mode1 to set Mode 1 as
the default mode

    def initUI(self):
        self.layout = QVBoxLayout()

        mode_layout = QHBoxLayout()

        self.mode1_label = QLabel("Mode 1")
        self.mode1_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode1_label.mousePressEvent = self.select_mode1
        mode_layout.addWidget(self.mode1_label)

        self.mode2_label = QLabel("Mode 2")
        self.mode2_label.setStyleSheet("border: 1px solid black;
padding: 10px;")
        self.mode2_label.mousePressEvent = self.select_mode2
        mode_layout.addWidget(self.mode2_label)

        self.layout.addLayout(mode_layout)

        self.Min_Power_label = QLabel("Min Power:")
```

```
        self.Min_Power_input = QLineEdit()
        self.layout.addWidget(self.Min_Power_label)
        self.layout.addWidget(self.Min_Power_input)

        self.Max_Power_label = QLabel("Max Power:")
        self.Max_Power_input = QLineEdit()
        self.layout.addWidget(self.Max_Power_label)
        self.layout.addWidget(self.Max_Power_input)

        self.Power_Step_label = QLabel("Power Step:")
        self.Power_Step_input = QLineEdit()
        self.layout.addWidget(self.Power_Step_label)
        self.layout.addWidget(self.Power_Step_input)

        self.Space_Step_label = QLabel("Time space Step:")
        self.Space_Step_input = QLineEdit()
        self.layout.addWidget(self.Space_Step_label)
        self.layout.addWidget(self.Space_Step_input)

        # Change the name of the Mode 2 input label to "Percentage"
        self.Power_label_mode2 = QLabel("Power:")
        self.Power_input_mode2 = QLineEdit()
        self.layout.addWidget(self.Power_label_mode2)
        self.layout.addWidget(self.Power_input_mode2)
        self.Power_label_mode2.hide()
        self.Power_input_mode2.hide()

        self.start_button = QPushButton("Start")
        self.start_button.clicked.connect(self.start_sequence_thread)
        self.layout.addWidget(self.start_button)

        self.stop_button = QPushButton("Stop")
        self.stop_button.clicked.connect(self.stop_sequence_thread)
        self.layout.addWidget(self.stop_button)

        self.setLayout(self.layout)
        self.setWindowTitle('Serial Control GUI')
        self.show()

    def select_mode1(self, event):
        print("Mode 1 selected")
        self.current_mode = 'Mode 1'
```

```
        self.show_mode1()

    def select_mode2(self, event):
        print("Mode 2 selected")
        self.current_mode = 'Mode 2'
        self.show_mode2()

    def show_mode1(self):
        self.Min_Power_label.show()
        self.Min_Power_input.show()
        self.Max_Power_label.show()
        self.Max_Power_input.show()
        self.Power_Step_label.show()
        self.Power_Step_input.show()
        self.Space_Step_label.show()
        self.Space_Step_label.show()
        self.Power_label_mode2.hide()
        self.Power_input_mode2.hide()

    def show_mode2(self):
        self.Min_Power_label.hide()
        self.Min_Power_input.hide()
        self.Max_Power_label.hide()
        self.Max_Power_input.hide()
        self.Power_Step_label.hide()
        self.Power_Step_input.hide()
        self.Space_Step_label.hide()
        self.Space_Step_input.hide()
        self.Power_input_mode2.show()
        self.Power_label_mode2.show()

    def start_sequence_thread(self):
        if self.current_mode == 'Mode 1':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode1)
        elif self.current_mode == 'Mode 2':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode2)

        if self.sequence_thread:
```

```
            self.sequence_thread.start()

    def start_sequence_mode1(self):
        try:
            min_power = int(self.Min_Power_input.text())
            max_power = int(self.Max_Power_input.text())
            power_step = int(self.Power_Step_input.text())
            space_step = int(self.Space_Step_input.text())

            # number_of_periods = 5
############################################################   SOS
change number of periods here!!!!!!!!!!!!!!!!!!!!

            max_per = (max_power / 100)
            min_per = (min_power / 100)
            min_val = int(14 + min_per * (60 - 14))
            max_val = int(14 + max_per * (60 - 14))
            val = min_val
            percentage = min_power
            iterations = int((max_power - min_power) / power_step)
            self.command("-1")
            time.sleep(3)
            while (True):
                for i in range(iterations):
                    if self.stop_flag:
                        self.command("0")  # Send '0' to ensure a
graceful stop
                        return
                    print(str(percentage) + "%")
                    val = int(14 + (percentage / 100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    val += power_step
                    percentage += power_step
                    time.sleep(space_step)

                for j in range(iterations):
                    if self.stop_flag:
                        self.command("0")  # Send '0' to ensure a
graceful stop
                        return
                    print(str(percentage) + "%")
```

```
                    val = int(14 + (percentage / 100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    percentage -= power_step
                    time.sleep(space_step)
                time.sleep(3)

        except ValueError:
            print("Please enter valid integer values for Percentage,
Period, and Space")

    def start_sequence_mode2(self):

        try:
            self.command("-1")
            percentage = int(self.Power_input_mode2.text())  # Using
the correct input field for Mode 2
            per = (percentage / 100)
            val = int(14 + per * (60 - 14))
            self.command(str(val))
            # Implement Mode 2 sequence using mode2_specific_value
            # Perform tasks related to Mode 2 using the
'mode2_specific_value'
            pass
        except ValueError:
            print("Please enter valid values for Mode 2")

    def stop_sequence_thread(self):
        self.stop_flag = True
        self.command("0")
        if self.sequence_thread:
            self.sequence_thread.join()  # Wait for the sequence
thread to complete

    def command(self, value):
        self.serial.write(str.encode(value + "%"))
        time.sleep(0.1)


def main():
    app = QApplication(sys.argv)
    ex = SerialControlGUI()
```

```
    sys.exit(app.exec())


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

Python Code

(tkinter)

```
import sys
import serial
import time
import threading
import tkinter as tk
from tkinter import ttk

class SerialControlGUI(tk.Tk):
def __init__(self):
 super().__init__()
        self.serial = None
        self.stop_flag = False
        self.sequence_thread = None
        self.current_mode = None
        self.initUI()
        self.select_mode1(None)

    def initUI(self):
        self.title('Serial Control GUI')
        self.geometry('400x300')  # Fixed window size

        # Connection tab
        connection_tab = ttk.LabelFrame(self, text="Connection")
        connection_tab.pack(fill='both', expand=True)

        self.com_port_label = ttk.Label(connection_tab, text="COM
Port:")
        self.com_port_entry = ttk.Entry(connection_tab)
        self.com_port_label.grid(row=0, column=0, sticky="e")
        self.com_port_entry.grid(row=0, column=1)

        self.connect_button = ttk.Button(connection_tab,
```

```
text="Connect", command=self.connect_serial)
        self.connect_button.grid(row=1, column=0, columnspan=2)

        # Main tab
        main_tab = ttk.LabelFrame(self, text="Main")
        main_tab.pack(fill='both', expand=True)

        mode_frame = ttk.Frame(main_tab)
        mode_frame.grid(row=0, column=0)

        self.mode1_label = ttk.Label(mode_frame, text="Mode 1")
        self.mode1_label.configure(style="Mode.TLabel")
        self.mode1_label.bind('<Button-1>', self.select_mode1)
        self.mode1_label.grid(row=0, column=0)

        self.mode2_label = ttk.Label(mode_frame, text="Mode 2")
        self.mode2_label.configure(style="Mode.TLabel")
        self.mode2_label.bind('<Button-1>', self.select_mode2)
        self.mode2_label.grid(row=0, column=1)

        self.Min_Power_label = ttk.Label(main_tab, text="Min Power:")
        self.Min_Power_input = ttk.Entry(main_tab)
        self.Min_Power_label.grid(row=1, column=0, sticky="e")
        self.Min_Power_input.grid(row=1, column=1)

        self.Max_Power_label = ttk.Label(main_tab, text="Max Power:")
        self.Max_Power_input = ttk.Entry(main_tab)
        self.Max_Power_label.grid(row=2, column=0, sticky="e")
        self.Max_Power_input.grid(row=2, column=1)

        self.Power_Step_label = ttk.Label(main_tab, text="Power
Step:")
        self.Power_Step_input = ttk.Entry(main_tab)
        self.Power_Step_label.grid(row=3, column=0, sticky="e")
        self.Power_Step_input.grid(row=3, column=1)

        self.Space_Step_label = ttk.Label(main_tab, text="Time space
Step:")
        self.Space_Step_input = ttk.Entry(main_tab)
        self.Space_Step_label.grid(row=4, column=0, sticky="e")
        self.Space_Step_input.grid(row=4, column=1)
```

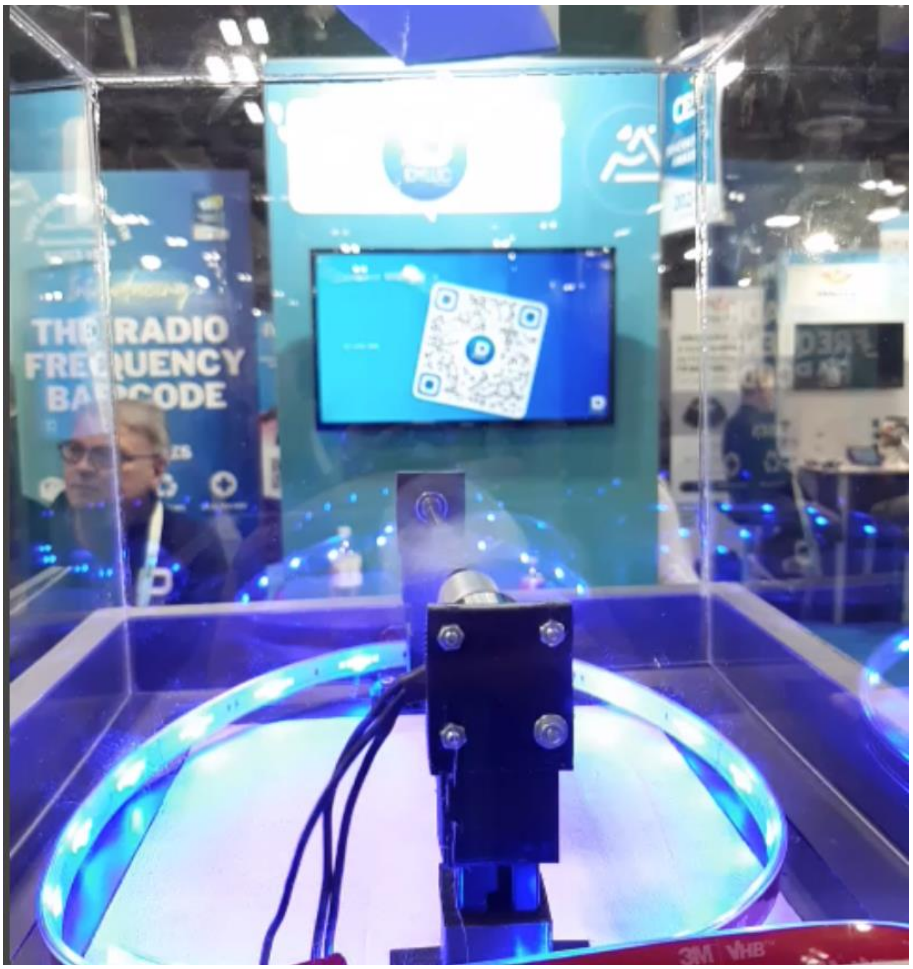
```
        # Change the name of the Mode 2 input label to "Percentage"
        self.Power_label_mode2 = ttk.Label(main_tab, text="Power:")
        self.Power_input_mode2 = ttk.Entry(main_tab)
        self.Power_label_mode2.grid(row=5, column=0, sticky="e")
        self.Power_input_mode2.grid(row=5, column=1)
        self.Power_label_mode2.grid_remove()
        self.Power_input_mode2.grid_remove()

        self.start_button = ttk.Button(main_tab, text="Start",
command=self.start_sequence_thread)
        self.start_button.grid(row=6, column=0, columnspan=2)

        self.stop_button = ttk.Button(main_tab, text="Stop",
command=self.stop_sequence_thread)
        self.stop_button.grid(row=7, column=0, columnspan=2)

        self.style = ttk.Style()
        self.style.configure("Mode.TLabel", borderwidth=1, padding=10)

    def connect_serial(self):
        com_port = self.com_port_entry.get()
        try:
            self.serial = serial.Serial(com_port, 9600, timeout=10)
            print(f"Connected to {com_port}")
        except Exception as e:
            print(f"Error connecting to {com_port}: {e}")

    def select_mode1(self, event):
        print("Mode 1 selected")
        self.current_mode = 'Mode 1'
        self.show_mode1()

    def select_mode2(self, event):
        print("Mode 2 selected")
        self.current_mode = 'Mode 2'
        self.show_mode2()

    def show_mode1(self):
        self.Power_label_mode2.grid_remove()
        self.Power_input_mode2.grid_remove()
        self.Min_Power_label.grid()
        self.Min_Power_input.grid()
```

```
        self.Max_Power_label.grid()
        self.Max_Power_input.grid()
        self.Power_Step_label.grid()
        self.Power_Step_input.grid()
        self.Space_Step_label.grid()
        self.Space_Step_input.grid()

    def show_mode2(self):
        self.Min_Power_label.grid_remove()
        self.Min_Power_input.grid_remove()
        self.Max_Power_label.grid_remove()
        self.Max_Power_input.grid_remove()
        self.Power_Step_label.grid_remove()
        self.Power_Step_input.grid_remove()
        self.Space_Step_label.grid_remove()
        self.Space_Step_input.grid_remove()
        self.Power_input_mode2.grid()
        self.Power_label_mode2.grid()

    def start_sequence_thread(self):
        if not self.serial:
            print("Please connect to a COM port first.")
            return

        if self.current_mode == 'Mode 1':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode1)
        elif self.current_mode == 'Mode 2':
            self.stop_flag = False
            self.sequence_thread =
threading.Thread(target=self.start_sequence_mode2)

        if self.sequence_thread:
            self.sequence_thread.start()

    def start_sequence_mode1(self):
        try:
           min_power = int(self.Min_Power_input.get())
            max_power = int(self.Max_Power_input.get())
            power_step = int(self.Power_Step_input.get())
            space_step = int(self.Space_Step_input.get())
```

```
            max_per = (max_power/ 100)
            min_per = (min_power/100)
            min_val = int(14 + min_per * (60 - 14))
            max_val = int(14 + max_per * (60 - 14))
            val = min_val
            percentage = min_power
            iterations = int((max_power-min_power)/power_step)
            self.command("-1")
            time.sleep(3)
            while True:
                for _ in range(iterations):
                    if self.stop_flag:
                        self.command("0")
                        return
                    print(str(percentage)+"%")
                    val = int(14 + (percentage/100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    val += power_step
                    percentage += power_step
                    time.sleep(space_step)

                for _ in range(iterations):
                    if self.stop_flag:
                        self.command("0")
                        return
                    print(str(percentage) + "%")
                    val = int(14 + (percentage / 100) * (60 - 14))
                    print(val)
                    self.command(str(val))
                    percentage -= power_step
                    time.sleep(space_step)
                time.sleep(3)

        except ValueError:
            print("Please enter valid integer values for Percentage,
Period, and Space")

    def start_sequence_mode2(self):
        try:
            self.command("-1")
```

```
            percentage = int(self.Power_input_mode2.get())  # Using
the correct input field for Mode 2
            per = (percentage / 100)
            val = int(14 + per * (60 - 14))
            self.command(str(val))
            # Implement Mode 2 sequence using mode2_specific_value
            # Perform tasks related to Mode 2 using the
'mode2_specific_value'
            pass
        except ValueError:
            print("Please enter valid values for Mode 2")

    def stop_sequence_thread(self):
        self.stop_flag = True
        self.command("0")
        if self.sequence_thread:
            self.sequence_thread.join()

    def command(self, value):
        if self.serial:
            self.serial.write(str.encode(value + "%"))
            time.sleep(0.1)

    def on_close(self):
        self.stop_sequence_thread()
        if self.serial:
            self.serial.close()
        self.destroy()

def main():
    app = SerialControlGUI()
    app.mainloop()


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

I was asked to write the code this time in PySide6 and tkinter. The only difference this time is that i am sending “–1” before i send any value to the motor in Mode 1 and in Mode 2 which is the code for the Arduino to turn on the led strip with the green color.


RADIO
FREQUENCY
3M VHB