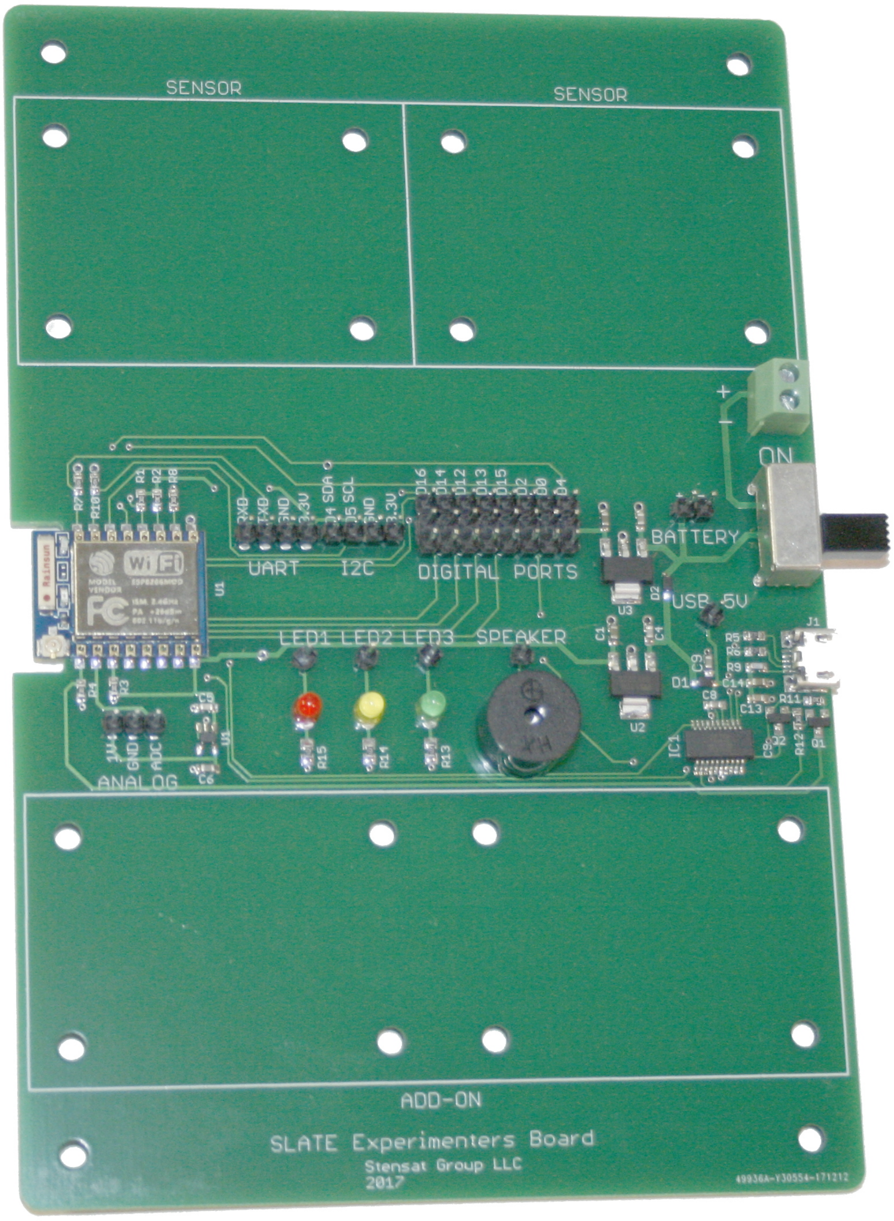
Sten-SLATE ESP Kit

Sending Data Over WiFi



Sending Data over WiFi

In this section, the IMU data will be sent to the python program over WiFi using a UDP network connection. Remember a UDP connection does not require the SLATE to make a connection to a computer. In this example, the python program will operate as a server and listen for UDP packets.

Arduino Code

The setup code will be similar to the original accelerometer code except with the addition of configuring the WiFi.

The **ESP8266WiFi.h** include file is added to the top of the code. The **WiFiUdp.h** include file is also added.

The **udp** object is declared to send and receive UDP packets.

The variable **send_data** is declared and set to zero. This will be used to indicate when a packet is received from the python program.

The **remote** variable is declared as an IPAddress type variable. This will be used to store the IP address of the computer running the python program.

The variable **buf** will hold the data to be sent to the python program as a character array.

In the setup() function, the I2C interface is configured as before and the WiFi if configured. Last, the accelerometer is configured as before.

A UDP port is declared so the SLATE can get a packet from the python program and acquire the IP address to send data.

Arduino Program

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <MPU6050_tockn.h>
#include <Wire.h>

WiFiUDP udp;

int send_data = 0;
IPAddress remote;
char buf[32];

MPU6050 mpu6050(Wire);

void setup() {
  Wire.begin(4,5);
  Serial.begin(115200);
  WiFi.mode(WIFI_AP);
  WiFi.softAP(“SSID name”);
  udp.begin(10000);
  mpu6050.begin(ACCEL_2G,GYRO_500);
  mpu6050.calcGyroOffsets(true);
}
```

Arduino Code

In the loop() function, the udp port is checked for any UDP packet from the python program if a packet has not been received before. The contents do not matter for this program. The purpose is to acquire the IP address of the computer the python program is running on. When a packet is received, the **send_data** variable is set to 1 and the IP address is captured.

When the program has captured the IP address and send_data is set to 1, the program will then get the accelerometer data and send it to the python program.

The **snprintf()** function format the data into a character array that can be read by a person. The values are converted into an ASCII string. The first parameter is the character array to store the string. The second parameter is the size of the character array. The third parameter specifies the formatting of the string. After that, the rest of the parameters are the variables that are used to put values in the string.

To send a UDP packet, **udp.beginPacket()** is required to specify the IP address and network port. **udp.print()** fills the packet with the contents. Multiple **udp.print()** statements are allowed. The packet is sent when **udp.endPacket()** is executed.

Arduino Program

```
void loop() {
  int reg[6];
  int i;
  if(send_data == 0) {
    int ps = udp.parsePacket();
    if(ps > 0) {
      send_data = 1;
      remote = udp.remoteIP();
    }
  }
  if(send_data == 1) {
    mpu6050.update();
    float gx = mpu6050.getAngleX();
    float gy = mpu6050.getAngleY();
    float gz = mpu6050.getAngleZ();
    snprintf(buf,32,”%f,%f,%f\n”,gx,gy,gz);
    udp.beginPacket(remote,10000);
    udp.print(buf);
    udp.endPacket();
    delay(10);
  }
}
```

Python Code

The same plotting python program will be modified to use the network socket. The UDP protocol will be used.

Line 3 is modified to import the socket library instead of the serial library.

Line 16 sets up the type of network connection to be UDP. Line 17 establishes a connection to the SLATE. This is done to get the local IP of the device making the connection. There can be more than one network device. Line 18 gets the local IP address of the host computer that the python program is hosted. Line 19 closes the network connection.

Line 20 sets up the UDP network type again. Line 21 tells the program to start listening to the network port on the local IP address.

Line 22 sends a packet to the SLATE. This allows the SLATE to get the IP address of the host computer running the python program.

Python Program

```
1  import matplotlib.pyplot as plt
2  import matplotlib.animation as anim
3  import socket
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 = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
17 s.connect((‘192.168.4.1’,80))
18 localip = s.getsockname()[0]
19 s.close()
20 s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
21 s.bind((localip,10000))
22 s.sendto(b‘start’,(‘192.168.4.1’,10000))
```

Python Code

Lines 23, 24 and 25 create plots for each axis of the IMU. The object **ax** refers to the one subplot. Any plot functions to the same object gets plotted on the same subplot.

Lines 26 to 28 set up the labels. Line 29 makes the legend visible.

Lines 31 through 47 are the same as before. The only change is line 32. It is changed to read a UDP packet that has been received. If none has been received, the program waits here. Notice the function returns two values. The first is the packet contents into variable a. The second is the IP address and port from the SLATE.

The rest of the program is not changed.

Python Program

```
23 linex, = ax.plot(xs, xa,label='X')
24 liney, = ax.plot(xs,ya,label='Y')
25 linez, = ax.plot(xs,za,label='Z')
26 plt.title('MPU6050')
27 plt.xlabel('Samples')
28 plt.ylabel('Degrees')
29 ax.legend()
30
31 def animate(i, xa,ya,za):
32     a,d = s.recvfrom(256)
33     b = a.decode('utf-8')
34     c = b.split(',')
35     if len(c) == 3:
36         xa.append(float(c[0]))
37         ya.append(float(c[1]))
38         za.append(float(c[2]))
39         xa = xa[-x_len:]
40         ya = ya[-x_len:]
41         za = za[-x_len:]
42         linex.set_ydata(xa)
43         liney.set_ydata(ya)
44         linez.set_ydata(za)
45         return linex,liney,linez,
46     else:
47         return linex,liney,linez,
48 ani = anim.FuncAnimation(fig,animate,
49     fargs=(xa,ya,za,),
50     interval=1,
51     blit=True)
52 plt.show()  # show the figure
```

WiFi with Matlab

Matlab provides functions for interacting with devices. The interface used in the following example is the USB interface which appears to be a COM port. It is the same COM port used for uploading programs.

The Arduino code from the previous lesson will be used. Reload the code with the WiFi accelerometer program if necessary. You will need to change the **delay()** at the bottom of the program. Set it to 100ms. This is because Matlab is a bit slow and the high data rate will overload Matlab.

```
loop() {
  int reg[6];
  int i;
  client = server.available();
  if(client) {
    Serial.println(“Connected”);
    while(client.connected()) {
       Wire.beginTransmission(0x1c);
      Wire.write(0x01);
      Wire.endTransmission(false);
      while(Wire.available() < 6) {
        delay(1);
      }
      for(i=0;i<6;i++)
        reg[i] = Wire.read();
      short x = (reg[0] << 8) | reg[1];
      short y = (reg[2] << 8) | reg[3];
      short z = (reg[4] << 8) | reg[5];
      x = x >> 2;
      y = y >> 2;
      z = z >> 2;
      float gx = x / 4095.0;
      float gy = y / 4095.0;
      float gz = z / 4095.0;
      client.print(gx,2);
      client.print(“,“);
      client.print(gy,2);
      client.print(“,“);
      client.println(gz,2);
      delay(100);
    }
  }
}
```

Accelerometer Code

WiFi with Matlab

First, three arrays will be created to hold the data for each axis of the accelerometer. A figure will be created to display three plots. Next, variable t will be created and be the network object. A tcp-ip connection is being created. The first argument is the Experimenters Kit IP address. The second argument is the port number. The third argument indicates a network connection and the last argument specifies the program operates as a client. **fopen(t)** connects to the Experimenters Kit.

An infinite loop is created with the **while 1.** A **for** loop is used to collect 50 data samples and fill the arrays.

```
xa = zeros(50,1);
ya = zeros(50,1);
za = zeros(50,1);

figure;
t = tcpip('192.168.4.1',80,'NetworkRole','client');
fopen(t);
while 1
  for b=1:50
    p = fscanf(t,'%e %e %e');
    xa(b) = p(1);
    ya(b) = p(2);
    za(b) = p(3);
  end
  subplot(3,1,1);
  plot(xa);
  title('Accel X');
  axis([1,50,-8200,8200]);
  subplot(3,1,2);
  plot(ya);
  title('Accel Y');
  axis([1,50,-8200,8200]);
  subplot(3,1,3);
  plot(za);
  title('Accel Z');
  axis([1,50,-8200,8200]);
  drawnow;
end
```

WiFi with Matlab

After the 50 samples are collected, the data is plotted in three separate plots. drawnow is executed to update the display.

To stop the Matlab code, click on the Pause button. Then click on the Quit Debugging button. In the command window enter **fclose(t)** and press enter. This properly stops the code and closes the network connection. If this is not done, the program cannot be rerun. If an error indicating the network connection is not available, restart Matlab.

```
xa = zeros(50,1);
ya = zeros(50,1);
za = zeros(50,1);

figure;
t = tcpip('192.168.4.1',80,'NetworkRole','client');
fopen(t);
while 1
  for b=1:50
    p = fscanf(t,'%e %e %e');
    xa(b) = p(1);
    ya(b) = p(2);
    za(b) = p(3);
  end
  subplot(3,1,1);
  plot(xa);
  title('Accel X');
  axis([1,50,-8200,8200]);
  subplot(3,1,2);
  plot(ya);
  title('Accel Y');
  axis([1,50,-8200,8200]);
  subplot(3,1,3);
  plot(za);
  title('Accel Z');
  axis([1,50,-8200,8200]);
  drawnow;
end
```

End