Dr. Lei Lei
College of Science and Engineering
Cairns, Queensland, 4878 Australia

Discipline of Electrical & Computer Engineering

ASSESSMENT COVER SHEET

Student details

Surname(s)	Given names	Student's I.D.
Hiyas	Bienvenido Jr	13824819

Subject details

Subject code	EE3901

Subject name	Sensor Technologies

Title of the work	EE5901 (Advanced) Sensor Technologies Final Project

Name of Coordinator	Dr. Lei Lei

Due Date: 29 May 2020	Date submitted: 29 May 2020

Student's statement:

I/We certify that :

- I have not plagiarized the work of others;
- I have not participated in unauthorized collusion when preparing this work.

Signature(s) …Bienvenido Hiyas Jr……
……

EE5901 (Advanced) Sensor Technologies Final Project

Bienvenido Hiyas Jr. (13824819)

I. Introduction

THIS document is a report about the final project for EE5901 Sensor Technologies subject using Arduino development boards to communicate with different sensors and let them communicate with each other too using a cloud server/platform on mobile phones called Blynk. The Arduino Integrated Development Environment (IDE) is a cross-platform application that is written in functions from C and C++ and is used to write and upload programs to Arduino compatible boards. Furthermore, this project also uses an IMU or Inertial Measurement Unit and display its orientation in MATLAB. MATLAB is programming platform designed specifically for engineers and scientists. It is a matrix-based language allowing the most natural expression of computational mathematics according to MathWorks website (https://au.mathworks.com).

The project uses an Arduino Uno Development board, a WEMOS ESP32 Development Board, an inertial measurement unit (IMU) MPU9250/MPU6050 sensor, Capacitive Soil Moisture Sensor v1.2, usb cables, connectors, breadboard and laptop/pc.

The project aims to; understand the structure and working principle of the different boards and sensors mentioned above and how they are used in the projects that will discussed later on the report; display the orientation of the inertial measurement unit (IMU) MPU9250/MPU6050 in MATLAB ; develop a wireless soil moisture (or temperature and humidity) monitoring sensor node using the provided ESP board and soil moisture/DHT sensor and connect it to the Blynk cloud/platform and; develop a wireless sensor network with two devices and upload data to Blynk Platform and display it in the Blynk Moble App and use the first device to control the other device.

II. Research

A. WEMOS ESP32

ESP32 is a Wi-Fi and Bluetooth system-on-chip (SoC) with industry-leading RF performance, low power consumption and high integration. The core processor ESP32 provides a complete 802.11 b/g/n/e/i wireless local area network (WLAN) and Bluetooth 4.2 solution with minimal physical size[1]. Designed for low-power and mobile consumer electronics, wearable and IoT devices, this chip integrates all the features of WLAN and Bluetooth on-chip, with low cost and good layout. ESP32 also provides an open platform that allows users to flexibly customize functions for different application scenarios.

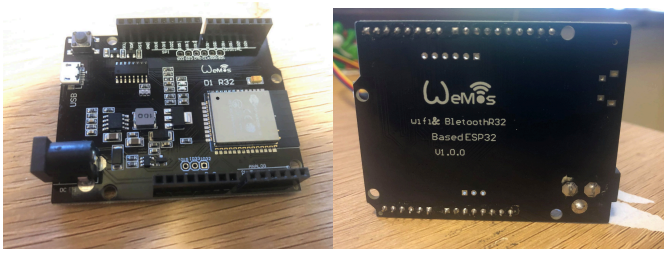

Figure 1: Actual picture of WeMos ESP32 Development Board used.

In order for the ESP32 board to be used like an Arduino board and use its Wi-Fi communication feature, the ESP8266 library, drivers and board manager should be configured and installed in the Arduino IDE software. To do this you can follow the steps on this link: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

B. CAPACITIVE SOIL MOISTURE SENSOR

This is an analog capacitive soil moisture sensor which measures soil moisture levels by capacitive sensing, i.e. capacitance is varied on the basis of water content present in the soil. The capacitance is converted into voltage level basically from 1.2V to 3.0V maximum. The advantage of Capacitive Soil Moisture Sensor is that they are made of a corrosion-resistant material giving it a long service life compared to the resistive type soil moisture sensor[2].

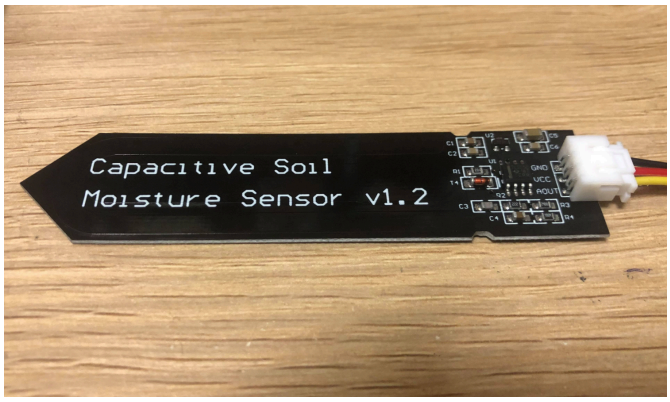

Figure 2: Actual Picture of Capacitive Soil Moisture Sensor v1.2

In addition, this sensor includes an on-board voltage regulator which gives it an operating voltage range of **3.3 ~ 5.5V.** It is perfect for low-voltage **microcontroller** with both 3.3V and 5V power supply. The **Capacitive soil moisture sensor** also has one conductive copper plate in the center and then a ground plate that goes around the outside.

Features & Specifications

1. Supports 3-Pin Sensor interface
2. Analog output
3. Operating Voltage: DC 3.3-5.5V
4. Output Voltage: DC 0-3.0V
5. Interface: PH2.0-3P
6. Size: 99x16mm/3.9x0.63″

C. MPU 9250

MPU-9250 is a multi-chip module (MCM) consisting of two dies integrated into a single QFN package. One die houses the 3-Axis gyroscope and the 3-Axis accelerometer. The other die houses the AK8963 3-Axis magnetometer from Asahi Kasei Microdevices Corporation. Hence, the MPU-9250 is a 9-axis Motion Tracking device that combines a 3-axis gyroscope, 3-axis accelerometer, 3-axis magnetometer and a Digital Motion Processor™ (DMP) all in a small 3x3x1mm package available as a pin-compatible upgrade from the MPU6515[3].

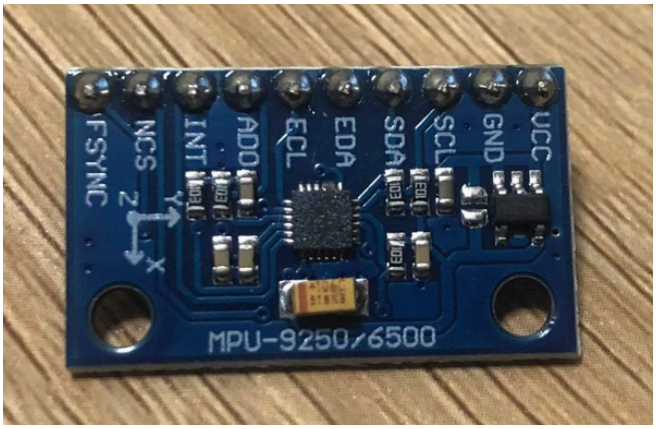

Figure 3: Actual Picture MPU-9250

The accelerometer measures acceleration, the gyroscope measures angular velocity and the magnetometer measures magnetic field in x-,y-, and z- axis. The axis on the sensor depends on the make of the sensor and in this case, since we are using the InvenSense MPU-9250 IMU sensor and below figure 4 are the axis of the sensor[4].

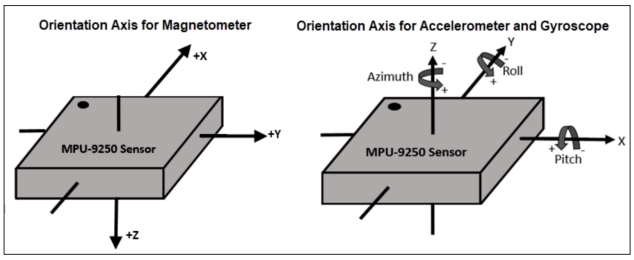

Figure 4: Orientation Axis of Magnetometer, Accelerometer and Gyroscope

In addition, Sensor Fusion algorithms used in this project use North-East-Down(NED) as a fixed,

parent coordinate system. In the NED reference frame, the X-axis points north, the Y-axis points east, and the Z-axis points down. Depending on the algorithm, north may either be the magnetic north or true north. The algorithms in this example use the magnetic north. The algorithms used here expects all the sensors in the object to have their axis aligned and is in accordance with NED convention.

MPU-9250 has two devices, the magnetometer and the accelerometer-gyroscope, on the same board. The axes of these devices are different from each other. The magnetometer axis is aligned with the NED coordinates. The axis of the accelerometer-gyroscope is different from magnetometer in MPU-9250. The accelerometer and the gyroscope axis need to be swapped and/or inverted to match the magnetometer axis

D. Blynk Iot Mobile Platform

Blynk is a hardware-agnostic IoT platform with white-label mobile apps, private clouds, device management, data analytics, and machine learning. According the Blynk website (https://blynk.io/), it is the most popular IoT platform to connect your devices to the cloud, design apps to control them, analyze telemetry data, and manage your deployed products at scale.

To use Blynk, first , you need to install the Blynk app. It is available for iOS and Android. Then, Install the Blynk Library. Blynk Library is an extension that runs on top of your hardware application. It handles all the connection routines and data exchange between your hardware, Blynk Cloud, and your app project, no iOS or Android coding required. After that, Connect the hardware. To get your hardware online and connect it to Blynk Cloud, you would need a device Authentication Token. Once you download the app you will be able to generate Auth Token for every device. Finally, you can now use Blynk. You can learn Blynk basics that are available online. Blynk also provides a lot of examples to get you started.

III. IMU Orientation in MATLAB

This project shows how to get data from the MPU-9250 IMU sensor and use the 6 -axis and 9-axis fusion algorithms in the sensor data to compute the orientation of the device then finally plot the orientation in animation in MATLAB.

A. Hardware Setup

First, the MPU-925 is connected to the Arduino board and then the Arduino board is connected to the laptop. Below is the hardware connection and actual setup of Figure 5 and figure 6.

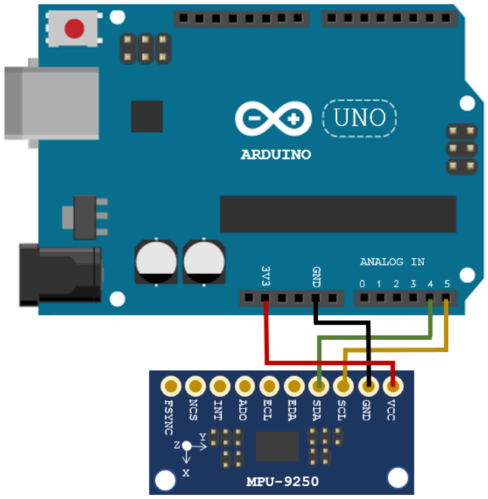

Figure 5: Hardware Connection taken from

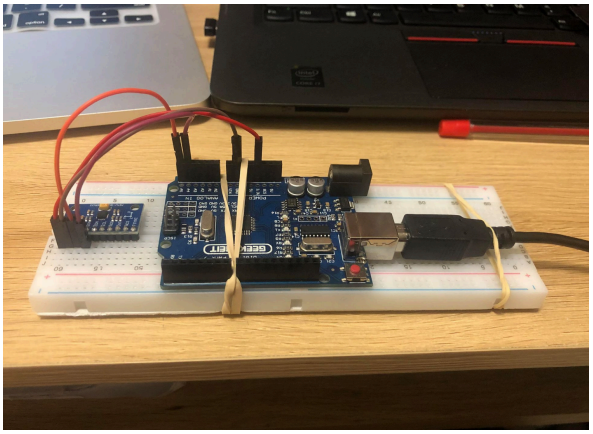

Figure 6: Actual setup of MPU-9250 and Arduino Board connected to laptop

B. Codes used in MATLAB

Then, install and open MATLAB in the laptop and start coding for the project. The project uses two programs files, first is for the compensation of the hard Iron distortions which are produced by materials that create a magnetic field, resulting in shifting the origin on the response surface of the sensor. The codes below are used to get the corrected x, y and z axis of the sensor's magnetometer. Make sure to follow the instruction in green text below and note the corrected values of x, y and z. Make sure to Rotate the sensor around each axis from 0 to 360 degrees. Take 2-3 rotations for improve accuracy.

First you need to create an arduino object and include the I2C library then create an MPU-9250 sensor object.

```
A = arduino('COM3', 'Uno', 'Libraries', 'I2C');
fs = 100; % Sample Rate in Hz
imu = mpu9250(a,'SampleRate',fs,'OutputFormat','matrix');

tic;
stopTimer = 100;
magReadings=[];
while(toc<stopTimer)
   [accel,gyro,mag] = read(imu);
   magReadings = [magReadings;mag];
end
magx_min = min(magReadings(:,1));
magx_max = max(magReadings(:,1));
magx_correction = (magx_max+magx_min)/2;
magy_min = min(magReadings(:,2));
magy_max = max(magReadings(:,2));
magy_correction = (magy_max+magy_min)/2;
magz_min = min(magReadings(:,3));
magz_max = max(magReadings(:,3));
magz_correction = (magz_max+magz_min)/2;
```

Basically, the code above, gets the mean of x, y and z axis of the magnetometer. These are used to correct the hard-iron distortion of the IMU.

Then the second code uses AHRS Filter. An attitude and heading reference system (AHRS) consist of a 9-axis system that uses an accelerometer, gyroscope, and magnetometer to compute orientation of the device. It produces a smoothly changing estimate of orientation of the device, while correctly estimating the north direction. The ahrsfilter has the ability to remove gyroscope bias that drifts the orientation and can also detect and reject mild magnetic jamming[3].

```
function [accel,gyro,mag] = readSensorDataMPU9250(imu)
   %below and remove hard iron   distortion
   magx_correction = 35.039062500000000;
   magy_correction = 14.414062500000000;
   magz_correction = -22.691015625000000;
   [accel,gyro,mag] = read(imu);

   % code below aligns the axis to NED coordinates
   accel = [-accel(:,2), -accel(:,1), accel(:,3)];
   gyro = [gyro(:,2), gyro(:,1), -gyro(:,3)];
   mag = [mag(:,1)-magx_correction, mag(:, 2)-
   magy_correction, mag(:,3)-magz_correction];
end
```

The function above named [accel,gyro,mag] = readSensorDataMPU9250(imu) reads the imu sensor created using mpu9250 system object data from the 9 DOF sensor align the axis in accordance with NED coordinates and removes hard iron distortions. Hard iron distortion correction values in units of microtesla.

Finally run the code below.

```
GyroscopeNoiseMPU9250 = 3.0462e-06;
% GyroscopeNoise (variance value) in units of rad/s
AccelerometerNoiseMPU9250 = 0.0061;
% AccelerometerNoise(variance value)in units of m/s^2
viewer = HelperOrientationViewer('Title',{'AHRS Filter'});
FUSE = ahrsfilter('SampleRate',imu.SampleRate,
'GyroscopeNoise',GyroscopeNoiseMPU9250,'AccelerometerN
oise',AccelerometerNoiseMPU9250);
stopTimer = 500;

tic;
while(toc < stopTimer)
   [accel,gyro,mag] = readSensorDataMPU9250(imu);
   rotators = FUSE(accel,gyro,mag);
   for j = numel(rotators)
      viewer(rotators(j));
   end
end
```

The GyroscopeNoiseMPU9250 and AccelerometerNoiseMPU9250 value are given from the sensor's datasheet. The HelperOrientationViewer configures the animation display and the FUSE combine the IMU sample rate, gyroscope noise and accelerometer noise using the AHRS filter.

For MPU-9250, the magnetometer axis can be considered as device axis. To find the X- axis , you need to find the mark as shown below.

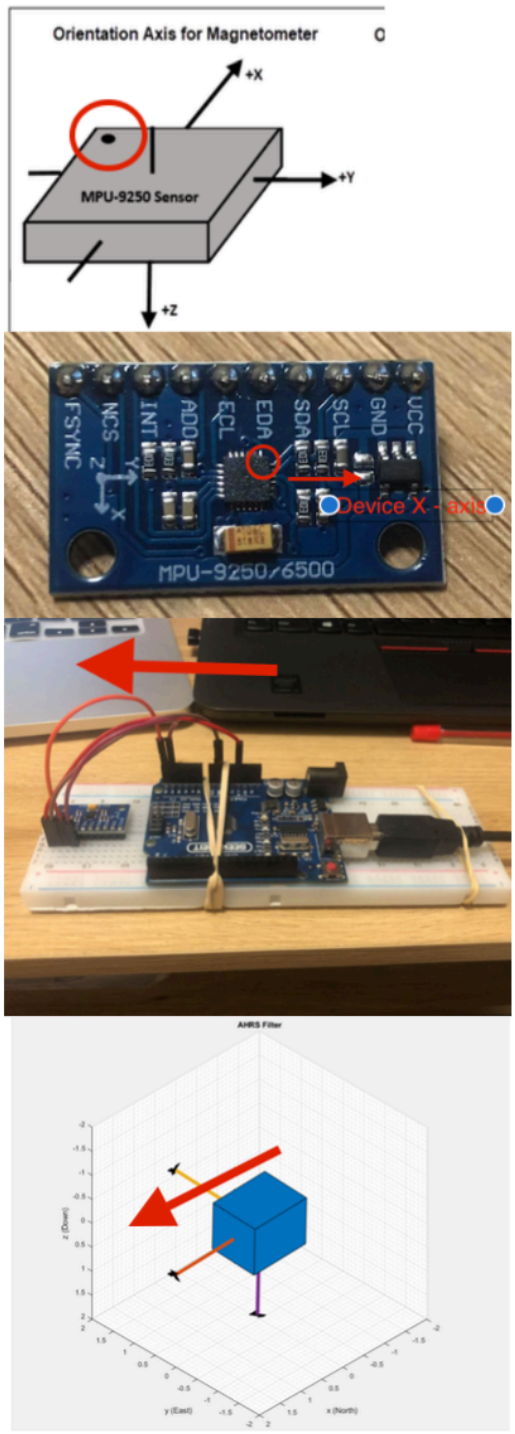

Figure 7: IMU Device Heading/X-axis Orientation.

This means that in our plot, the X- axis is our heading and tells us the direction of the orientation.

As shown in figure 7, when the device X axis of sensor is pointing to north, the device Y-axis is pointing to east and device Z-axis is pointing down.

Below is the actual output in MATLAB animation when the device (x-axis) is facing North shown in Figure 8 below.

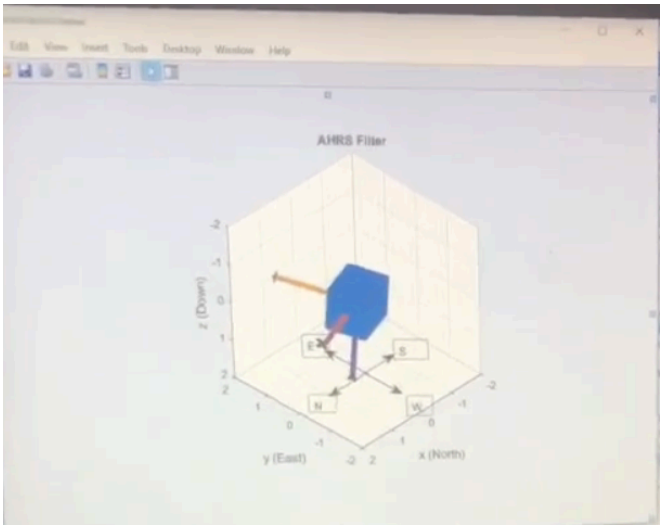

Figure 8: Actual Device orientation pointing to Magnetic North in MATLAB

Below is the actual output in MATLAB animation when the device (x-axis) is facing West shown in Figure 9 below.

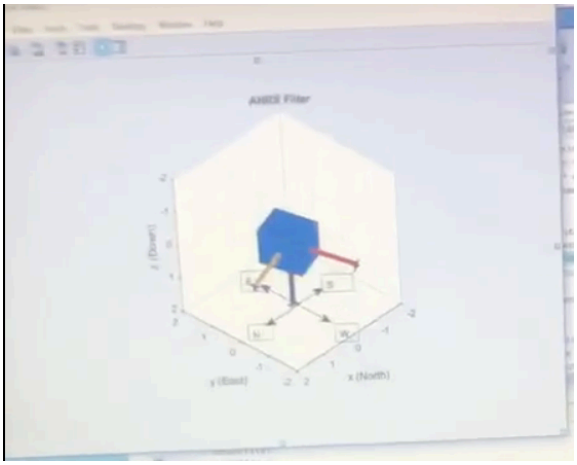

Figure 9: Actual Device orientation pointing Westh in MATLAB

IV. Wireless Soil Monitoring Sensor Node Project

In this project, the Capacitive Soil Moisture sensor is used and connect it to the WEMOS ESP32 Development board to create a wireless sensor node and then connect it to the cloud/internet using the Blynk App and read the data from the soil sensor in mobile phone. The capacitive soil moisture sensor is dip in the soil/water and depending how deep it is submerged, then determines the soil moisture whether its Dry, Wet or Very Wet according to the level of water or moisture.

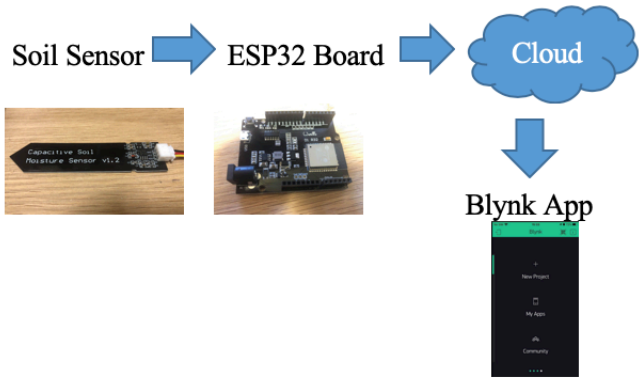

Figure 10: Block Diagram of the Soil Sensor Wireless Network

A. Hardware Setup

First, the soil sensor is connected to the ESP32 board by connecting the Vcc, Ground and Aout pins of the soil sensor to the 3.3V, Ground and IO39 pins of the ESP32 board respectively then connect it to the Laptop/PC using a USB cable. This set up is shown in the Figure 11below.

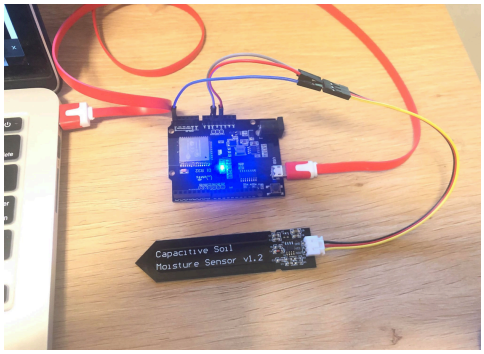

Figure 11: Actual setup of the Soil Sensor and ESP32 Board

Then, the BLynk App was then installed and configured in the mobile phone. A new project was created named SoilSensor and then created a device named SoilSensorBien, this device is what we setup in Figure 6. To make sure that the setup is connected to Blynk, an authentication token was automatically created and was sent to email. This token is used in the arduino sketch/code so that our soil sensor device can be seen or connected to the Blynk app. After that, an LCD widget was created to display the data from the soil sensor and set its input Pin to V2. This steps can be seen on below figure 12 below.

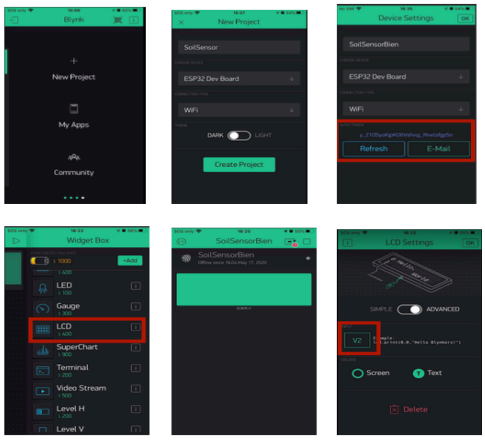

Figure 12: Setting up the Blynk App on mobile phone

Finally, the ESP32 board is coded in Arduino IDE so that it will receive data from the soil sensor and send it to the Blynk server and then to our mobile phone using the BLynk App.

B. Sketch/Code Used in Arduino IDE.

The sketch/code on the Soil Sensor starts by including the libraries needed for the ESP32 board and WiFI.

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
```

Then, the Authentication Token from blynk and WiFI credentials are setup. The Authenticatin Token from blynk is used and the home wifi was used.

```
char auth[] = "y_21OSyoKjpKOIlVshvqj_RrwtsfgpSn";
char ssid[] = "Optus X Bien";
char pass[] = "biengwapo";
```

Next, the method void setup is used to implement the Debug console, authentication token and wifi credentials.

```
void setup()
{ Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);}
```

After that, the **WidgetLCD** lcd from blynk and the variables for the sensor are declared. The **WidgetLCD** allows the use of the virtual LCD on the Blynk App in order to display the data. The string **data** is declared so that we can output a txt/message to our Blynk app LCD when we satisfy a certain condition. The constant **AirValue** is set to 4070. This value is the value during calibration when the soil sensor is not inserted to the soil/water. The constant **WaterValue** is set to 1975. This value is the value during calibration when the soil sensor is inserted on the maximum level of soil/water. The **intervals** variable is set to one third of the difference between **AirValue** and **WaterValue.**

```
WidgetLCD lcd(V2);
String data ="";
const int AirValue = 4070;
const int WaterValue = 1957;
int intervals = (AirValue - WaterValue)/3;
int soilMoistureValue = 0;
```

Then the main code that detects the soil moisture. By implementing the method void **level**, the variable **soilMoistureValue** is set to the analog pin 39 of the ESP32board. This pin is the output value of the soil sensor that reads the soil moisture data. If the **soilMoistureValue** is greater than the **WaterValue** and less than the sum of **WaterValue** and **intervals,** the serial monitor and the Blynk virtual LCD will display the data "**Very Wet".** If the **soilMoistureValue** is greater than the sum of **WaterValue** and **intervals**, and the **soilMoistureValue** is less than the difference of **AirVvalue** and **intervals**, the serial monitor and Blynk virtual LCD will display the data **"Wet"**. Finally, if the **soilMoistureValue** is less than the **AirValue**, and the **soilMoistureValue** is greater than the sum of **AirValue** and **intervals**, the serial monitor and the BLynk virtual LCD will display the data **"Dry".** Basically, the water level in the capacitane soil moisture sensor is divided into three parts, the lower part means its Dry, middle part is Wet and the upper part is Very Wet.

```
Serial.println("Very Wet");   //print text to Serial Monitor
lcd.clear();                  //Clear message on Blynk Virtual LCD
lcd.print(0,0, "Soil Moisture = "); //display text to 1st line of
Blynk Virtual LCD
data = ("Very Wet");          //set data text"Very Wet"
lcd.print(0,1, data);         //display data to 2nd line of Blynk
Virtual LCD
```

Finally, the blynk method is called to run Blynk on the mobile app.

```
void blynk()
{
  Blynk.run();
}
```

C. Actual Output

Below is the actual output from mobile Blynk App when the soil sensor is not dip in the water, dip 1/3 on the water and full dip in the water.

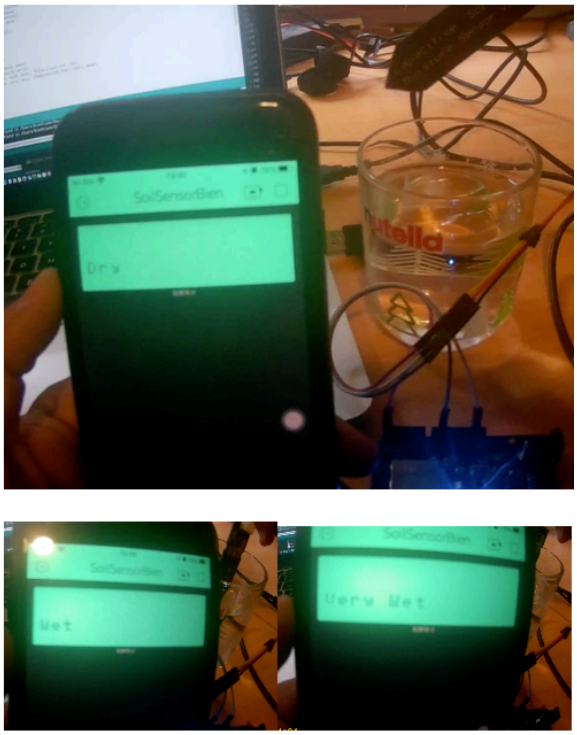

Figure 13: Actual Blynk app LCD output when soil sensor is not Dip or Out of water, it is dipped to 1/3 of its body and dipped full level of its body.

V. Soil Sensor Wireless Network Project

This wireless network connects our sensor node device to another device using the Blynk Bridge feature. The Blynk Bridge allows two or more devices to communicate with each other using the Blynk Platform wirelessly by using an encrypted Authentication Token the same as in the previous project. The aim of this project is to send the data from the soil sensor node to the other device and triggers a relay whenever a certain condition is satisfied. To have a visual representation, see the block diagram below on Figure 14 below.

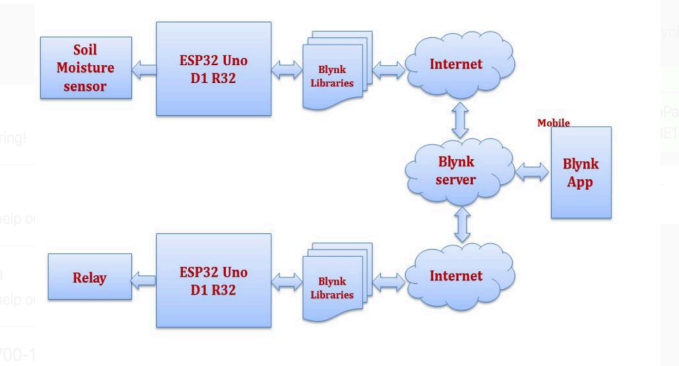

Figure 14: Block Diagram of the Wireless Sensor Network using BLynk Platform

A. Hardware and Software Setup

This time our sensor node is now the receiving node that is connected to a relay. The soil sensor node was from the other team member and uses his wireless soil sensor node. Still, Blynk platform is used to communicate between each sensor node to the internet then to the Blynk server and then is controlled and monitored by mobile phone using the Blynk App. The relay is connected to the Pin 2 of the ESP32 board. The actual setup is shown on the figure 15 below.

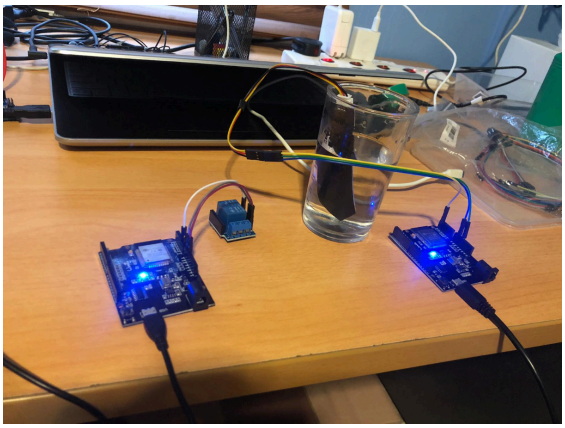

Figure 15: Actual setup of the Wireless Sensor Network with a soil sensor and a relay.

The output of the soil sensor is mapped in percentage, meaning a value of 0 means there is no moisture or the soil is dry. On the other hand, if the value is 100, it means the moisture is high and the soil is wet or the soil sensor is fully submerged to the water. In the Blynk App, A Widget level is used to indicate the water level from the sensor whether from low or full, a Widget Relay is also used to represent a virtual relay for the other sensor node that uses a relay. When the sensor value is less than 30, a notification will be pushed to your Blynk app saying that you need to add water to your soil, moreover, the relay from the other sensor node will be triggered and becomes ON, as well as the virtual relay in the Blynk app, if the sensor value is greater than 100, a notification saying “Moisture level is Full” is pushed on the Blynk app and the relay. See below figure 16.

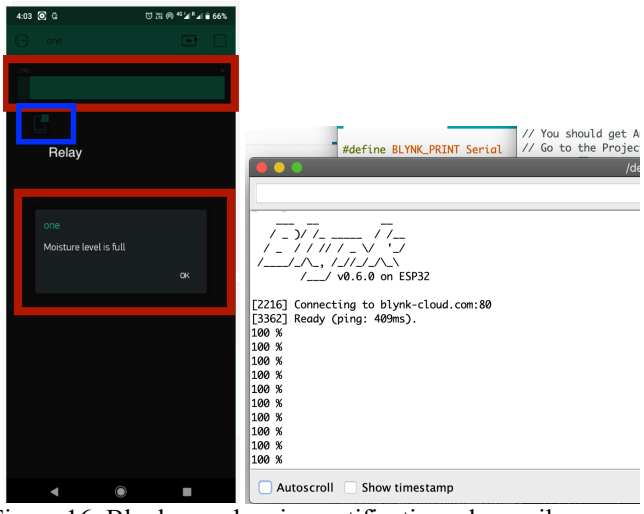

Figure 16: Blynk app showing notification when soil sensor output has a value 100 and Serial Monitor display.

B. Sketch/Code Used

The sketch/code used in the Soil Sensor Node for this project is almost the same as the previous project, where you need to set the Device Authentication Token and Wi-Fi Credentials. The only difference is the additional Bridge Authentication Token that allows the two devices to communicate with each other wirelessly using the Blynk Platform and an separate sketch/code for the device that triggers the relay

B.1. Soil Sensor Code

First the Widget Bridge “bridge1” from Blynk is declared. Then the BLYNK_CONNECTED function is implemented in the second device Authentication Token. This Token can also be found in the Blynk app and can be sent to your email.

```
WidgetBridge bridge1(V1);

BLYNK_CONNECTED() {
bridge1.setAuthToken("60zoaQJI2c4T8WeGUC383uMuVcd7FGZ5");
}
```

Moreover, the method use in the void level() is different than what is used in the previous project. First the variables and constants are declared.

```
const int AirValue = 3900;
const int WaterValue = 2100
int soilMoistureValue = 0;
int soilmoisturepercent=0;
```

Then variables are initialized where soilMoistureValue will read the soil sensor output from PIN39 of the ESP32 board. Then the soilmoisturepercent is map from 0 to 100 depending on the soilMoistureValue,AirValue and WaterValue. If the value of the soilmoisturepercent is less than 0, a text of “0%” will be print to the serial monitor, a notification will be pushed to the Blynk app saying “ Need to be water”, a signal or data is sent to the other sensor node with relay to make the pin 2 High or turn it ON, and lastly the soil sensor data will be sent to the virtual level indicator on the BLynk app.

```
Serial.println("0 %");
Blynk.notify("Need to be water");
bridge1.digitalWrite(2, HIGH);
bridge1.virtualWrite(V5, 1000);
```

Likewise, when the soilmoisture percent is more than 100, serial monitor will print “100%”, notification is sent to Blynk App saying “Moisture Level is full”, send data to sensor node with relay and turn it Off and data will be sent to virtual level indicator on the Blynk app.

```
Serial.println("100 %");
Blynk.notify("Moisture level is full");
bridge1.digitalWrite(2, LOW);
bridge1.virtualWrite(V5, 0);
```

B.2. Relay Device Code.

The code for the relay device starts with including the needed libraries for ESP32 board. Then sets the Authentication Token and the Wi-Fi Credentials to connect the Relay Device to the Blynk cloud/platform. Then the code update the virtual port 5 of the BLynk app which , which is triggered by the soil sensor code bridge1.digitalWrite(2, HIGH OR LOW); Then turn the relay that is connected in PIN 2.

```
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

char auth[] = "60zoaQJI2c4T8WeGUC383uMuVcd7FGZ5";
char ssid[] = "Optus X Bien";
char pass[] = "biengwapo";

// This code will update the virtual port 5
BLYNK_WRITE(V5) {
  int pinData = param.asInt();
}
void setup(){
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
}

void loop(){
  Blynk.run();
}
```

C. Actual Output.

Below is the actual output of the project when all the devices are connected and the sketch mentioned above was run in the Arduino IDE.

Figure 16: Actual Output of the Soil Sensor Wireless Network when sensor is not dipped or Out of Water.

As shown in the figure 16 above, when the soil sensor is not dipped or out of the water, a notification in be sent to say “Need to be water”, the virtual level indicator in the Blynk app is empty, the relay sensor device led turns ON and the relay also turns ON.

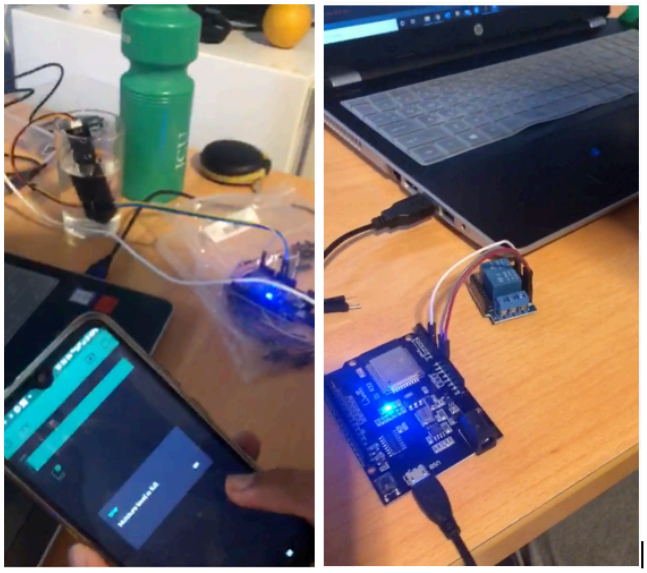

Figure 17: Actual Output of the Soil Sensor Wireless Network when sensor is not dipped or Out of Water.

As shown in the figure 17 above, when the soil sensor is dipped in the water, the notification in the Blynk app will push “Moisture level is full”, the virtual level indicator is full, the led in the sensor and relay are OFF.

VI. Appendix

A. IMU Orientation in MATLAB

The codes use for this project are taken from The Mathworks Inc [3].

The code below gets the corrected value of x,y and z axis of the Magnetometer

```
a = arduino('COM3', 'Uno', 'Libraries', 'I2C');
fs = 100; % Sample Rate in Hz
imu = mpu9250(a,'SampleRate',fs,'OutputFormat','matrix');

viewer = HelperOrientationViewer;

tic;
stopTimer = 50;
magReadings=[];
while(toc<stopTimer)
    % Rotate the sensor around x axis from 0 to 360 degree.
    % Take 2-3 rotations to improve accuracy.
    % For other axes, rotate around that axis.
    [accel,gyro,mag] = read(imu);
    magReadings = [magReadings;mag];
```

```
end

magx_min = min(magReadings(:,1));
magx_max = max(magReadings(:,1));
magx_correction = (magx_max+magx_min)/2;

magy_min = min(magReadings(:,2));
magy_max = max(magReadings(:,2));
magy_correction = (magy_max+magy_min)/2;

magz_min = min(magReadings(:,3));
magz_max = max(magReadings(:,3));
magz_correction = (magz_max+magz_min)/2;
```

The code below displays the orientation in MATLAB using the corrected value from above code.

```
% GyroscopeNoise and AccelerometerNoise is determined
from datasheet.
GyroscopeNoiseMPU9250 = 3.0462e-06; % GyroscopeNoise
(variance value) in units of rad/s
AccelerometerNoiseMPU9250 = 0.0061; %
AccelerometerNoise(variance value)in units of m/s^2
viewer = HelperOrientationViewer('Title',{'AHRS Filter'});
FUSE = ahrsfilter('SampleRate',imu.SampleRate,
'GyroscopeNoise',GyroscopeNoiseMPU9250,'AccelerometerN
oise',AccelerometerNoiseMPU9250);
stopTimer = 500;

tic;
while(toc < stopTimer)
   [accel,gyro,mag] = readSensorDataMPU9250(imu);
   rotators = FUSE(accel,gyro,mag);
   for j = numel(rotators)
      viewer(rotators(j));
   end
end


function [accel,gyro,mag] = readSensorDataMPU9250(imu)
% This function reads the imu sensor created using mpu9250
system object data from the 9 DOF sensor
% align the axis in accordance with NED coordinates and
removes hard iron distortions
% Hard iron distortion correction values in units of microtesla
magx_correction = 35.039062500000000;
magy_correction = 14.414062500000000;
magz_correction = -22.691015625000000;
[accel,gyro,mag] = read(imu);
% Align coordinates in accordance with NED convention
accel = [-accel(:,2), -accel(:,1), accel(:,3)];
gyro = [gyro(:,2), gyro(:,1), -gyro(:,3)];
mag = [mag(:,1)-magx_correction, mag(:, 2)-
magy_correction, mag(:,3)-magz_correction];
end
```

B. Wireless Soil Monitoring Sensor Node

The code below used to calibrate the AirValue and WaterValue.

```
void setup() {
  Serial.begin(9600); // open serial port, set the baud rate as
9600 bps
}
void loop() {
  int val;
  val = analogRead(39); //connect sensor to Analog 0
  Serial.println(val); //print the value to serial port
  delay(100);
}
```

The Code below gets the data from Soil Sensor and display the data in the Blynk App.

```
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>


// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "y_21OSyoKjpKOIlVshvqj_RrwtsfgpSn";
//SoilSensorBien

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "Optus X Bien";
char pass[] = "biengwapo";

void setup()
{
  // Debug console
  Serial.begin(9600);

  Blynk.begin(auth, ssid, pass);
  // You can also specify server:
  //Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 80);
  //Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100),
8080);
}

WidgetLCD lcd(V2);
String data ="";

const int AirValue = 4070;   //you need to replace this value
with Value_1
const int WaterValue = 1957;  //you need to replace this value
with Value_2
int intervals = (AirValue - WaterValue)/3;
int soilMoistureValue = 0;
```

```
void level() {
  Serial.begin(9600); // open serial port, set the baud rate to
9600 bps
}
void loop() {
soilMoistureValue = analogRead(39);  //put Sensor insert into
soil
if(soilMoistureValue > WaterValue && soilMoistureValue <
(WaterValue + intervals))
{
  Serial.println("Very Wet");   //print text to Serial Monitor
  lcd.clear();                 //Clear message on Blynk Virtual LCD
  lcd.print(0,0, "Soil Moisture = "); //display text to 1st line of
Blynk Virtual LCD
  data = ("Very Wet");          //set data text"Very Wet"
  lcd.print(0,1, data);         //display data to 2nd line of Blynk
Virtual LCD

}
else if(soilMoistureValue > (WaterValue + intervals) &&
soilMoistureValue < (AirValue - intervals))
{
  Serial.println("Wet");
    lcd.clear();
    data = ("Wet");
    lcd.print(0,0, "Soil Moisture = ");
    lcd.print(0,1, data);
}
else if(soilMoistureValue < AirValue && soilMoistureValue
> (AirValue - intervals))
{
    Serial.println("Dry");
    lcd.clear();
    data = ("Dry");
    lcd.print(0,0, "Soil Moisture = ");
    lcd.print(0,1, data);

}
delay(100);
}

void blynk()
{
  Blynk.run();
}
```

C. Soil Sensor Wireless Network

The code is used by the Soil Sensor node.

```
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
BlynkTimer timer;

const int AirValue = 3900;   //you need to replace this value
with Value_1
const int WaterValue = 2100;  //you need to replace this value
with Value_2
int soilMoistureValue = 0;
int soilmoisturepercent=0;

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "otFZATECx4jswZbvvTMnHjwYKrVSC6XV";

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "SANKAR";
char pass[] = "sankar331";

WidgetBridge bridge1(V1);

BLYNK_CONNECTED() {
  // Place the AuthToken of the second hardware here

bridge1.setAuthToken("60zoaQJI2c4T8WeGUC383uMuVcd7
FGZ5");
}
void level()
{
   soilMoistureValue = analogRead(39);  //put Sensor insert
into soil
   //Serial.println(soilMoistureValue);
   soilmoisturepercent = map(soilMoistureValue, AirValue,
WaterValue, 0, 100);
//    Blynk.virtualWrite(V5, soilmoisturepercent);
//    Blynk.notify("")

if(soilmoisturepercent < 0)
   {
    Serial.println("0 %");
    Blynk.notify("Need to be water");
    bridge1.digitalWrite(2, HIGH);
    bridge1.virtualWrite(V5, 1000);
   }
   else if(soilmoisturepercent > 100)
   {
    Serial.println("100 %");
    Blynk.notify("Moisture level is full");
    bridge1.digitalWrite(2, LOW);
    bridge1.virtualWrite(V5, 0);
   }
   else if(soilmoisturepercent >0 && soilmoisturepercent <
100)
   {

    Blynk.virtualWrite(V5, soilmoisturepercent);
    Serial.print(soilmoisturepercent);
    Serial.println("%");}
}

void setup(){
```

```
  // Debug console
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
  pinMode(39, INPUT);
  timer.setInterval(1000L, level);
}
void loop(){
  Blynk.run();
  timer.run();
}
```

The code below is used in the Relay Device node

```
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "60zoaQJI2c4T8WeGUC383uMuVcd7FGZ5";

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "SANKAR";
char pass[] = "sankar331";

// This code will update the virtual port 5
BLYNK_WRITE(V5) {
  int pinData = param.asInt();
}

void setup(){
  Serial.begin(9600);
  Blynk.begin(auth, ssid, pass);
}

void loop(){
  Blynk.run();
}
```

VII. REFERENCES

[1] Modtronix, “Overview”.[Online]. Available: https://wiki.modtronix.com/doku.php?id=tutorials:wireless:esp:esp32:esp32_additional_information
[2] df[Online]. Available: https://www.arduino.cc/en/Tutorial/AnalogInOutSerial
[3] InvenSense. “MPU-9250 Product Specification Revision 1.1”. Online PDF: https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
[4] The Mathworks Inc. (2020). Estimating Orientation Using Interial Sensor Fusion and MPU-9250”. Retrieved from: https://au.mathworks.com/help/fusion/examples/Estimating-Orientation-Using-Inertial-Sensor-Fusion-and-MPU-9250.html