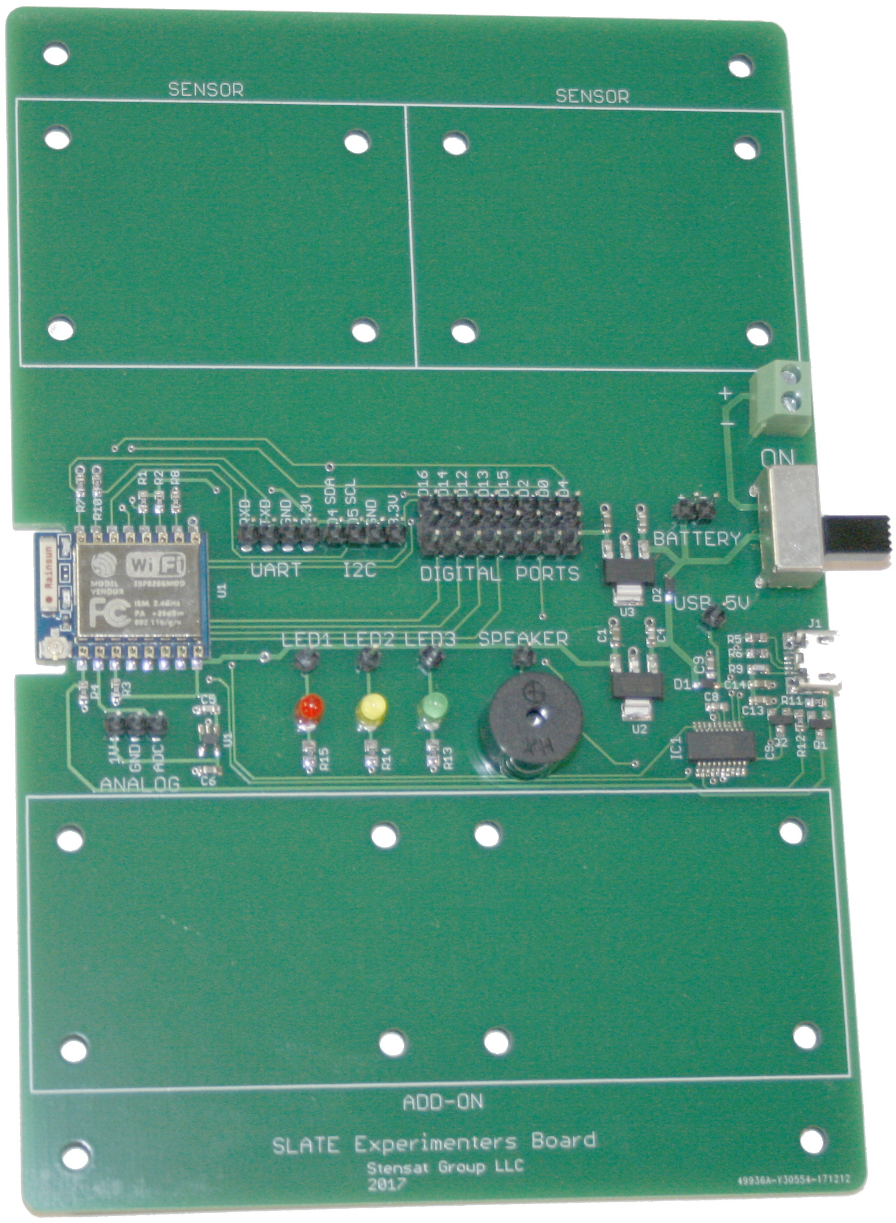
Sten-SLATE ESP Kit

Sending Serial Data



Stensat Group LLC,

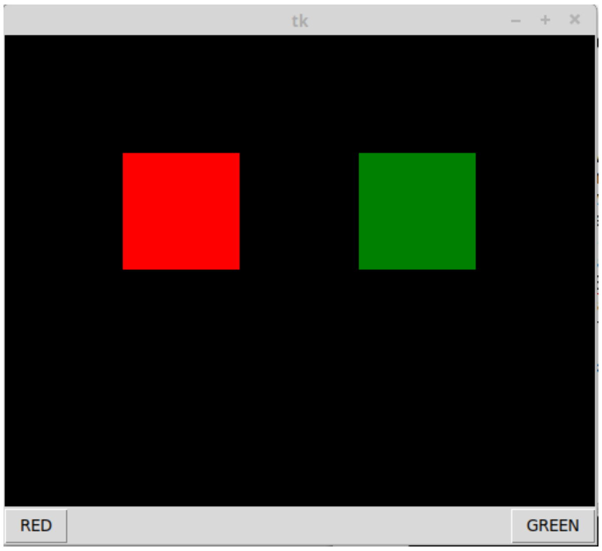
Sending Data

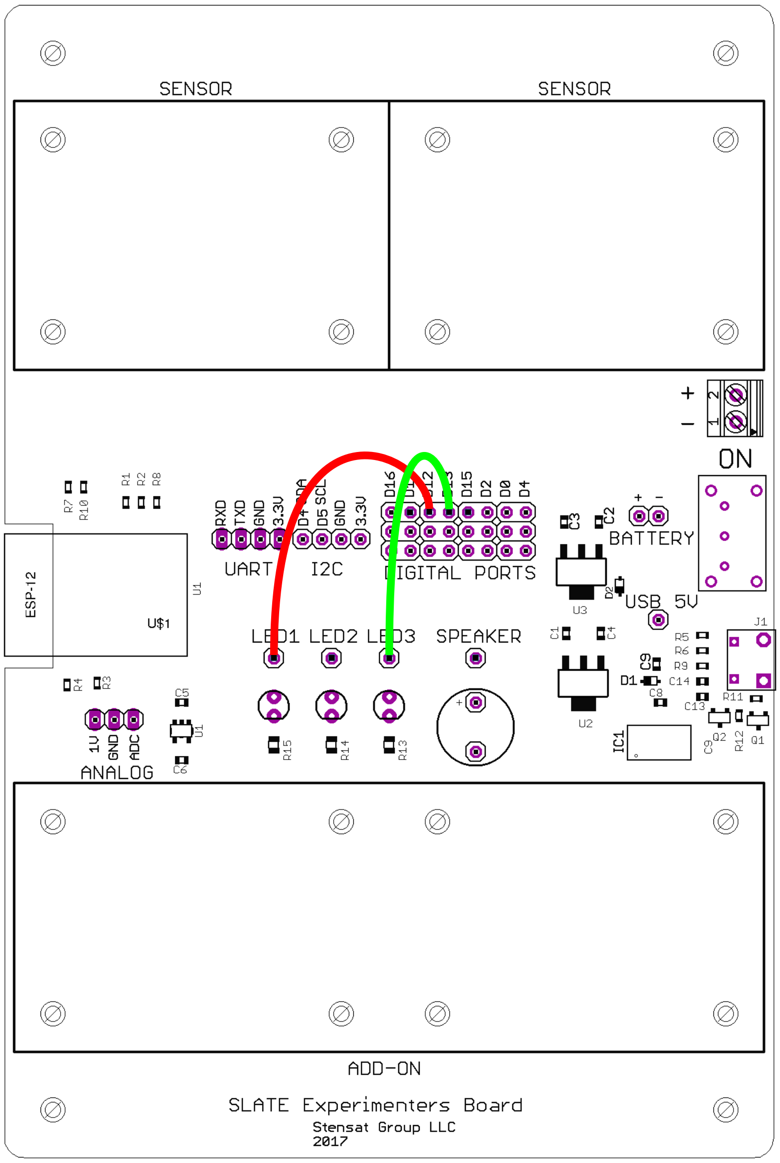
In this section, you will be shown how to send data to and from the SLATE to the a python program. The python program will receive the data and plot it in real time. To plot data, the matplotlib library will be used. This library work similar to Matlab plotting functions.

The first version will show how to collect data over the serial interface and plot it. The second version will show how to collect data over WiFi and plot it.

Graphical Interface

In this section, you will learn how to build a simple graphical user interface to control two LEDs through the USB port. Reuse the two LED circuit from the BASIC Circuits lesson. Two buttons will be created to control the two LEDs. A new prorgram will be created in Processing to command the SLATE and a new program for the SLATE will be created to accept and process the commands.





Graphical Interface

The Arduino program will look for a single character command from the serial interface over the USB port and process it. It first checks if a command character has been received. Once received, it will read the character and then use the **switch()** function to determine which command was received and turn the appropriate LED on or off.

Upload this to the Experimenters board. Save the program with the file name **RedGreenSerial**.

Open the serial monitor. Type each letter in the serial monitor and press the **Enter** key. The program should respond to the letter selected. Make sure to use caps.

Arduino Program

```
void setup() {
  Serial.begin(115200);
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
}

void loop() {
  if(Serial.available() > 0) {
    int a = Serial.read();
    switch(a) {
      case 'F' : digitalWrite(12,HIGH);
                 break;
      case 'B' : digitalWrite(12,LOW);
                 break;
      case 'L' : digitalWrite(13,HIGH);
                 break;
      case 'R' : digitalWrite(13,LOW);
    }
  }
}
```

Graphical Interface

We will reuse the program buttons.py.

In order for the python program to talk to the SLATE board, a new library needs to be added.

Open a terminal or powershell. Enter the command:

pip3 install pyserial

You do need to be connected to the internet for the library to be installed. Once completed, you can start writing the code.

The highlighted lines are where the new serial code will be added.

The serial interface is opened and configured to operate at 115200 baud. Specify the COM port used by the Arduino software.

Python Program

```
from tkinter import *
import serial

top = Tk()

toggle_red = 0
toggle_green = 0

s = serial.Serial("COM4",115200)

c = Canvas(top,bg='black',height=400,width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.write(b'F')
    else:
        color = 'gray'
        toggle_red = 0
        s.write(b'B')
    c.itemconfig(rrect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.write(b'L')
    else:
        color = 'gray'
        toggle_green = 0
        s.write(B'R')
    c.itemconfig(grect,fill=color)

rrect = c.create_rectangle(100,100,200,200,fill ='gray')
grect = c.create_rectangle(300,100,400,200,fill = 'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```

Graphical Interface

s.write() is the function that writes the command byte to the SLATE. Notice the character being sent is preceded with the letter b. This converts the character which is automatically in unicode to a byte character.

Python 3 handles all strings as unicode which is 16-bits long. The serial interface cannot support that so the string needs to be converted to an 8-bit character.

Notice the last two **pack()** functions. The argument added will make the two buttons be positioned at the same level with the RED button the left and the GREEN button on the right.

Save the program as **button_serial.py**.

Load and run the SLATE program. Then start the python program. Click on each button multiple times to see SLATE react.

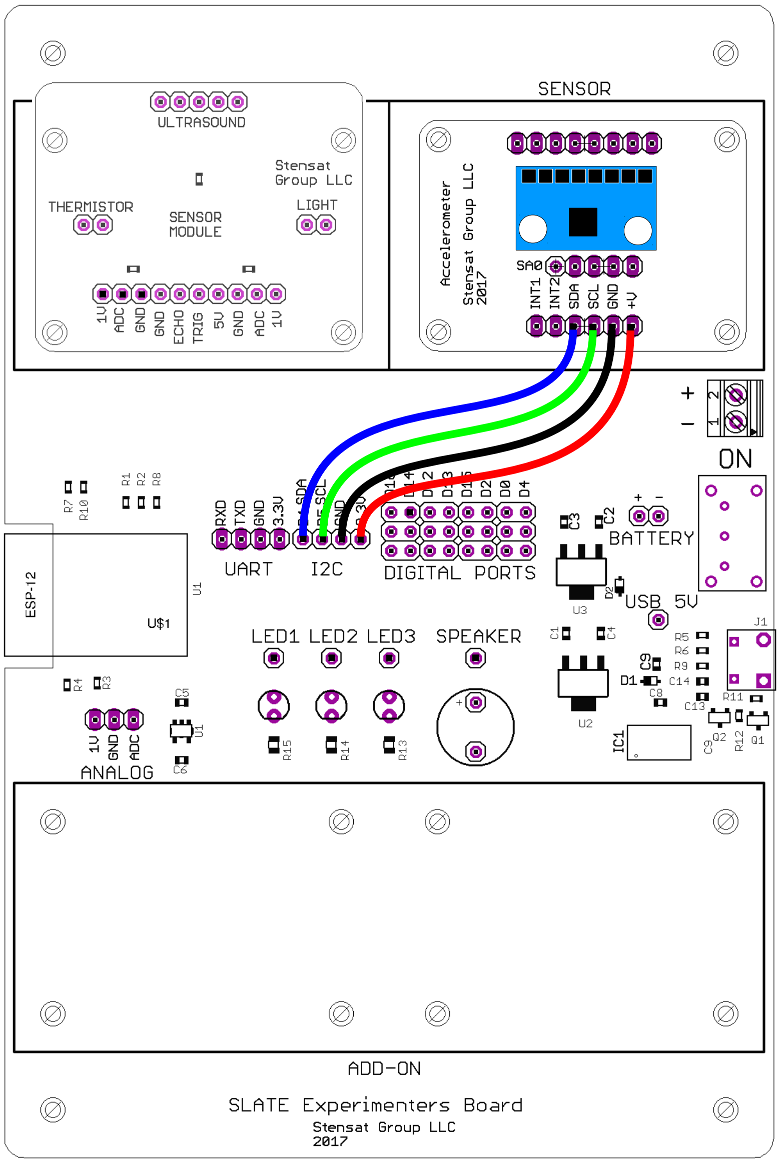
Python Program

```
from tkinter import *
import serial

top = Tk()

toggle_red = 0
toggle_green = 0

s = serial.Serial("COM4",115200)

c = Canvas(top,bg='black',height=400,width=500)

def toggle_red_rect():
    global toggle_red
    if toggle_red == 0:
        color = 'red'
        toggle_red = 1
        s.write(b'F')
    else:
        color = 'gray'
        toggle_red = 0
        s.write(b'B')
    c.itemconfig(red_rect,fill = color)

def toggle_green_rect():
    global toggle_green
    if toggle_green == 0:
        color = 'green'
        toggle_green = 1
        s.write(b'L')
    else:
        color = 'gray'
        toggle_green = 0
        s.write(b'R')
    c.itemconfig(green_rect,fill=color)

red_rect = c.create_rectangle(100,100,200,200,fill ='gray')
green_rect = c.create_rectangle(300,100,400,200,fill =
'gray')
rb = Button(top,text='RED',command=toggle_red_rect)
gb = Button(top,text='GREEN',command=toggle_green_rect)

c.pack()
rb.pack(side=LEFT)
gb.pack(side=RIGHT)
top.mainloop()
```

Sending Data

The IMU program from earlier will be used for this section. The IMU program generated orientation in degrees for the three axis.

Reconnect the IMU as shown.



Sending Data

The program from the IMU section will be reused as is. The code is shown to the right. The only change in the code is changing the delay to 10 milliseconds instead of 100 ms at the end of the program. The python program can handle higher data rates.

For python, a new library needs to be added. Open a terminal or powershell and execute the programs below.

pip3 install matplotlib

For more information on matplotlib, go to https://matplotlib.org/

Arduino Program

```
#include <MPU6050_tockn.h>
#include <Wire.h>

MPU6050 mpu6050(Wire);
long timer = 0;
char buf[64];

void setup() {
  Serial.begin(115200);
  Wire.begin(4,5);
  mpu6050.begin(ACCEL_2G,GYRO_500);
  mpu6050.calcGyroOffsets(true);
}

void loop() {
  mpu6050.update();
  Serial.print(mpu6050.getAngleX());
  Serial.print(“,“);
  Serial.print(mpu6050.getAngleY());
  Serial.print(“,“);
  Serial.println(mpu6050.getAngleZ());
  delay(10);
}
```

Sending Data

Python Program

This python program will receive data from the serial interface and plot the X,Y,Z data from the IMU as the data becomes available.

Three libraries are imported. First is the matplotlib library. Notice the **as plt**. This renames the library to **plt** so there is less typing. The second matplotlib library is to support real time updates of the plots.

Line 5 creates a figure. This will be window that contains the plot. Line 6 adds a plot to the figure. There is only one plot.

Line 8 sets the number of data points to be plotted. Line 9 declares a variable to hold the Y scale of the plot. The IMU is operating at 2 G range.

Line 11 fills in the x-axis values from 0 to 199. Line 12, 13, 14 fill arrays with zero of length 200. Each of the arrays will hold the IMU X, Y, Z values.

Line 15 sets the Y scale for the plot using the variable value.

Line 16 opens the serial port.

Line 17 declares a line plot for the X value. Line 18 does the same for the Y value and line 19 does the same for the X value. Each data plot will be plotted in the one plot in the figure. The name of the plot is **ax**.

Lines 20 to 22 set up the plot labels. Line 23 makes the data legend visible.

```
1  import matplotlib.pyplot as plt
2  import matplotlib.animation as animation
3  import serial
4
5  fig = plt.figure()
6  ax = fig.add_subplot(1, 1, 1)
7
8  x_len = 200
9  y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5',115200)
17 line, = ax.plot(xs, xa,label='X')
18 line2, = ax.plot(xs,ya,label='Y')
19 line3, = ax.plot(xs,za,label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa,ya,za):
26     a = s.readline()
27     b = a.decode('utf-8',’ingore’)
28     c = b.split(',')
29     if len(c) == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line,line2,line3,
40     else:
41         return line,line2,line3,
42 ani = animation.FuncAnimation(fig,animate,
43     fargs=(xa,ya,za,),
44     interval=1,
45     blit=True)
46 plt.show()  # show the figure
```

Sending Data

Lines 25 through 41 are for the function **animate**. This function is called repeatedly to update the plot. Variables i**,xa,ya,za** are passed to the function. variable **i** is automatically passed and provides a count update. It is not used.

Line 26 reads data from the serial interface. If data is not available, the program halts until it becomes available. The received data is a byte array stored in variable a.

Line 27 converts the byte array into a string which uses unicode. Python 3 works with strings so the byte array needs to be converted. The ‘ignore’ argument tells the function to not throw an error and stop the program if it cannot properly convert the byte array. This can happen when the program starts while the SLATE is sending data and the byte array is not received properly the first time. The issue only happens at start up.

https://en.wikipedia.org/wiki/UTF-8

Line 28 splits the received data into individual X, Y, Z values. The data is separated by a space. the split() function argument is the character that is used to separate the values.

Line 29 verifies there are three values. Some times the program will start reading the serial interface in the middle of data being sent and not all the data is received. This makes sure all three values have been received so the program will not crash.

Lines 30 to 32 add the data to the arrays. Since the split values are still strings, the strings need to be converted to floating point values.

Lines 33 to 35, trim the arrays back to 200 values removing the oldest value.

```
1   import matplotlib.pyplot as plt
2   import matplotlib.animation as animation
3   import serial
4
5   fig = plt.figure()
6   ax = fig.add_subplot(1, 1, 1)
7
8   x_len = 200
9   y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5',115200)
17 line, = ax.plot(xs, xa,label='X')
18 line2, = ax.plot(xs,ya,label='Y')
19 line3, = ax.plot(xs,za,label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa,ya,za):
26     a = s.readline()
27     b = a.decode('utf-8',’ignore’)
28     c = b.split(',')
29     if len(c)  == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line,line2,line3,
40     else:
41         return line,line2,line3,
42 ani = animation.FuncAnimation(fig,animate,
43     fargs=(xa,ya,za,),
44     interval=1,
45     blit=True)
46 plt.show()  # show the figure
```

Sending Data

Lines 36 to 38 update the plot data by reloading the **linex,liney,linez** with the updated arrays.

The updated arrays are returned in line 39.

Line 40 and 41 handle the situation where data was not in the proper format. The non updated **linex,liney, linez** are returned.

Line 42 sets up the real time plotting. The first argument identifies the figure to be updated. The second argument is the function that does update the data. The third argument specifies the data arrays the animate function will use. The **interval** argument specifies how fast to animate the plot. The **blit=True** accelerates the plotting so it can keep up with the data.

interval is set to 1 millisecond. This is faster than the data being generated. This is done so that the program does not lag behind the data. The **readline()** function will control the speed of the plotting based on the rate the data is received.

Line 46 makes the figure with the plot visible.

Load the SLATE with the IMU program and let it start. Start the python program. It may take several seconds to start. This is based on how fast the laptop is and how long it takes to load the modules and start executing.

```
1    import matplotlib.pyplot as plt
2    import matplotlib.animation as animation
3    import serial
4
5    fig = plt.figure()
6    ax = fig.add_subplot(1, 1, 1)
7
8    x_len = 200
9    y_range = [-200, 200]
10
11 xs = list(range(0, 200))
12 xa = [0] * x_len
13 ya = [0] * x_len
14 za = [0] * x_len
15 ax.set_ylim(y_range)
16 s = serial.Serial('COM5',115200)
17 line, = ax.plot(xs, xa,label='X')
18 line2, = ax.plot(xs,ya,label='Y')
19 line3, = ax.plot(xs,za,label='Z')
20 plt.title('IMU')
21 plt.xlabel('Samples')
22 plt.ylabel('Degrees')
23 ax.legend()
24
25 def animate(i, xa,ya,za):
26     a = s.readline()
27     b = a.decode('utf-8',’ignore’)
28     c = b.split(',')
29     if len(c)  == 3:
30         xa.append(float(c[0]))
31         ya.append(float(c[1]))
32         za.append(float(c[2]))
33         xa = xa[-x_len:]
34         ya = ya[-x_len:]
35         za = za[-x_len:]
36         line.set_ydata(xa)
37         line2.set_ydata(ya)
38         line3.set_ydata(za)
39         return line,line2,line3,
40     else:
41         return line,line2,line3,
42 ani = animation.FuncAnimation(fig,animate,
43     fargs=(xa,ya,za,),
44     interval=1,
45     blit=True)
46 plt.show()  # show the figure
```