Innovative wireless platforms for the internet of things

Report of Lab3

BLE Sensor Node

Hossein Zahedi Nezhad s309247
Mohammad Eftekhari Pour s307774
Hamid Shabanipour s314041

Professor: Daniele Trinchero

April 2024

1. Introduction

This experiment utilizes the GY-521 module, equipped with both gyroscope and accelerometer functionalities, to measure and analyze the dynamic changes in roll, pitch, and yaw angles. Connected to an ESP32 board, the GY-521 provides precise motion data that is critically analyzed through the Arduino platform, where results are displayed on a Serial Monitor for real-time observation. The ESP32 board, a robust microcontroller with extensive connectivity options, further extends the utility of the data by broadcasting it via Bluetooth Low Energy (BLE). This data is observable using the nRF Application. This setup not only allows for monitoring on a single device but also enables simultaneous observation across multiple devices that are searching for Bluetooth signals and have the nRF Application installed.

The objective of this experiment is to demonstrate the seamless integration of sensor data acquisition with wireless transmission technologies, providing a foundation for further exploration in applications ranging from navigation systems to interactive interfaces.

2. Tasks

1. Connect the ESP32 and the GY-521 module (Research the module components and datasheets)

To complete this task, we need to connect the ESP32 Board and the GY-521, which is both a gyroscope and an accelerometer, together. By utilizing Arduino and serial monitoring, we can observe the data transferred between them. To establish this connection, we will wire the following pins of each device together:

ESP32 Board ↔ **GY-521**
G (Ground) ↔ GND
3V3 ↔ VCC
8 ↔ SCL
19 ↔ SDA

In order to establish connection between sensor and board, the **`wire.begin()`** function is used to assign the 8th and 19th ports of the ESP32-S3 board to the SCL and SDA pins of the GY521 sensor, respectively.

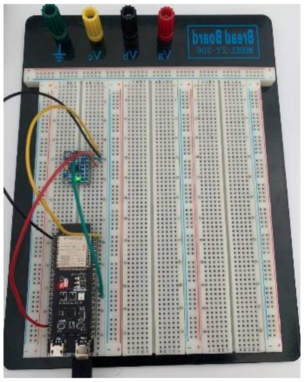
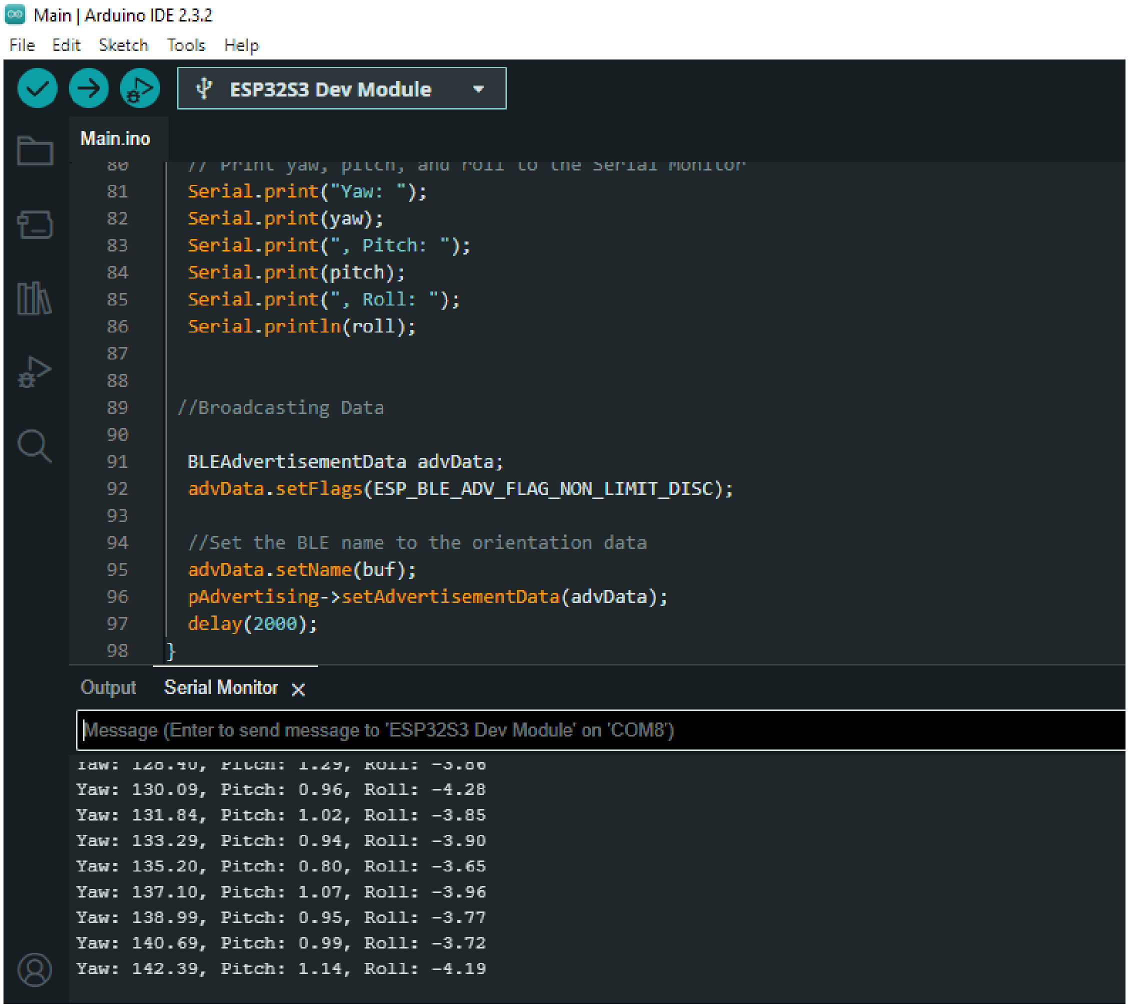


Figure 1. Physical Configuration of Sensor and Board

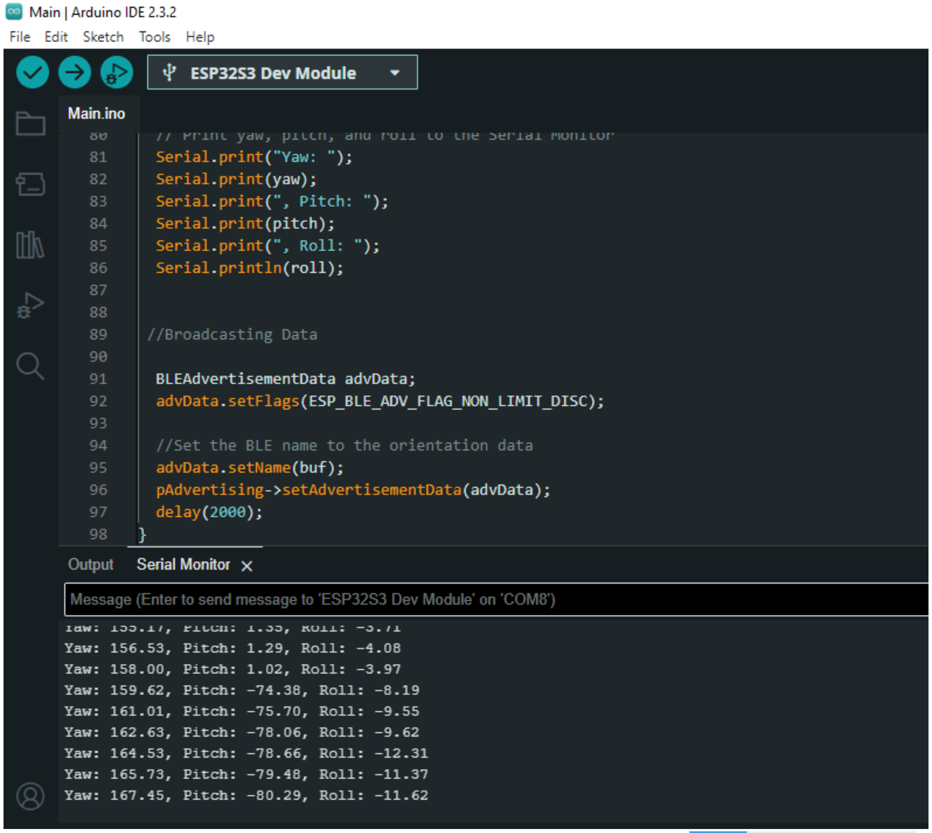
2. Read the accelerometer and gyroscope data through I2C protocol, and calculate the angular orientation of the device.

The serial monitor of Arduino Program illustrates the measured and calculated yaw, roll, pitch and their changes by gyroscope and accelerator.


Main | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ESP32S3 Dev Module
Main.ino
Output
Serial Monitor
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM8')

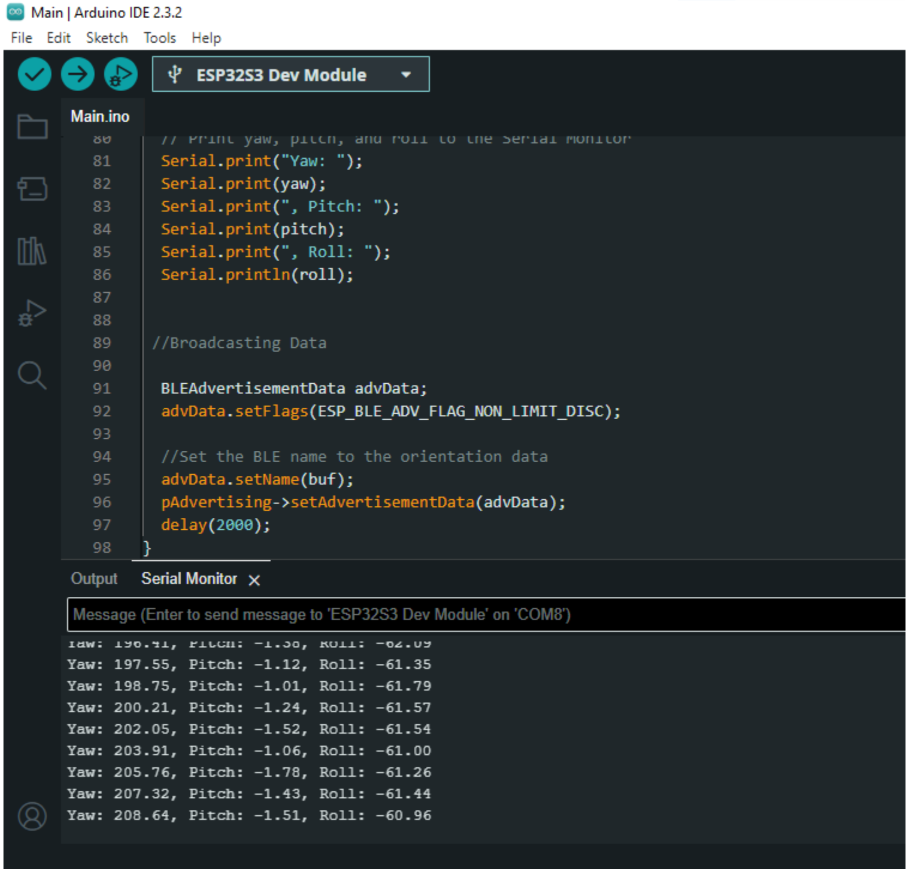
```
  // Print yaw, pitch, and roll to the Serial Monitor
  Serial.print("Yaw: ");
  Serial.print(yaw);
  Serial.print(", Pitch: ");
  Serial.print(pitch);
  Serial.print(", Roll: ");
  Serial.println(roll);


 //Broadcasting Data

  BLEAdvertisementData advData;
  advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

  //Set the BLE name to the orientation data
  advData.setName(buf);
  pAdvertising->setAdvertisementData(advData);
  delay(2000);
}
```

```
Yaw: 128.40, Pitch: 1.29, Roll: -3.86
Yaw: 130.09, Pitch: 0.96, Roll: -4.28
Yaw: 131.84, Pitch: 1.02, Roll: -3.85
Yaw: 133.29, Pitch: 0.94, Roll: -3.90
Yaw: 135.20, Pitch: 0.80, Roll: -3.65
Yaw: 137.10, Pitch: 1.07, Roll: -3.96
Yaw: 138.99, Pitch: 0.95, Roll: -3.77
Yaw: 140.69, Pitch: 0.99, Roll: -3.72
Yaw: 142.39, Pitch: 1.14, Roll: -4.19
```



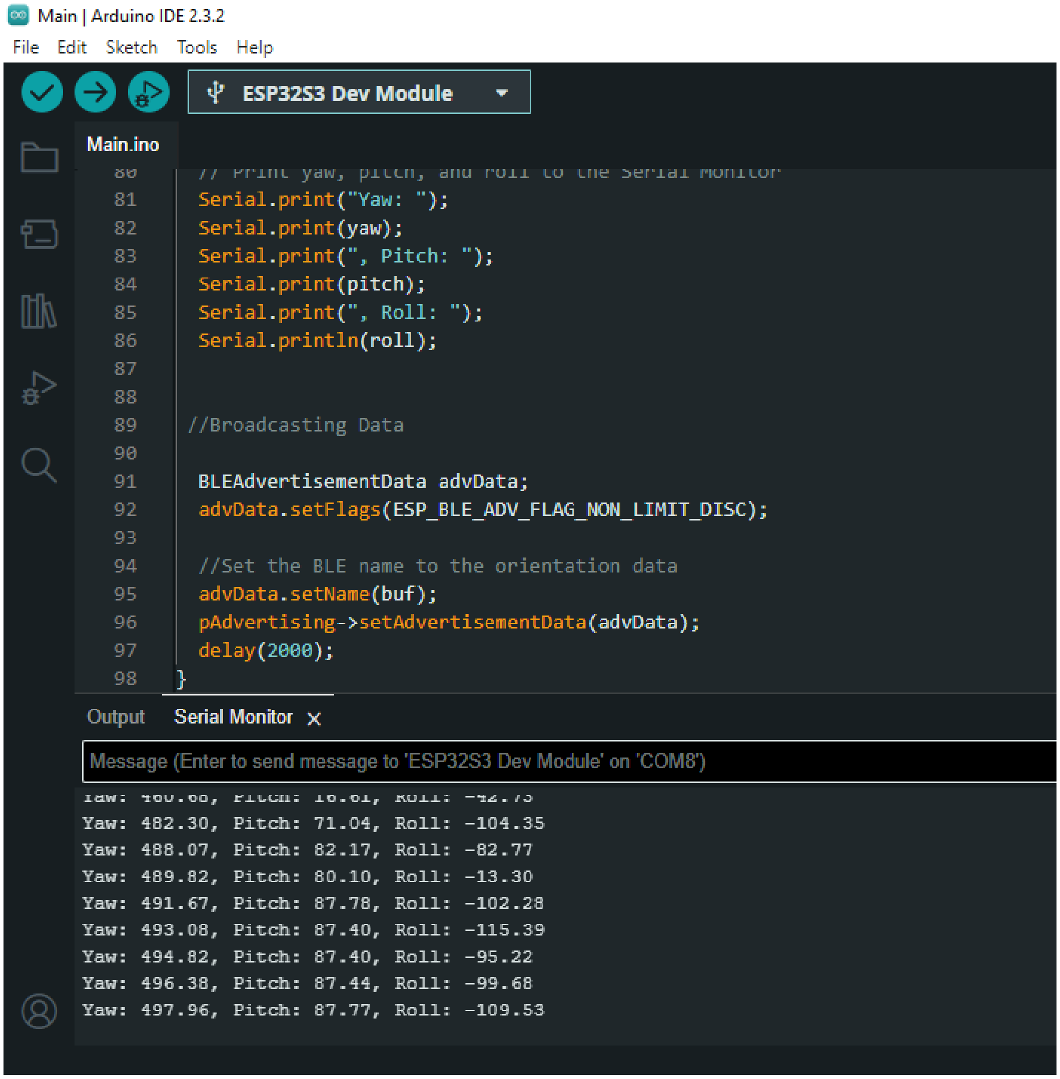
```
  // Print yaw, pitch, and roll to the Serial Monitor
  Serial.print("Yaw: ");
  Serial.print(yaw);
  Serial.print(", Pitch: ");
  Serial.print(pitch);
  Serial.print(", Roll: ");
  Serial.println(roll);


 //Broadcasting Data

  BLEAdvertisementData advData;
  advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

  //Set the BLE name to the orientation data
  advData.setName(buf);
  pAdvertising->setAdvertisementData(advData);
  delay(2000);
}
```

```
Yaw: 155.17, Pitch: 1.35, Roll: -3.71
Yaw: 156.53, Pitch: 1.29, Roll: -4.08
Yaw: 158.00, Pitch: 1.02, Roll: -3.97
Yaw: 159.62, Pitch: -74.38, Roll: -8.19
Yaw: 161.01, Pitch: -75.70, Roll: -9.55
Yaw: 162.63, Pitch: -78.06, Roll: -9.62
Yaw: 164.53, Pitch: -78.66, Roll: -12.31
Yaw: 165.73, Pitch: -79.48, Roll: -11.37
Yaw: 167.45, Pitch: -80.29, Roll: -11.62
```



```
  // Print yaw, pitch, and roll to the Serial Monitor
  Serial.print("Yaw: ");
  Serial.print(yaw);
  Serial.print(", Pitch: ");
  Serial.print(pitch);
  Serial.print(", Roll: ");
  Serial.println(roll);


 //Broadcasting Data

  BLEAdvertisementData advData;
  advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

  //Set the BLE name to the orientation data
  advData.setName(buf);
  pAdvertising->setAdvertisementData(advData);
  delay(2000);
}
```

```
Yaw: 196.41, Pitch: -1.38, Roll: -62.09
Yaw: 197.55, Pitch: -1.12, Roll: -61.35
Yaw: 198.75, Pitch: -1.01, Roll: -61.79
Yaw: 200.21, Pitch: -1.24, Roll: -61.57
Yaw: 202.05, Pitch: -1.52, Roll: -61.54
Yaw: 203.91, Pitch: -1.06, Roll: -61.00
Yaw: 205.76, Pitch: -1.78, Roll: -61.26
Yaw: 207.32, Pitch: -1.43, Roll: -61.44
Yaw: 208.64, Pitch: -1.51, Roll: -60.96
```


Main | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ESP32S3 Dev Module
Main.ino
Output
Serial Monitor
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM8')

```
  // Print yaw, pitch, and roll to the Serial Monitor
  Serial.print("Yaw: ");
  Serial.print(yaw);
  Serial.print(", Pitch: ");
  Serial.print(pitch);
  Serial.print(", Roll: ");
  Serial.println(roll);


 //Broadcasting Data

  BLEAdvertisementData advData;
  advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

  //Set the BLE name to the orientation data
  advData.setName(buf);
  pAdvertising->setAdvertisementData(advData);
  delay(2000);
}
```

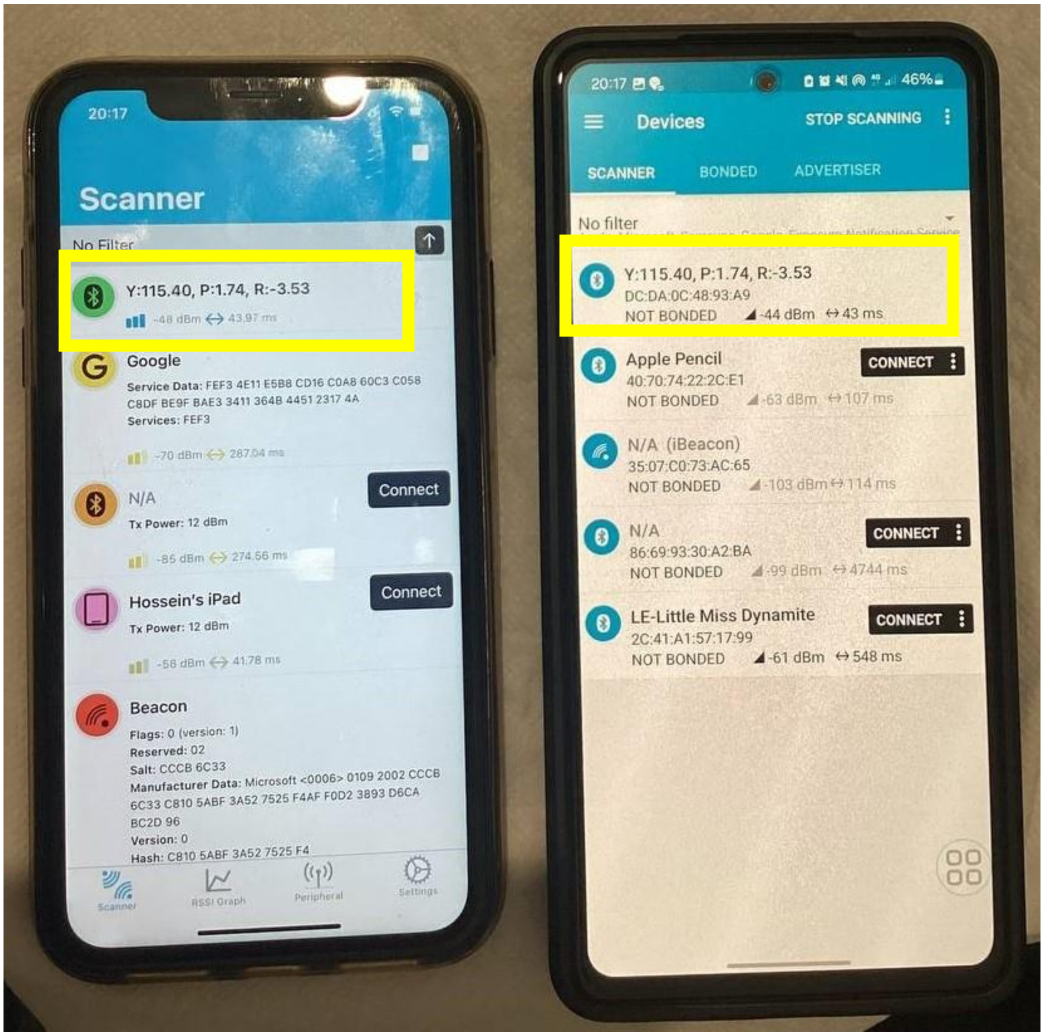
```
Yaw: 480.68, Pitch: 18.81, Roll: -42.75
Yaw: 482.30, Pitch: 71.04, Roll: -104.35
Yaw: 488.07, Pitch: 82.17, Roll: -82.77
Yaw: 489.82, Pitch: 80.10, Roll: -13.30
Yaw: 491.67, Pitch: 87.78, Roll: -102.28
Yaw: 493.08, Pitch: 87.40, Roll: -115.39
Yaw: 494.82, Pitch: 87.40, Roll: -95.22
Yaw: 496.38, Pitch: 87.44, Roll: -99.68
Yaw: 497.96, Pitch: 87.77, Roll: -109.53
```

Figure 2. illustrating Computed Angels through Serial Monitor of Arduino Program

3. Broadcast the angular orientation of the device for roll, pitch, and yaw using BLE

4. Monitor the data from the nRF Connect App from multiple phones simultaneously to verify the data is being broadcasted

As illustrated in the images below, the values measured and computed by the sensors are transmitted via Bluetooth Low Energy which is one of the abilities of ESP32-S3 Boards. The Yaw, Pitch, and Roll values are displayed under the labels Y, P, and R, respectively, and can be viewed through various device using the nRF Application.



Y:115.40, P:1.74, R:-3.53
-48 dBm ↔ 43.97 ms

Google
Service Data: FEF3 4E11 E5B8 CD16 C0A8 60C3 C058 C8DF BE9F BAE3 3411 364B 4451 2317 4A
Services: FEF3
-70 dBm ↔ 287.04 ms

N/A
Tx Power: 12 dBm
-85 dBm ↔ 274.56 ms

Hossein's iPad
Tx Power: 12 dBm
-56 dBm ↔ 41.78 ms

Beacon
Flags: 0 (version: 1)
Reserved: 02
Salt: CCCB 6C33
Manufacturer Data: Microsoft <0006> 0109 2002 CCCB 6C33 C810 5ABF 3A52 7525 F4AF F0D2 3893 D6CA BC2D 96
Version: 0
Hash: C810 5ABF 3A52 7525 F4

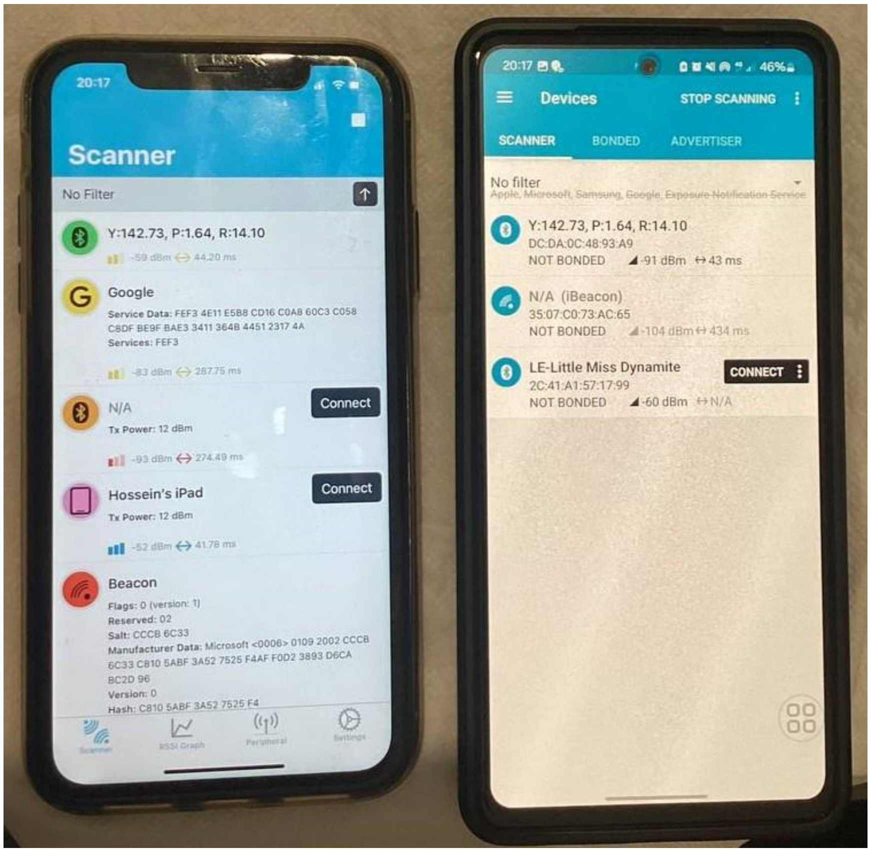
Y:115.40, P:1.74, R:-3.53
DC:DA:0C:48:93:A9
NOT BONDED -44 dBm ↔ 43 ms

Apple Pencil
40:70:74:22:2C:E1
NOT BONDED -63 dBm ↔ 107 ms

N/A (iBeacon)
35:07:C0:73:AC:65
NOT BONDED -103 dBm ↔ 114 ms

N/A
86:69:93:30:A2:BA
NOT BONDED -99 dBm ↔ 4744 ms

LE-Little Miss Dynamite
2C:41:A1:57:17:99
NOT BONDED -61 dBm ↔ 548 ms



20:17

Scanner

No Filter

Y:142.73, P:1.64, R:14.10
-59 dBm ⟷ 44.20 ms

Google
Service Data: FEF3 4E11 E5B8 CD16 C0A8 60C3 C058 C8DF BE9F BAE3 3411 364B 4451 2317 4A
Services: FEF3
-83 dBm ⟷ 287.75 ms

N/A
Connect
Tx Power: 12 dBm
-93 dBm ⟷ 274.49 ms

Hossein's iPad
Connect
Tx Power: 12 dBm
-52 dBm ⟷ 41.78 ms

Beacon
Flags: 0 (version: 1)
Reserved: 02
Salt: CCCB 6C33
Manufacturer Data: Microsoft <0006> 0109 2002 CCCB 6C33 C810 5ABF 3A52 7525 F4AF F0D2 3893 D6CA BC2D 96
Version: 0
Hash: C810 5ABF 3A52 7525 F4

20:17 46%

Devices STOP SCANNING

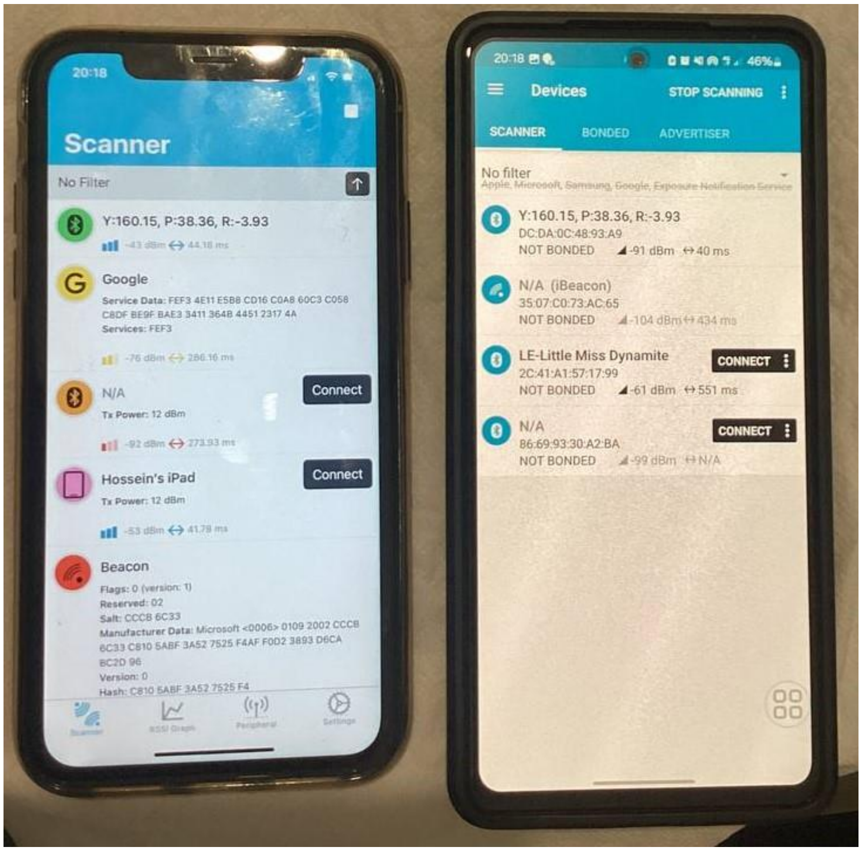
SCANNER BONDED ADVERTISER

No filter
~~Apple, Microsoft, Samsung, Google, Exposure Notification Service~~

Y:142.73, P:1.64, R:14.10
DC:DA:0C:48:93:A9
NOT BONDED -91 dBm ↔ 43 ms

N/A (iBeacon)
35:07:C0:73:AC:65
NOT BONDED -104 dBm ↔ 434 ms

LE-Little Miss Dynamite
CONNECT
2C:41:A1:57:17:99
NOT BONDED -60 dBm ↔ N/A



20:18

Scanner

No Filter

Y:160.15, P:38.36, R:-3.93
-43 dBm ⟷ 44.16 ms

Google
Service Data: FEF3 4E11 E5B8 CD16 C0A8 60C3 C058 C8DF BE9F BAE3 3411 364B 4451 2317 4A
Services: FEF3
-76 dBm ⟷ 286.16 ms

N/A
Connect
Tx Power: 12 dBm
-92 dBm ⟷ 273.93 ms

Hossein's iPad
Connect
Tx Power: 12 dBm
-53 dBm ⟷ 41.78 ms

Beacon
Flags: 0 (version: 1)
Reserved: 02
Salt: CCCB 6C33
Manufacturer Data: Microsoft <0006> 0109 2002 CCCB 6C33 C810 5ABF 3A52 7525 F4AF F0D2 3893 D6CA BC2D 96
Version: 0
Hash: C810 5ABF 3A52 7525 F4

20:18 46%

Devices STOP SCANNING

SCANNER BONDED ADVERTISER

No filter
~~Apple, Microsoft, Samsung, Google, Exposure Notification Service~~

Y:160.15, P:38.36, R:-3.93
DC:DA:0C:48:93:A9
NOT BONDED -91 dBm ↔ 40 ms

N/A (iBeacon)
35:07:C0:73:AC:65
NOT BONDED -104 dBm ↔ 434 ms

LE-Little Miss Dynamite
CONNECT
2C:41:A1:57:17:99
NOT BONDED -61 dBm ↔ 551 ms

N/A
CONNECT
86:69:93:30:A2:BA
NOT BONDED -99 dBm ↔ N/A

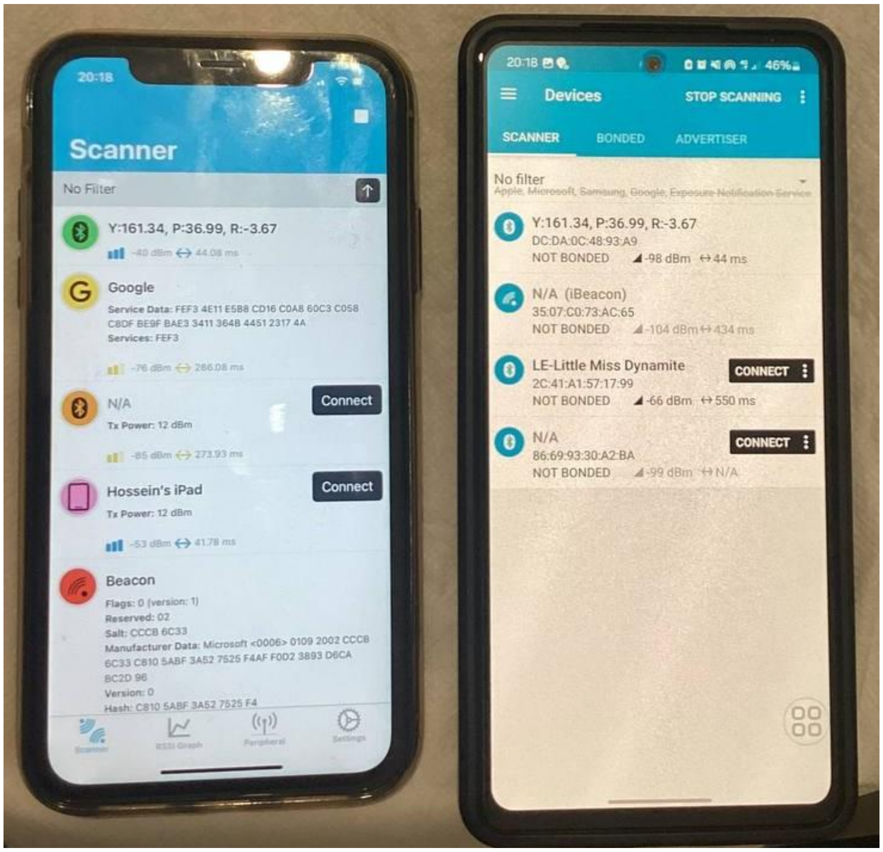
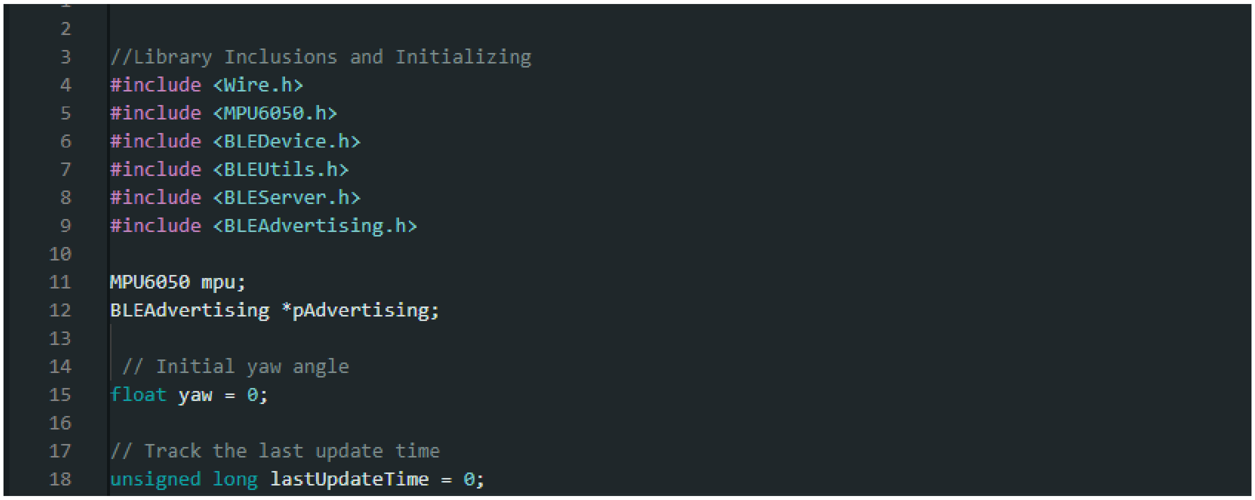


Figure 3. Broadcasting Data through BLE and Observing it by devices simultaneously

3. Code description

1. Library Inclusions and Initializing



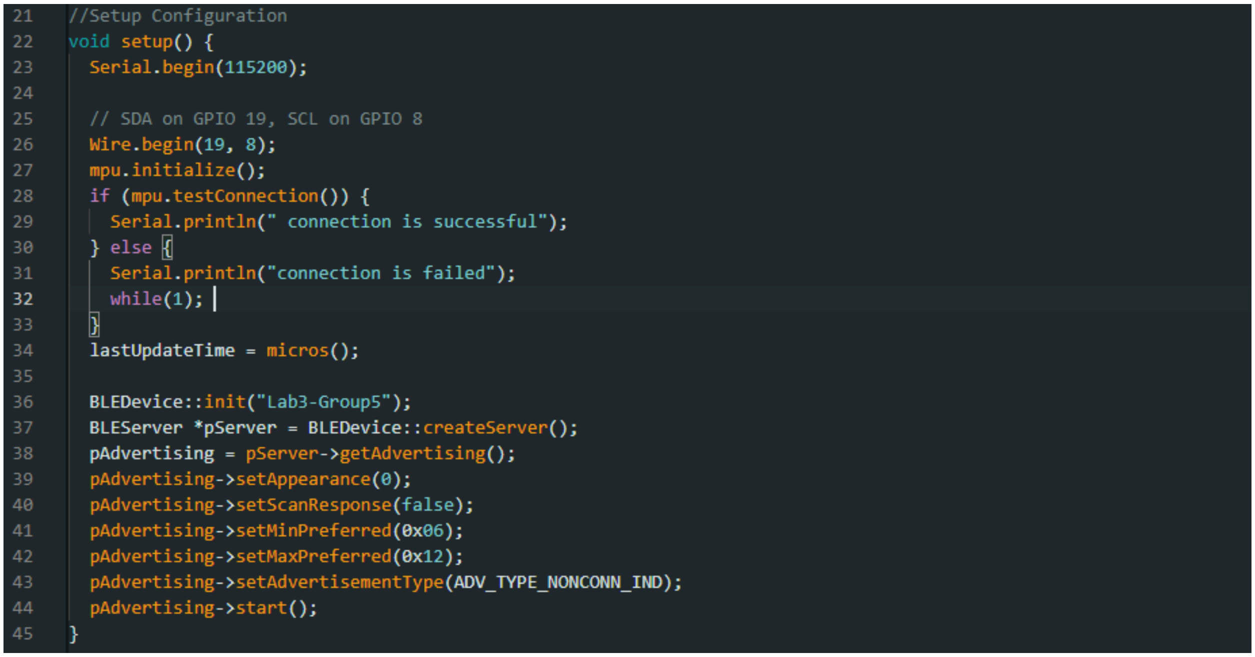
```

//Library Inclusions and Initializing
#include <Wire.h>
#include <MPU6050.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLEAdvertising.h>

MPU6050 mpu;
BLEAdvertising *pAdvertising;

 // Initial yaw angle
float yaw = 0;

// Track the last update time
unsigned long lastUpdateTime = 0;
```

The required libraries are included in this section for controlling BLE capabilities **(BLEDevice.h, BLEUtils.h, BLEServer.h, and BLEAdvertising.h)**, handling I2C communication **(Wire.h)**, and interacting with the MPU6050 sensor **(MPU6050.h)**. Additionally, we initialize the sensor's mpu object here.

Furthermore, we set the starting yaw angle to zero in order to get ready to compute orientation using sensor information. Actually, in this configuration, the yaw angle is determined by integrating the gyroscope data over time. It means that we begin from a known reference point and constantly update this angle based on the gyroscope's z-axis measurements.

Roll and pitch are obtained directly from the accelerometer data at each measurement, in fact, they are completely recalculated using the most recent sensor readings. due to the fact that they dont rely on their prior values.

2. Setup Configuration

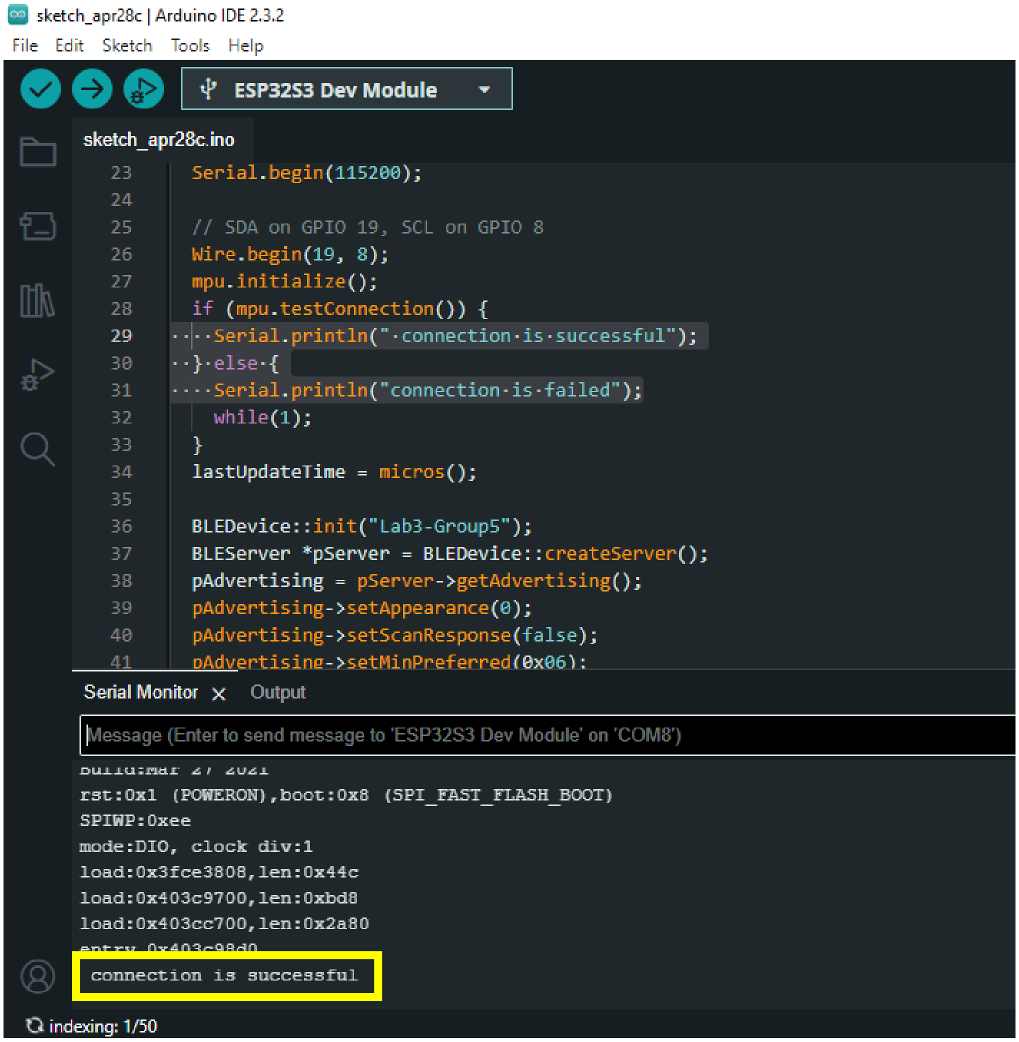


```
//Setup Configuration
void setup() {
  Serial.begin(115200);

  // SDA on GPIO 19, SCL on GPIO 8
  Wire.begin(19, 8);
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println(" connection is successful");
  } else {
    Serial.println("connection is failed");
    while(1);
  }
  lastUpdateTime = micros();

  BLEDevice::init("Lab3-Group5");
  BLEServer *pServer = BLEDevice::createServer();
  pAdvertising = pServer->getAdvertising();
  pAdvertising->setAppearance(0);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMaxPreferred(0x12);
  pAdvertising->setAdvertisementType(ADV_TYPE_NONCONN_IND);
  pAdvertising->start();
}
```

As mentioned, to connect the ESP32 board and the sensor, pins 19 and 8 of the ESP32 board are connected to SDA and SCL from the sensor, respectively.

To ensure that the ESP32S3 and GY-521 are connected together we use function **mpu.testConnection()** and

- If the connection is successful, it logs " connection is successful" to the serial monitor.
- If the connection fails, it logs " connection is failed".


sketch_apr28c | Arduino IDE 2.3.2
File Edit Sketch Tools Help
ESP32S3 Dev Module
sketch_apr28c.ino
Serial Monitor
Output
Message (Enter to send message to 'ESP32S3 Dev Module' on 'COM8')
indexing: 1/50

```
  Serial.begin(115200);

  // SDA on GPIO 19, SCL on GPIO 8
  Wire.begin(19, 8);
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println(" connection is successful");
  } else {
    Serial.println("connection is failed");
    while(1);
  }
  lastUpdateTime = micros();

  BLEDevice::init("Lab3-Group5");
  BLEServer *pServer = BLEDevice::createServer();
  pAdvertising = pServer->getAdvertising();
  pAdvertising->setAppearance(0);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
```

```
build:Mar 27 2021
rst:0x1 (POWERON),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fce3808,len:0x44c
load:0x403c9700,len:0xbd8
load:0x403cc700,len:0x2a80
entry 0x403c98d0
connection is successful
```

Figure 4. Ensuring correct configuration of components

Furthermore,

We create a BLE server on the device to manage incoming and outgoing connections and interactions. This server is responsible for handling all the BLE services and characteristics.

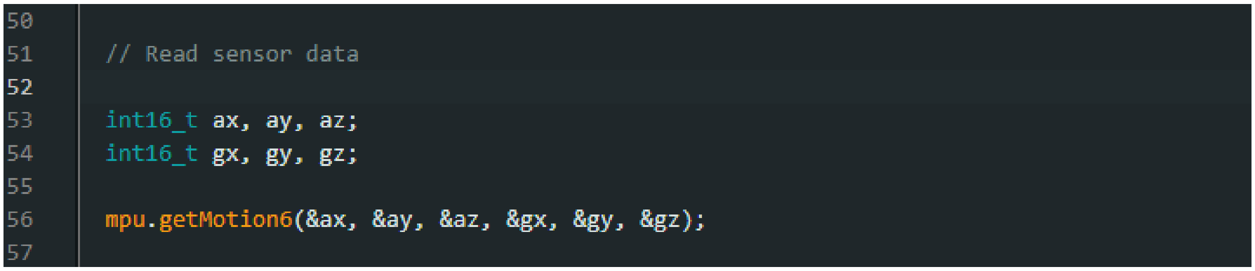
The name **"Lab3-Group5"** is considered for the name of the device.

`pAdvertising->star t()` activates the advertising process, allowing the device to begin broadcasting its BLE advertisement packets.

`setMinPreferred(0x06)` and **`setMaxPreferred(0x12)`** establish the ideal minimum and maximum connection intervals, respectively. These parameters determine the frequency at which the device can enter connection intervals.

In general, this piece of code is essential for configuring the device to function as a beacon that broadcast orientation information. This method is frequently applied in situations—like sensor broadcasting—where the data must be made publicly accessible without involving two-way communication.

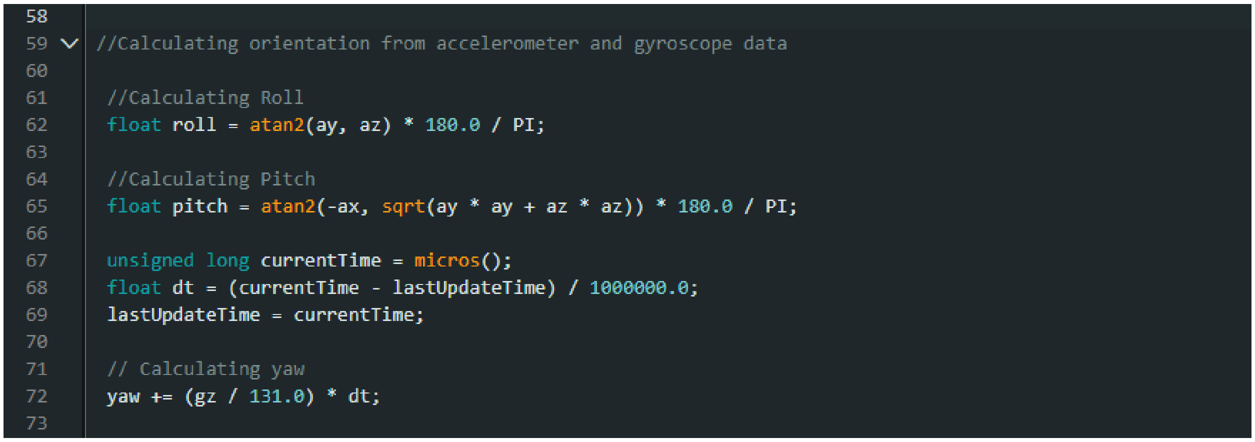
3. Read sensor data



```
// Read sensor data

int16_t ax, ay, az;
int16_t gx, gy, gz;

mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
```

The **accelerometer data** is stored along the **X**, **Y**, and **Z axes** using the **variables ax**, **ay**, and **az.** For the **gyroscope data**, the **variables gx**, **gy**, and **gz** are along the **same axes**. The function **`mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz)`** is used to obtain the six motion data points from the Sensor. By reference, the data is read into the specified variables.

4. Calculating orientation (Roll, Pitch, Yaw)

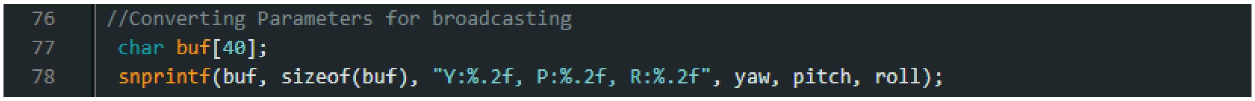


```
//Calculating orientation from accelerometer and gyroscope data

 //Calculating Roll
 float roll = atan2(ay, az) * 180.0 / PI;

 //Calculating Pitch
 float pitch = atan2(-ax, sqrt(ay * ay + az * az)) * 180.0 / PI;

 unsigned long currentTime = micros();
 float dt = (currentTime - lastUpdateTime) / 1000000.0;
 lastUpdateTime = currentTime;

 // Calculating yaw
 yaw += (gz / 131.0) * dt;
```

Roll is calculated by taking the arctangent of the ratio between the accelerometer's Y-axis and Z-axis values. This angle is then converted from radians to degrees. In addition, the pitch angle is determined by taking the arctangent of the negative X-axis value and the square root of the sum of the squares of the Y- and Z-axis values. In this part, after computing **dt**, **`lastUpdateTime`** is updated with the current time to ensure accurate interval measurements for the next loop iteration.

It should be mentioned that, in order to calculate ***yaw***, the variable gz, which is expressed in raw sensor units, represents the gyroscope's measurement of angular velocity around the Z-axis. The raw gyroscope data is converted to degrees per second using the formula (gz / 131.0). Multiplying this by dt gives the change in angle (in degrees) since the last measurement.

5) Converting Data



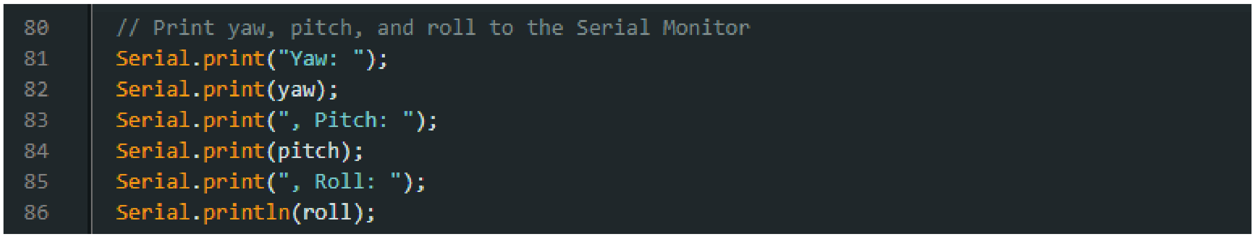
```
//Converting Parameters for broadcasting
 char buf[40];
 snprintf(buf, sizeof(buf), "Y:%.2f, P:%.2f, R:%.2f", yaw, pitch, roll);
```

To hold the formatted string containing the orientation values, we create a buffer with the size of 40 bytes in order to make sure there is enough space to keep the data and Prevent overflow. Then Using **snprintf(buf, sizeof(buf), "Y:%.2f, P:%.2f, R:%.2f", yaw, pitch, roll)** function, Each of these floating-point values (Roll, Pitch, Yaw) is formatted to two decimal places and combined into a single string value. For example, if the calculated parameters through Sensor is:

yaw = 58.32 degrees, pitch = -29.85 degrees, and roll = 65.31 degrees

The string stored in buffer would be "**Y:58.32, P:-29.85, R:65.31**"

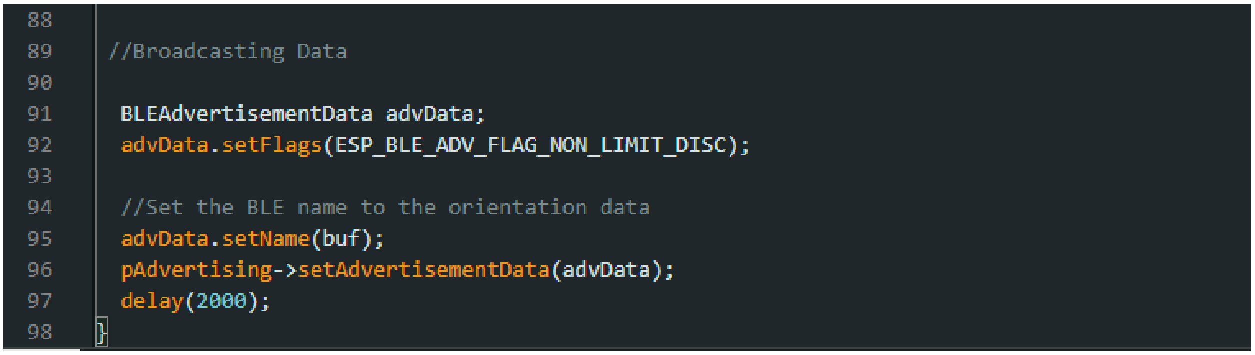
6)Displaying Data in Serial Monitor



```
 // Print yaw, pitch, and roll to the Serial Monitor
 Serial.print("Yaw: ");
 Serial.print(yaw);
 Serial.print(", Pitch: ");
 Serial.print(pitch);
 Serial.print(", Roll: ");
 Serial.println(roll);
```

This block of code is designed to display the values of yaw, pitch, and roll on the Arduino's Serial Monitor.

7) Broadcasting Orientation Data

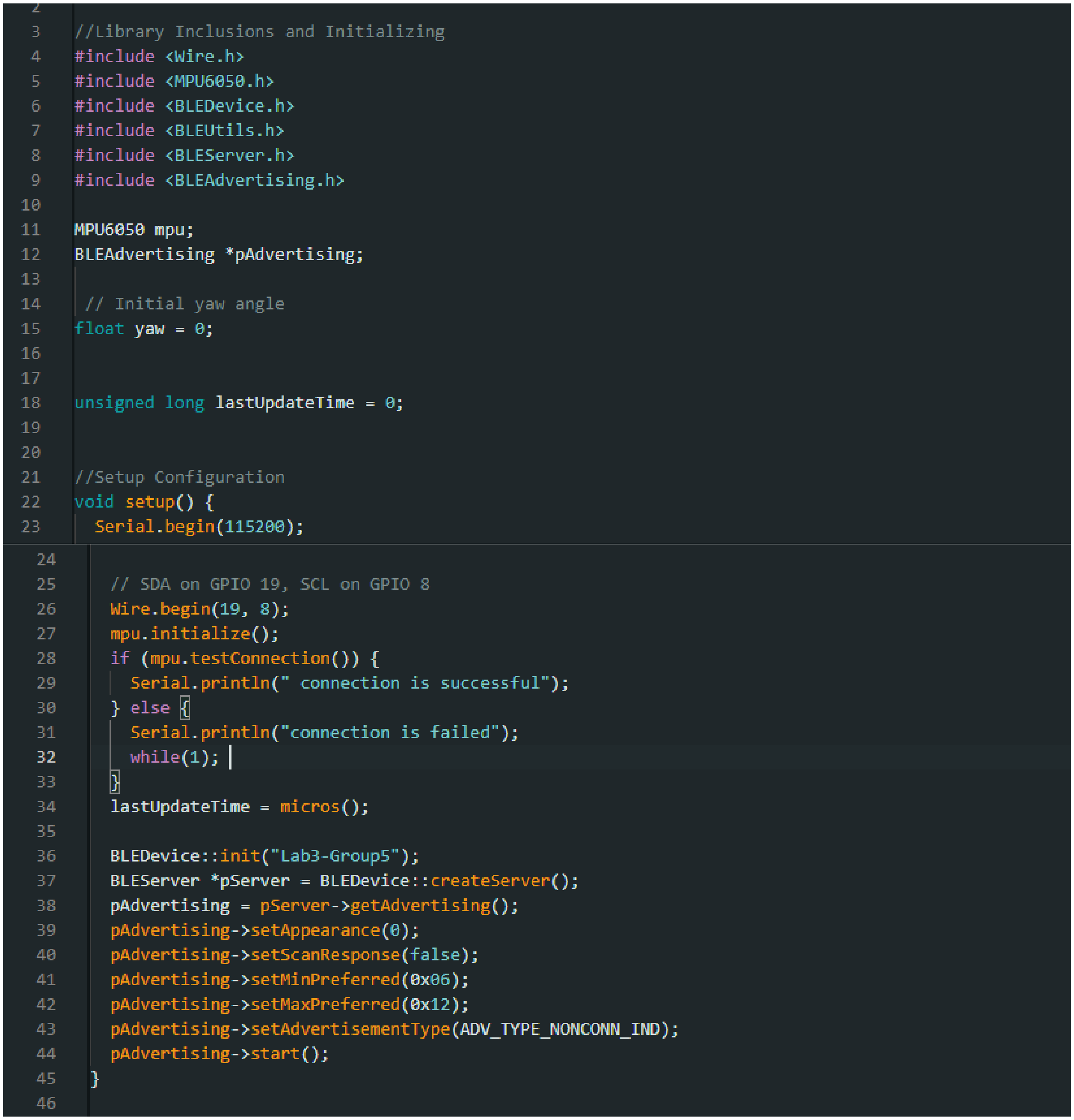


```

//Broadcasting Data

 BLEAdvertisementData advData;
 advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

 //Set the BLE name to the orientation data
 advData.setName(buf);
 pAdvertising->setAdvertisementData(advData);
 delay(2000);
}
```

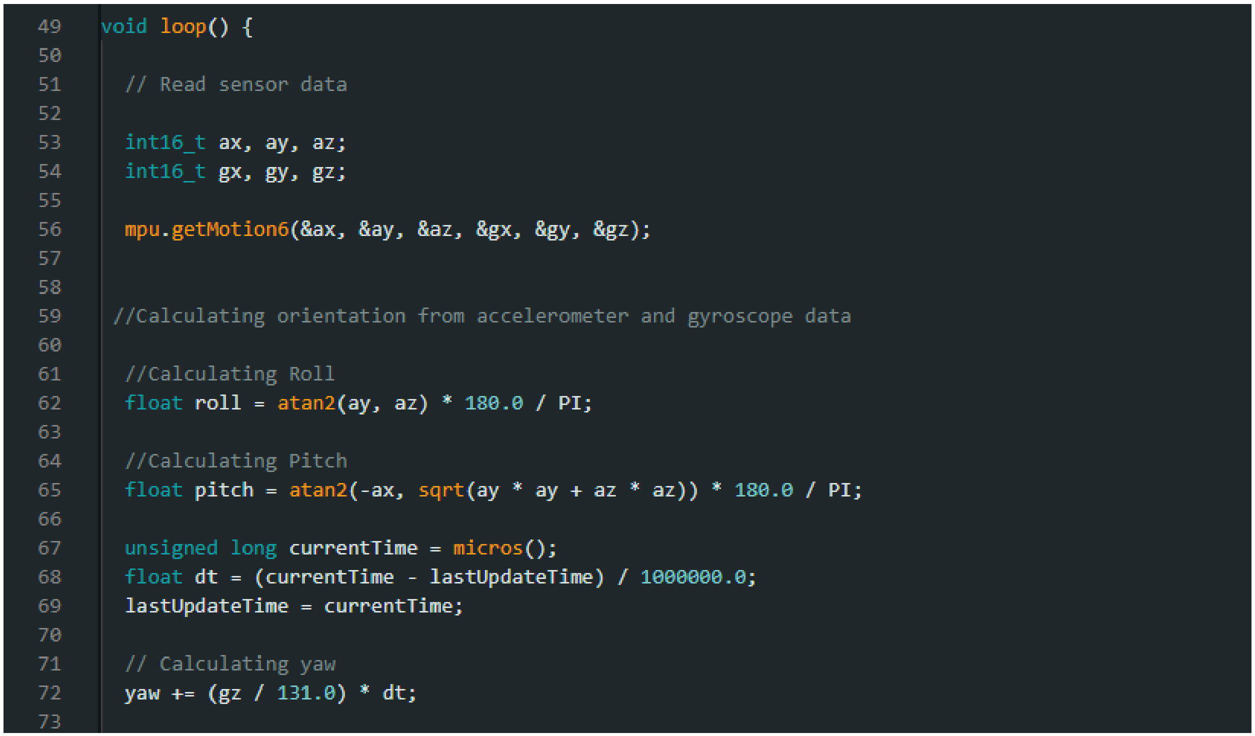
The main Duty of the Last Part of the code is broadcasting the orientation data (yaw, pitch, and roll) via Bluetooth Low Energy (BLE).

To accomplish that **advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);** function helps us by setting the advertisement to "non-limited discoverable mode", allowing the device to be discoverable by other BLE devices without time restrictions. Then we set the name of the BLE device to the string stored in buffer which contains formatted orientation data (yaw, pitch, roll) using **advData.setName(buf)** function. This makes the orientation data visible directly in the BLE advertisement packet, allowing other devices to see these values when they scan for BLE devices. Moreover, we defined a 2000-millisecond (2-second) delay in the execution of the code.

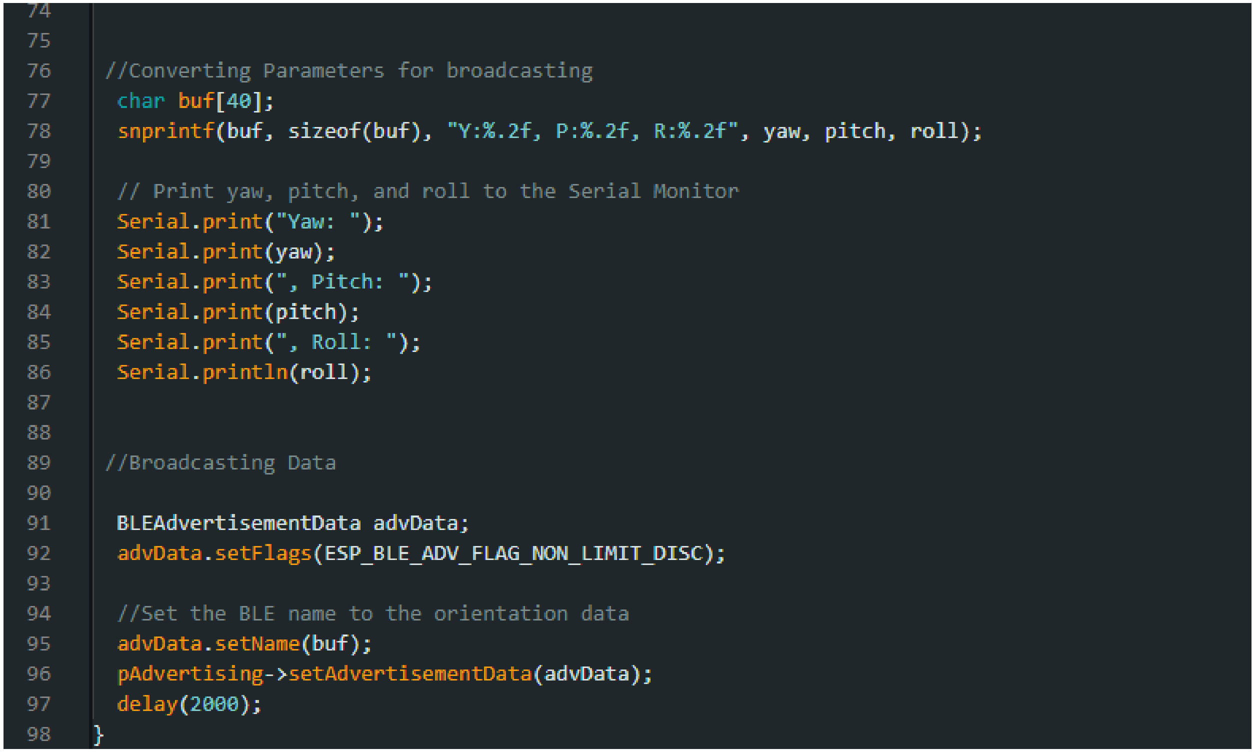
4. Code



```
//Library Inclusions and Initializing
#include <Wire.h>
#include <MPU6050.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#include <BLEAdvertising.h>

MPU6050 mpu;
BLEAdvertising *pAdvertising;

 // Initial yaw angle
float yaw = 0;


unsigned long lastUpdateTime = 0;


//Setup Configuration
void setup() {
  Serial.begin(115200);

  // SDA on GPIO 19, SCL on GPIO 8
  Wire.begin(19, 8);
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println(" connection is successful");
  } else {
    Serial.println("connection is failed");
    while(1);
  }
  lastUpdateTime = micros();

  BLEDevice::init("Lab3-Group5");
  BLEServer *pServer = BLEDevice::createServer();
  pAdvertising = pServer->getAdvertising();
  pAdvertising->setAppearance(0);
  pAdvertising->setScanResponse(false);
  pAdvertising->setMinPreferred(0x06);
  pAdvertising->setMaxPreferred(0x12);
  pAdvertising->setAdvertisementType(ADV_TYPE_NONCONN_IND);
  pAdvertising->start();
}

```



```
void loop() {

  // Read sensor data

  int16_t ax, ay, az;
  int16_t gx, gy, gz;

  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);


 //Calculating orientation from accelerometer and gyroscope data

  //Calculating Roll
  float roll = atan2(ay, az) * 180.0 / PI;

  //Calculating Pitch
  float pitch = atan2(-ax, sqrt(ay * ay + az * az)) * 180.0 / PI;

  unsigned long currentTime = micros();
  float dt = (currentTime - lastUpdateTime) / 1000000.0;
  lastUpdateTime = currentTime;

  // Calculating yaw
  yaw += (gz / 131.0) * dt;

```



```


 //Converting Parameters for broadcasting
  char buf[40];
  snprintf(buf, sizeof(buf), "Y:%.2f, P:%.2f, R:%.2f", yaw, pitch, roll);

  // Print yaw, pitch, and roll to the Serial Monitor
  Serial.print("Yaw: ");
  Serial.print(yaw);
  Serial.print(", Pitch: ");
  Serial.print(pitch);
  Serial.print(", Roll: ");
  Serial.println(roll);


 //Broadcasting Data

  BLEAdvertisementData advData;
  advData.setFlags(ESP_BLE_ADV_FLAG_NON_LIMIT_DISC);

  //Set the BLE name to the orientation data
  advData.setName(buf);
  pAdvertising->setAdvertisementData(advData);
  delay(2000);
}
```