DEPARTMENT OF PHYSICS
UNIVERSITY OF COLOMBO

PH3039-DATA ACQUISITION LABORATORY

2022

WEATHER STATION

Name: D.H.C.I. Dharmarathne

Index: 14885

Date: 30 September 2022

ABSTRACT

In everyday life, people need to know the weather. weather information is more useful for agriculture, various outdoor projects, and planning daily activities. Sometimes, the source which provides weather information can be incorrect. Because some websites only provide weather information for a large area or they may have a single weather data collection center for a large area. Therefore, if people can make a small weather station for their area, they can get more accurate weather information using it. Therefore, this project discusses how to make a small weather station, get data from it with sensors, how to upload, collect and save data from WiFi to Google Sheets, how present information using a user-friendly interface and keep the circuit in sleep mode in unnecessary times. The weather station is made based on a Micro Computer unit and the sensors connected to it collect the weather data. The circuit was made in this way, special support was created and attached to it, and a battery was also connected to provide power and the circuit was powered up. Then a Wi-Fi connection is provided and the data obtained by the sensors in that circuit is uploaded to Google Sheet and ThingSpeak using the WiFi module in the micro computer unit. Furthermore, in cases where the Wi-Fi connection is lost, the micro computer unit is set up so that the data obtained by the sensors can be temporarily saved in the flash memory of the micro computer unit. After the Wi-Fi connection is restored, the data will be uploaded to Google Sheets and ThingsSpeak. Then the data is analyzed using Google Sheets and ThingSpeak and information is presented to the users. Ultimately, a low-cost, energy-efficient mini-weather station was built to collect and analyze data. After that, this system was connected with ten other systems and data was analyzed by Google Sheets and Grafana, the areas belonging to each center were mapped and a system was built to provide weather information for those areas.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

One of the things we probably do every morning is look out the window and see what the weather is like. Then look at the outside and will do the planning for the day's work using the weather situation of the surrounding. Then it will help to plan what kind of clothes we must wear and we can identify where must we go and what kind of work we must do within that day. So, if that day will see like a rainy day, then we can plan to do work inside. Otherwise, it will be like a sunny day, and then we can go outside and do our work. But there is a problem. There are some days, the morning time like a sunny day, but when the time goes to up, and then the day is like fully rainy day. Due to this reason, we can't plan a day as a nice day. Also weather information is needed for agriculture. Various methods have been used in the past. Among them, they gained an understanding of the rainfall and the direction of the wind through the level and direction of nest building of woodpeckers. But nowadays with technological advancement we can use weather data for that. Therefore we have to use the weather forecast.

Weather forecast is a statement saying what the weather will be like next day or for next few days. A weather forecast is a statement saying what the weather will be like the next day or for the next few days. It is using the analysis of temperature, wind speed, wind direction, humidity, air pressure and rain of the previous day and then gives the weather forecast for the next days. In this project, a large weather station was made using several of single mini weather stations and then the data was collected using mini weather stations. The *DHT 11* temperature sensor was gave the data of temperature and humidity, the *Rain sensor* was gave the data of rain, the *BMP180* sensor was gave the data of air pressure, the *LDR* sensor was gave the data of sun light and the *ESP 32* was used and sent the data to a google sheet. The data of several single mini weather stations send to the google sheet like this. Then the data analyst can analyse the data and give a statement about the weather forecast. Finally, the "ThingSpeak" was used for analysing the forecast data.

1.1 DHT-11 Temperature & Humidity Sensor

The *DHT-11* is a basic, ultra-low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins are needed). It's fairly simple to use but requires careful timing to grab data. The new data can get from it once every 2 seconds. *DHT-11* sensor provides humidity value in percentage in relative humidity (20 to 90% RH) and temperature values in degrees Celsius (0 to 50 °C) DHT11 sensor uses resistive humidity measurement component, and *NTC* temperature measurement component. (Electronics, 2022)

1.2 The Rain Drop Sensor

A sensor that is used to notice water drops or rainfall is known as a rain sensor. This kind of sensor works like a switch. This sensor includes two parts like sensing pad and a sensor module. Whenever rain falls on the surface of a sensing pad then the sensor

module reads the data from the sensor pad to process and convert it into an analog or digital output. So the output generated by this sensor is analog (AO) and digital (DO). The operating voltage range from 3.3 V to 5V. The operating current is 15 mA. There is a nickel plate of one side in sensing pad. The comparator chip is LM393. In using this sensor module, the user can get data about rainfall.

1.3 LDR Sensor Module

LDR sensor is used to detect the intensity of light. LDR's meaning is "Light Dependant Resistors". When there is light, the resistance of LDR will become low according to the intensity of light. The greater the intensity of light, the lower the resistance of LDR. LDRs are light-sensitive devices most often used to indicate the presence or absence of light or to measure light intensity. We can use the light-dependent resistor circuit for controlling the loads based on the intensity of light. An LDR or a photoresistor is a device that is made up of high-resistance semiconductor material. In using this sensor module, the user can get data about light intensity of sun.

1.4 BMP180 Barometric Pressure Sensor Module

Barometric pressure (also known as atmospheric pressure), is the pressure caused by the weight of air pressing down on the Earth. BMP180 is a high-precision sensor that can give pressure as a digital output. Barometric Pressure is the weight of air applicable to everything. The air has weight and wherever there is air its pressure is felt. BMP180 sensor senses that pressure and provides that information in digital output. BMP180 can measure barometric pressure from 300 to 1100 hPa (9000m to -500m above sea level), and temperature from -40°C to 85°C with ±1.0°C accuracy. The pressure measurements are so precise (low altitude noise of 0.25m), that users can even use it as an altimeter with ±1 meter accuracy. The module comes with an onboard LM6206 3.3 V regulator, so it can use with a 3.3 V logic microcontroller like ESP 32 without worry. When using this sensor module, the user can get data about air pressure in the surroundings. (lastminuteengineers.com, 2022)

1.5 ESP32 Micro-computer unit

The ESP32 is a very versatile *System On a Chip* (*SoC*) that can be used as a general-purpose microcontroller with quite an extensive set of peripherals including *WiFi* and Bluetooth wireless capabilities. It is manufactured by Shanghai-based *Espressif Systems* and it is less cost micro-computer unit. ESP32 is a development module. This is basically an ESP32 module mounted on a board with additional support circuitry such as a voltage regulator and a serial to *USB IC*. It allows direct connection to a desktop PC that can then be used to compile, download, and run programs directly on this module. The compiler is for **C/C++** which is officially supported. The Arduino Ide can upload the code to the ESP32. For this weather station, the ESP32 which has Wi-Fi module was used. The data which were collected from sensors can send to the google sheet through Wi-Fi connection.

1.6 Google Sheet

Google Sheets is a spreadsheet program included as part of the free, web-based Google Docs Editors suite offered by Google. The Google product offers typical spreadsheet features, such as the ability to add, delete, and sort rows and columns. But unlike other spreadsheet programs, Google Sheets also allows multiple geographically dispersed users to collaborate on a spreadsheet simultaneously and communicate through the built-in instant messaging program. Users can upload spreadsheets directly from their computers or mobile devices. The app automatically saves all changes and users can see other users' changes as they are made. The Google Sheets online spreadsheet application enables users to *create*, *edit* and *format* spreadsheets online to organize and analyse information. Google Sheets is often compared to Microsoft Excel, as both applications are used for similar purposes. Google Sheets is essentially Google's cloud-based version of Microsoft Excel's basic features.

1.7 Arduino IDE

The Arduino IDE (Integrated Development Environment) is a program used to write code. The Arduino IDE is open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as Windows, Mac OS X, and Linux. It supports the programming languages C and C++. The program or code written in the Arduino IDE is often called as sketching. We need to connect the ESP32 board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.*ino*'. When the code is finished writing, the Arduino IDE can show the bugs which the code has. Then the user can check all errors and then it can upload to the ESP32 board. When uploading the code into ESP32, if there is any error it, it will show the error by Arduino IDE.

1.8 ThingSpeak platform

ThingSpeak is an IoT analytics platform service that allows to aggregate, visualize, and analyze live data streams in the cloud. Features of ThingSpeak include real-time data collection, data processing, visualizations, apps, and plugins. At the heart of ThingSpeak is a ThingSpeak Channel. A channel is where users send users data to be stored. Each channel includes 8 fields for any type of data, 3 location fields, and 1 status field. The ThingSpeak IoT platform enables clients to update and receive updates from channel feeds via the ThingSpeak MQTT broker. MQTT is a publish/subscribe communication protocol that uses TCP/IP sockets or WebSockets.

2 THEORY

2.1 DHT-11 Humidity & Temperature Sensor

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 3-pin single row pin package. (Electronics, 2022)

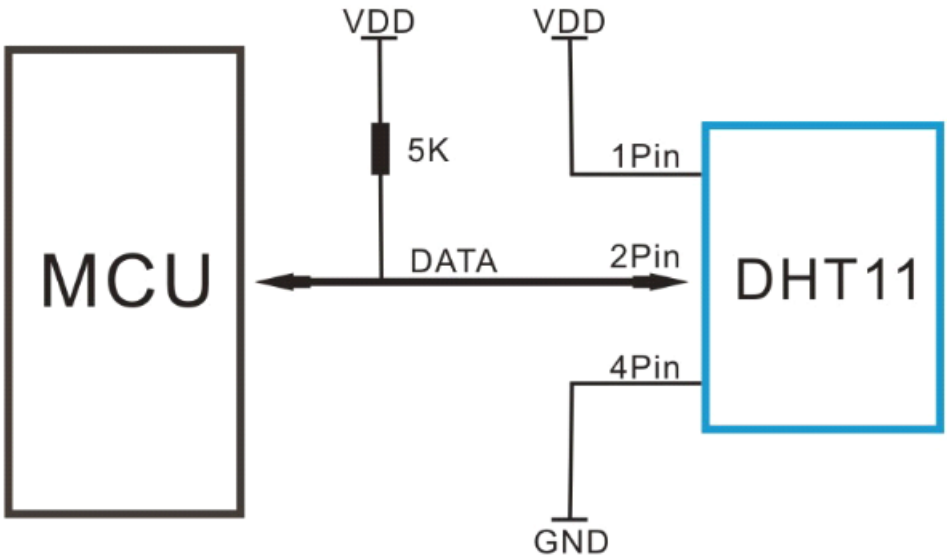


Figure 2-1: Figure of typical application for MCU with DHT-11 sensor
(Electronics, 2022)

According the figure 2-1, when the connection cable is shorter than the 20 meters, a 5 kΩ pull-up resistor is recommended before use it. The DHT-11's power supply is 3.5 V to 5.5 V DC current. DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture-holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process these changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with an increase in temperature. To get a larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

2.1.1 DHT-11 Sensor pin configure

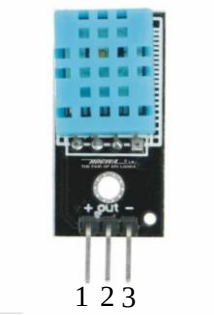


Figure 2-2: Figure of pin configuration of DHT-11 Sensor

Pin number 1 is the VCC pin and it needs to supply from 3.5 V to 5.5 V voltages for working as properly. Pin number 2 is the data pin and it gives the output as digital. DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to ESP32. Pin number 3 is the ground pin. In the blue box with thin holes, the middle of it has a capacitive humidity sensor and a thermistor to measure the surrounding air. (Electronics, 2022)

2.2 Rain Sensor module

This sensor includes two parts like sensing pad and a sensor module. Whenever rain falls on the surface of a sensing pad then the sensor module reads the data from the sensor pad to process and convert it into an analog or digital output. The sensing pad includes a set of uncovered copper traces which mutually work like a variable resistor or a potentiometer. Here, the sensing pad resistance will be changed based on the amount of water falling on

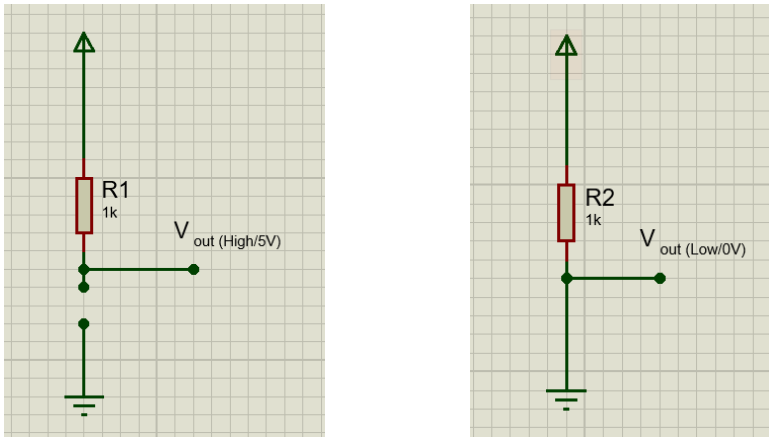


Figure 2-3:(a) Figure of dry situation of Rain sensor, (b) Figure of wet situation of Rain sensor

According to figure 2-3, the sensing pad resistance will be changed based on the amount of water falling on its surface. Therefore, the resistance is inversely related to the amount of water. When the water on the sensing pad is more, the conductivity is better & gives less resistance (case of Figure 2-3:(b)). Similarly, when the water on the surface pad is less, the conductivity is poor & gives high resistance (case of Figure 2-3:(a)). So the output of this sensor mainly depends on the resistance.

2.2.1 Rain Drop sensor pin configuration with circuit diagram

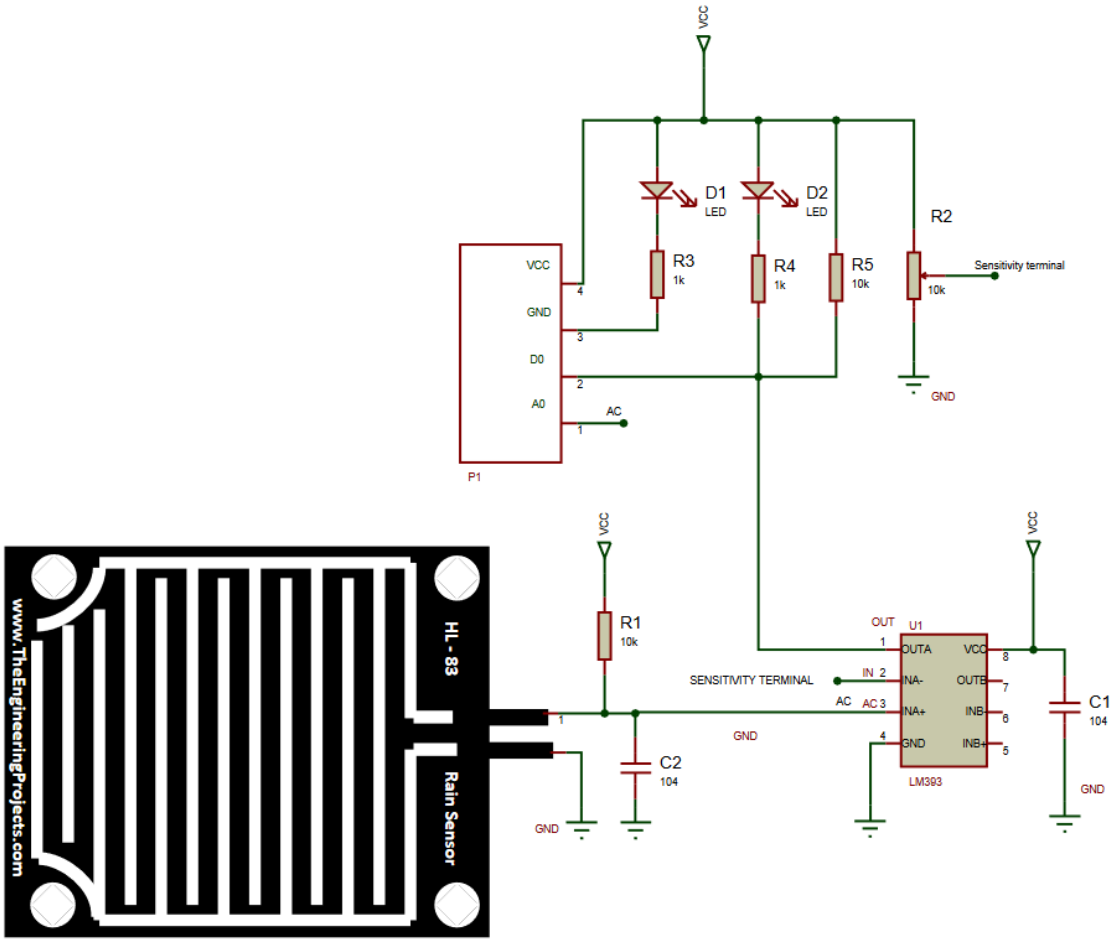


Figure 2-4: Figure of circuit diagram of Rain sensor (Source: www.components101.com)

The control module of the raindrop sensor has 4 outputs. VCC is connected to a 5V supply. The GND pin of the module is connected to the ground. The D0 pin is connected to the digital pin of the microcontroller for digital output or the analog pin can be used. To use the analog output, the A0 pin can be connected to the ADC pin of a microcontroller. The sensor module consists of a potentiometer, LN393 comparator, LEDs, capacitors and resistors. The pinout figure (Figure 2-4) shows the components of the control module. The Rain board module consists of copper tracks, which act as a variable resistor. Its resistance varies with respect to the wetness on the Rain board. (components101, 2019)

2.3 LDR Sensor Module

2.3.1 LDR Sensor calibration

LDR sensor module is used to detect the intensity of light. When there is light, the resistance of LDR will become low according to the intensity of light. The greater the intensity of light, the lower the resistance of LDR. The resistance of an LDR may typically have 5000Ω resistances in daylight and 20000000Ω resistance in dark.

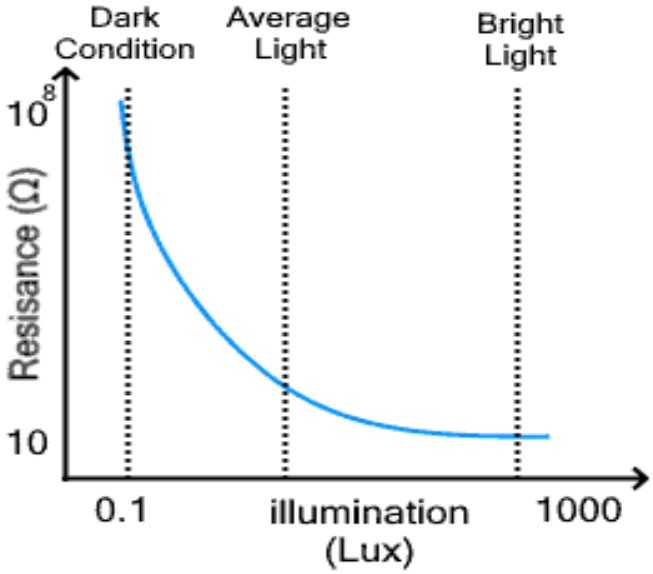


Figure 2-5: Figure of LDR resistance vs light intensity graph (Source: www.kitronik.co.uk)

There is a large variation between the resistance of daylight and the resistance of dark. After plotting this variation and other various times and resistance on a graph, the graph was got as above (figure 2-5).

2.3.2 LDR Sensor diode pin configuration

Figure 2-6: Figure of LDR Sensor diode (Source: [www.kitronik.co.uk)](www.kitronik.co.uk)

The working principle of an LDR is photoconductivity, an optical phenomenon. When the light is absorbed by the material then the conductivity of the material enhances. When the light falls on the LDR, then the electrons in the valence band of the material are eager to the conduction band. Hence, when light has ample energy, more electrons are excited to the conduction band which grades in a large number of charge carriers. When the effect of this process and the flow of the current starts flowing more, the resistance of the device decreases. Therefore, the users can measure that resistance using the voltage divider method and calculate the light intensity value according to resistance. (Electronics, 2022)

2.4 BMP180 Barometric Pressure Sensor Module

The BMP180 is a **piezo resistive** sensor that detects pressure. **Piezo resistive** sensors are made up of a semiconducting material (usually silicon) that changes resistance when a mechanical force like atmospheric pressure is applied. The BMP180 measures both pressure and temperature because temperature changes the density of gasses like air. At higher temperatures, the air is not as dense and heavy, so it applies less pressure on the sensor. At lower temperatures, the air is denser and weighs more, so it exerts more pressure on the sensor. The sensor uses real-time temperature measurements to compensate the pressure readings for changes in air density. The BMP180 outputs an uncompensated temperature (UT) value and an uncompensated pressure (UP) value. The temperature measurement is taken first and then the pressure measurement gave.

2.4.1 BMP180 Barometric Pressure Sensor module pin configuration

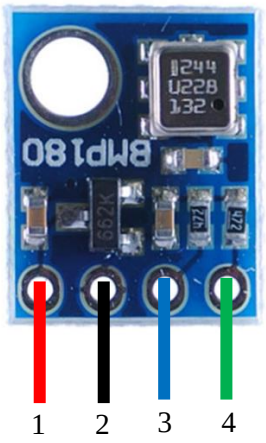


Figure 2-7: Figure of pin configuration of BMP180 Sensor Module

Pin number 1 is the VCC pin and it needs to supply from 3.3 V voltages for working as properly. Pin number 2 is the ground pin. Pin number 3 is the serial clock pin and pin number 4 is the serial data pin and it gives the output as digital. (lastminuteengineers.com, 2022)

2.5 ESP32 Micro-computer unit pin configure

2.5.1 GPIO pin configure of ESP32

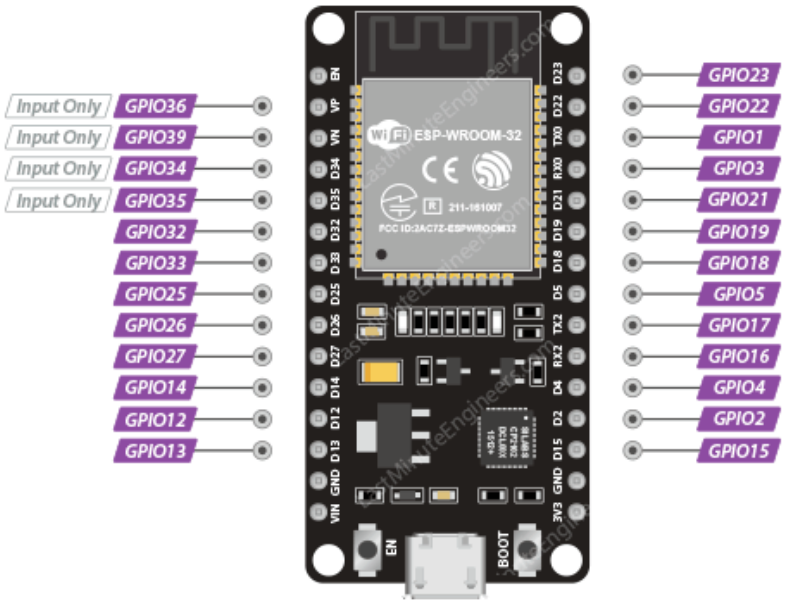


Figure 2-8: Figure of ESP32 MCU pin configuration (Source: www.lastminuteengineers.com)

According to figure 2-4, ESP32 development board has 25 GPIO pins which can be assigned to various functions programmatically. Each digital-enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. Because the ESP32 has many pins with specific functions, they may not be suitable for some projects. The following table shows which pins are safe to use and which pins require more attention before using them.

Table 2-1: Table of GPIO pins which can use more safely *(Source: www.lastminuteengineers.com)*

Label	GPIO	Safe to use	Reason
D0	0	Non	must be HIGH during boot and LOW for programming
TX0	1	No	Tx pin, used for flashing and debugging
D2	2	Non	must be LOW during boot and also connected to the on-board LED
RX0	3	No	Rx pin, used for flashing and debugging
D4	4	Yes	
D5	5	Non	must be HIGH during boot
D6	6	No	Connected to Flash memory
D7	7	No	Connected to Flash memory
D8	8	No	Connected to Flash memory
D9	9	No	Connected to Flash memory
D10	10	No	Connected to Flash memory

D11	11	No	Connected to Flash memory
D12	12	Non	must be LOW during boot
D13	13	Yes	
D14	14	Yes	
D15	15	Non	must be HIGH during boot, prevents startup log if pulled LOW
RX2	16	Yes	
TX2	17	Yes	
D18	18	Yes	
D19	19	Yes	
D21	21	Yes	
D22	22	Yes	
D23	23	Yes	
D25	25	Yes	
D26	26	Yes	
D27	27	Yes	
D32	32	Yes	
D33	33	Yes	
D34	34	Non	Input only GPIO, cannot be configured as output
D35	35	Non	Input only GPIO, cannot be configured as output
VP	36	Non	Input only GPIO, cannot be configured as output
VN	39	Non	Input only GPIO, cannot be configured as output

According to Table2-1, pins GPIO34, GPIO35, GPIO36(VP) and GPIO39(VN) cannot be configured as outputs, they can be used as either digital inputs, analog inputs, or for other unique purposes. Also, they do not have internal pull-up or pull-down resistors, like the other GPIO pins. And also pins GPIO36(VP) and GPIO39(VN) are an integral part of the ultra-low-noise pre-amplifier for the ADC, which help to configure the sampling time and noise of the pre-amp.

2.5.2 ADC pin configure of ESP32

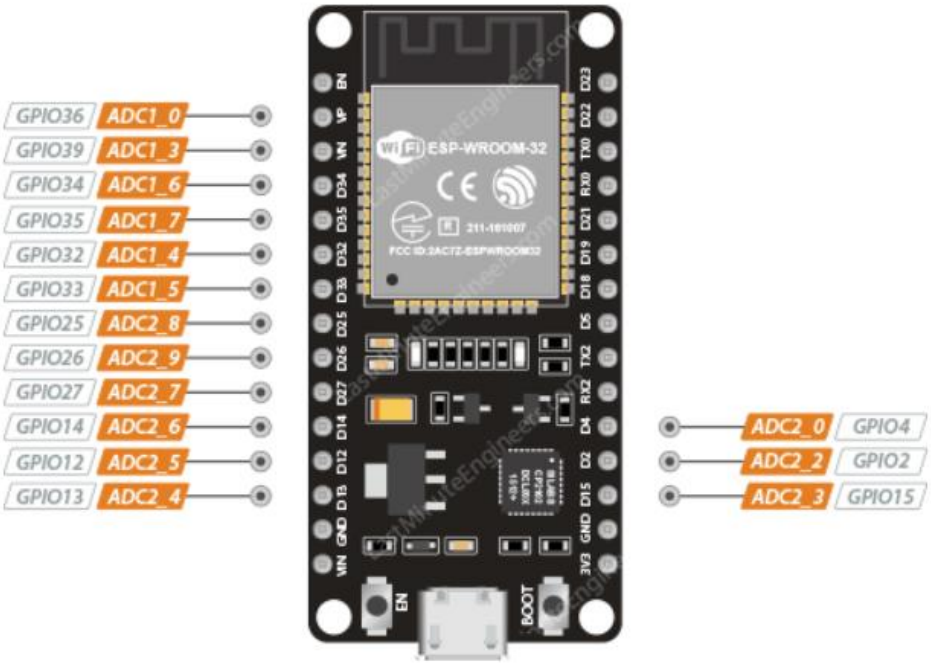


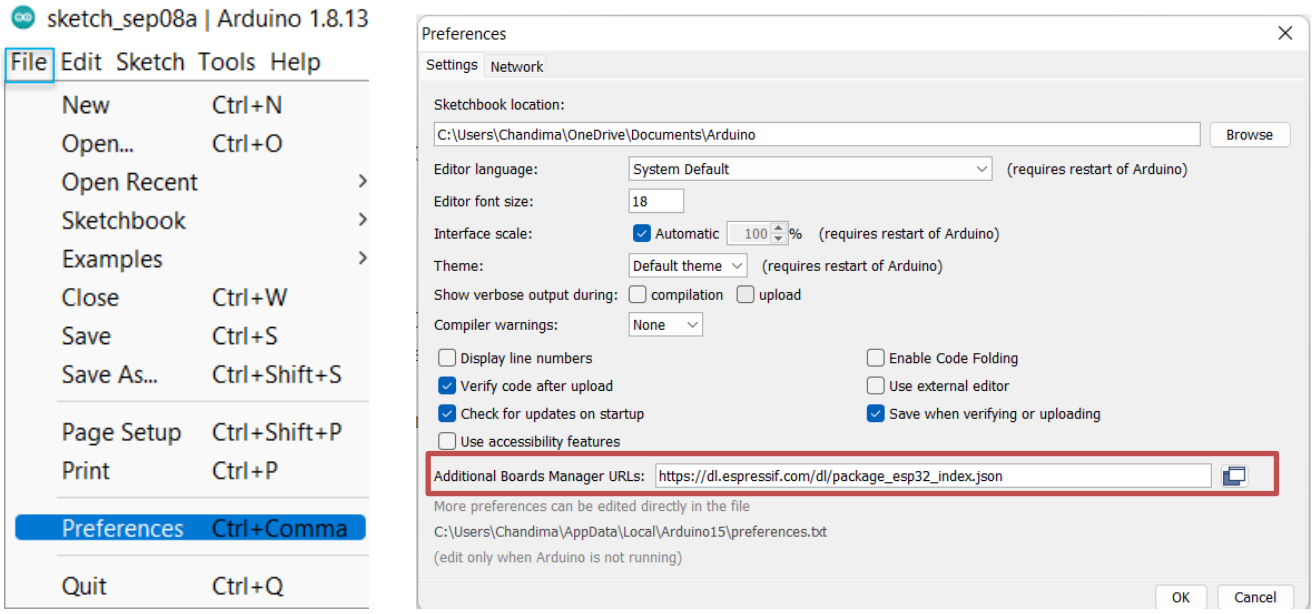
Figure 2-9: Figure of ADC pin configuration of ESP32 (Source: www.lastminuteengineers.com)

According the figure 2-4, the ADC on the ESP32 is a 12-bit ADC meaning it has the ability to detect 4096 (212) discrete analog levels. It will map input voltages between 0 and the operating voltage 3.3V into integer values between 0 and 4095. And also the ESP32 features two 8-bit DAC channels that can be used to convert digital signals into true analog voltages. Those are *GPIO25* and *GPIO26*. It can be used as a "digital potentiometer" to control analog devices. (lastminuteengineers.com, 2022)

3 METHODOLOGY

3.1 Setting up the Arduino IDE

In this part, the Arduino IDE was used. First, the Arduino IDE was opened and went to "File → Preference". Then the "Preference" dialogue box was opened. Then the "Additional Board Manager URLs was cleared and pasted the https://dl.espressif.com/dl/package_esp32_index.json URL in the setting part. The "OK" button was clicked after adding the above URL to "Additional Board Manager URLs".



(a) (b)

Figure 3-1:(a) Figure of path must follow to open the Preference dialogue box, (b) Figure of Preference dialogue box

In *Figure 3-1:(a)* was shown the path must follow to open the "Preference" dialogue box. The shortcut key is "Ctrl + Comma". Then *Figure 3-1:(b)* is the "Preference" dialogue box after opening following the *Figure 3-1:(a)* path. The URL was added into the typing space which has in the red box in *Figure 3-1:(b).*

Secondly, the "Boards Manager" dialogue box was opened using "Tools → Board → Boards Manager" path. Then the "Boards Manager" dialogue box was opened.

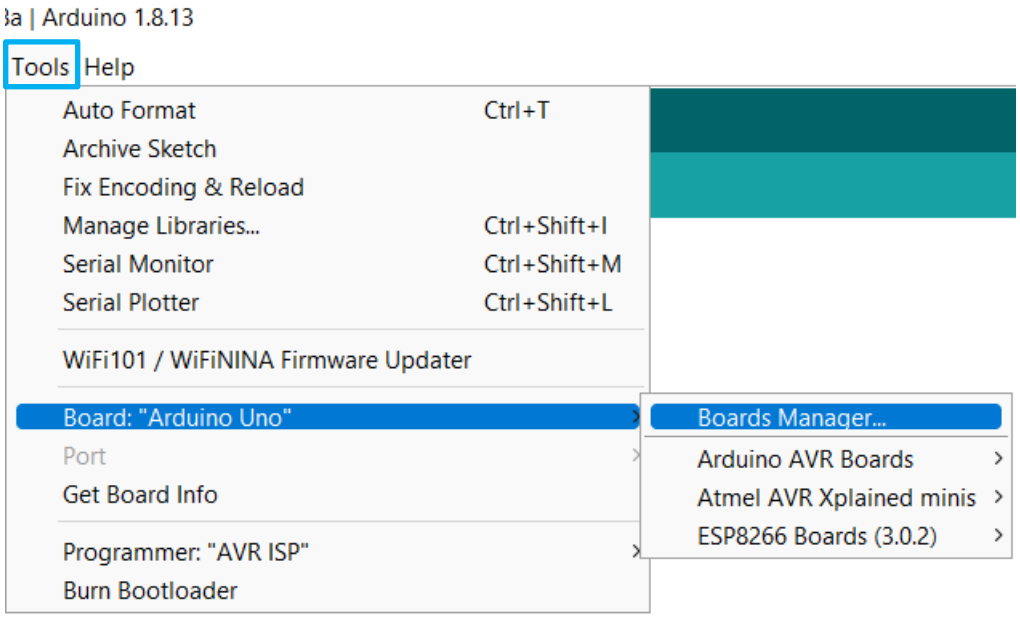


Figure 3-2: Figure of path must follow to open the Board Manager dialogue box

The Figure 3-2 was showed the path must follow to open the "Boards Manager" dialogue box. Below the "Boards Manager", the MCU boards which already been added to the Arduino IDE are shown.

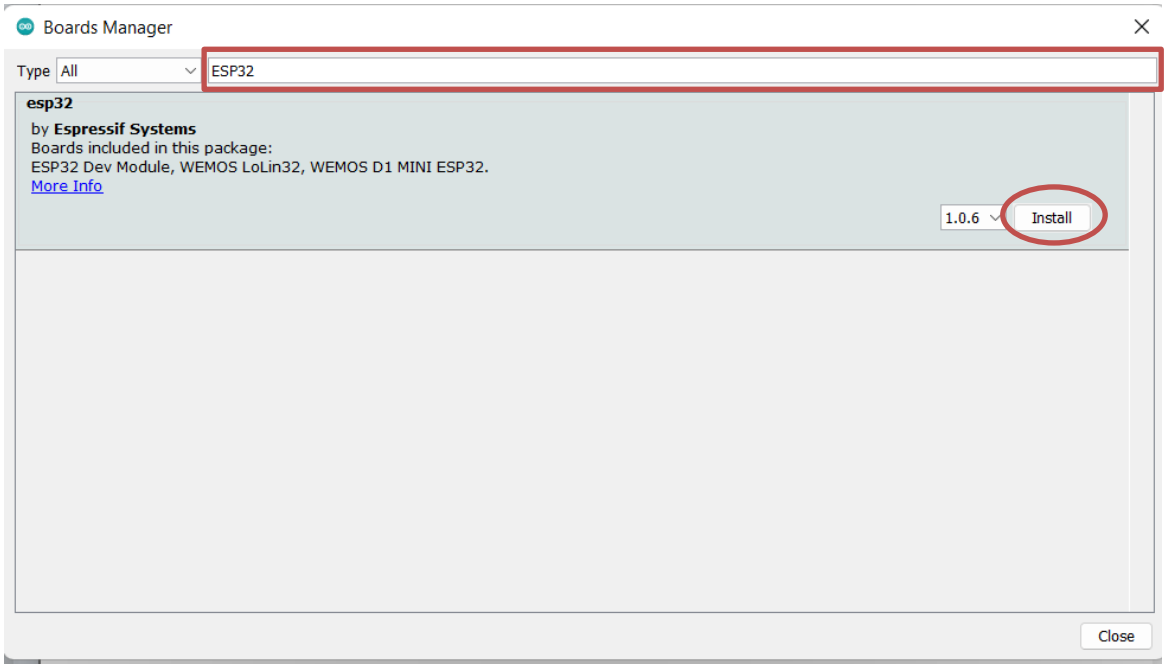


Figure 3-3: Figure of "Boards Manager" dialogue box

The Figure 3-3 showed the "Boards Manager" dialogue box after following the path shown in Figure 3-2. In this dialogue box, the Search bar was shown using the red colour box. The ESP32 was typed into the search bar and searched it. After searching the ESP32, the "esp32" board was shown. Then the "Install" button which showed a red colour circle

was clicked and installed it. Then the “Install” button was clicked and installed “ESP32 Boards”. After finishing the installation, the “close” button was clicked.

3.2 DHT-11 sensor connects with ESP32 MCU

In this part a DHT-11 sensor module, three female to female jumper wires and ESP32 MCU board were used.

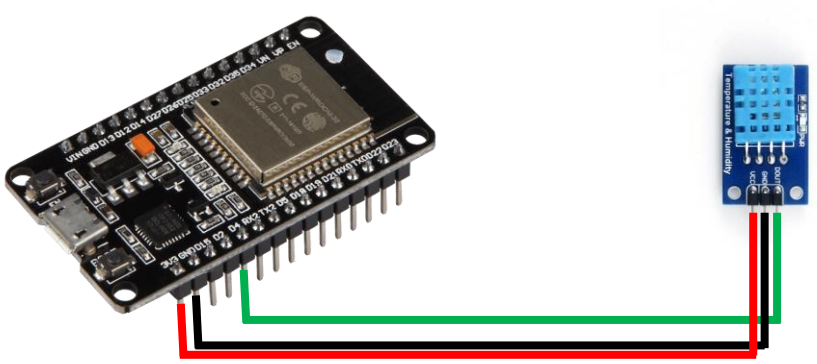


Figure 3-4: Figure of DHT-11 sensor connects with ESP32 MCU

According to Figure 3-4, the DHT-11 sensor module was connected to the ESP32 MCU. The VCC pin of DHT-11 was connected with 3V3 pin of ESP32. The Ground pin of DHT-11 was connected with the GND pin of ESP32. The Data pin of DHT-11 was connected with the D4 (ADC2/ GPIO4) pin of ESP32.

3.3 Creating and Setting up the Google Sheet

In this part, the Google sheet was used. Firstly, the web browser was opened and went to https://docs.google.com/spreadsheets/ web address. Then the blank sheet was created.

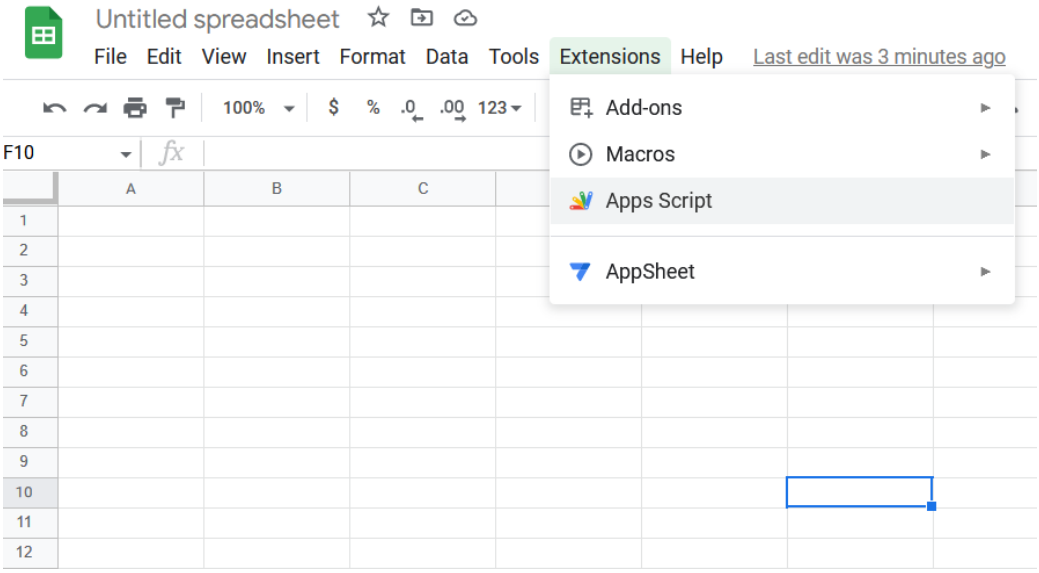


Figure 3-5: Figure of the path for opening the App Script web page in Google Sheet

Secondly, the “App Script” web page was opened by using “Extensions → Apps Script”. Then the “Apps Script” web page was opened. Figure 3-5 showed the path for opening the “Apps Scripts” web page.

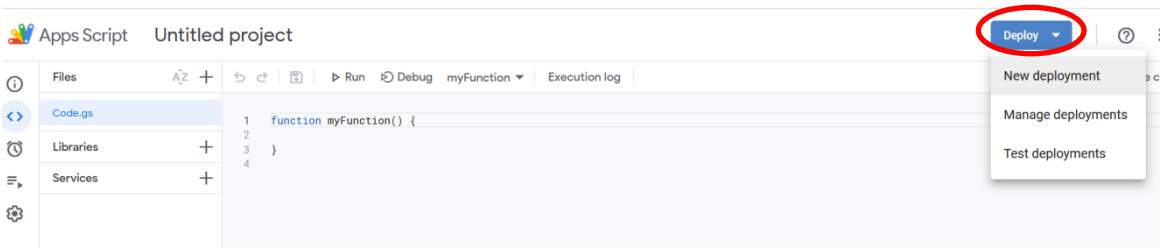


Figure 3-6: Figure of the path for creating the new deployment for Google Sheet

Then the “New deployment” was created using “Deploy → New deployment”. The red colour circle of Figure 3-6 is shown the “Deploy” button. Figure 3-6 showed the path for creating a new deployment.

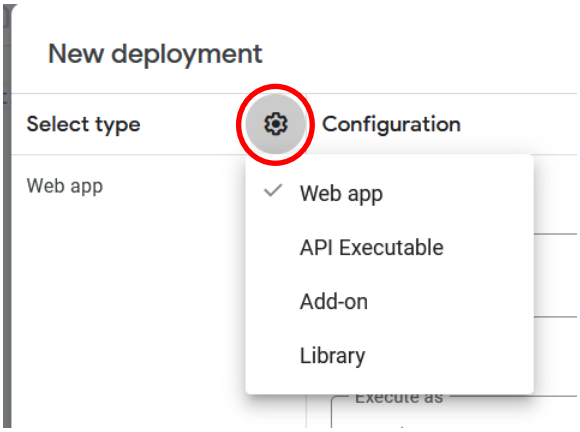


Figure 3-7: Figure of path for creating a new type for new deployment

Then the “New deployment” dialogue box was opened. The type of ‘Web app” was selected by using the setting button which has circled by red colour in Figure 3-7. Sometimes, the type of “Web app” can have automatically selected. Then the fields which have in the “New deployment” dialogue box were filled and clicked the button called “Deploy” which has right at the bottom of the dialogue box.

Finally, a dialogue box which has two URLs was opened after clicking the “Deploy” button. Then the notepad was opened and two links were copied from the “New deployment” dialogue box to the “Notepad” sheet. Then the ‘Done” button was clicked after copying the two URL links.

3.4 Installing the libraries in Arduino

The libraries were installed for getting the data from each sensor. In this part, the “WiFi.h”, “time.h”, “DHT.h”, “<Wire.h>” and “Adafruit_BMP085.h” libraries were installed.

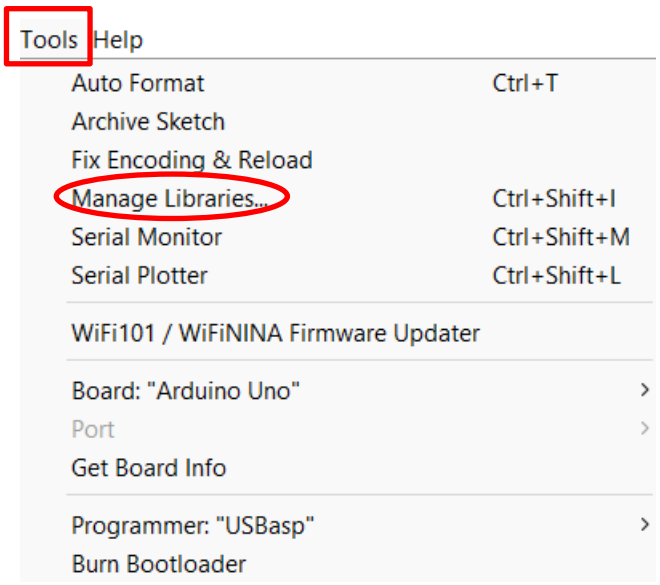


Figure 3-8: Figure of the path for opening the library installing dialogue box

First the library installing dialogue box was opened by using "Tools → Manage Libraries" path. The red colour ellipse of Figure 3-8 is shown the "Manage Libraries". The "Ctrl+Shift+I" is the short cut key for opening the "Library Manager" dialogue box.

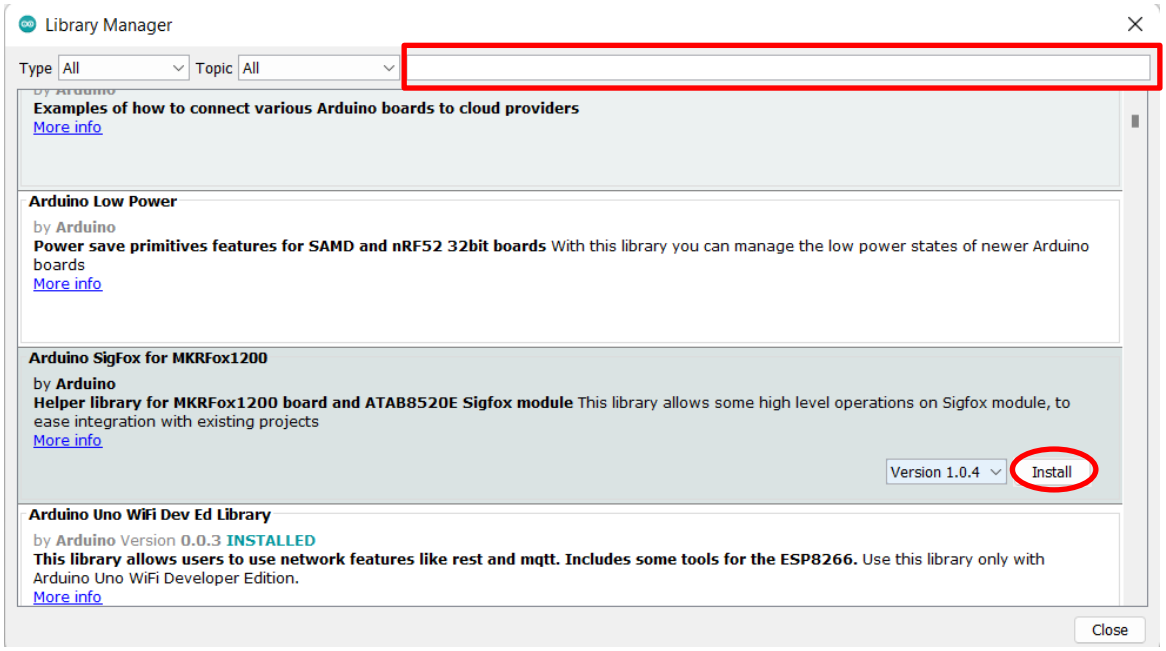


Figure 3-9: Figure of "Library Management" dialogue box

The "Library Manager" dialogue box shown in the figure 3-9 was opened after following the path shown in Figure 3-8. In this dialogue box, the Search bar was shown from the red colour box. The libraries were installed by clicking the "Install" button shown in the red circle in figure 3-9.

3.5 Code for getting the data from DHT-11

In this part, the "DHT.h" library was used. The following box was shown the code segment for getting data on temperature and humidity from DHT-11 Sensor.

```
1 #include "DHT.h"
2 #define DHTPIN 4
3 #define DHT_MOD_TYPE DHT11
4
5 DHT dht11(DHTPIN,DHT_MOD_TYPE);
6 void setup()
7 {
8
9   Serial.begin(115200);
10
11   dht.begin();
12 }
13
14 void loop()
15 {
16   float dhtHumidity = dht11.readHumidity();
17   float dhtTemperature = dht11.readTemperature();
18
19    //check the any reads fail to read, then try again dht11
20    if(isnan(dhtHumidity) || isnan(dhtTemperature)){
21     Serial.println(F("Failed read DHT11 sensor!!!"));
22     return;
23     }
24   Serial.print(F("Humidity: "));
25   Serial.println(dhtHumidity);
26   Serial.print(F("Temperature: "));
27   Serial.print(dhtTemperature);
28   Serial.println(F("°C "));
29 }
```

The "DHT.h" library was included by line number 1. The "D4" pin was defined as "DHTPIN" for getting the data from the sensor module by line number 2 and the sensor module type (DHT11) was defined as "DHT_MOD_TYPE" by line number 3. Then the data input pin number and data input sensor module type were defined as "dht11" by line number 5. The serial monitor was started and the 115200 was set up as boud by line number 9. After that, the "DHT-11" sensor was started by line number 11. "Humidity" and "Temperature" values were got from the data input pin and the value was converted to "float" by line numbers 16 and 17. "Humidity: " and "Temperature: " names were passed in serial monitor as flash-memory based strings typed by line numbers 24 and 26. The values of humidity and temperature were passed to the serial monitor by line number 25 and 27. Finally, the "0C" was passed to the serial monitor for printing the Celsius

symbol with the temperature value. The line number from 20 to 22 was written to get a message from the serial monitor when it failed to read the sensor.

3.6 Code for getting the data from LDR

In this part, the LDR sensor, lux meter and Octave Software were used. First, the lux meter was set upped using a dark room. Then the LDR sensor and lux meter were placed in the same position and got the values from the lux meter and analog values were given from the "Serial Monitor". In this type, several values were got for various light bright. Then the values were uploaded into octave software and defined the equation for converting the analog data of LDR to lux value. The following box was shown the code segment for getting the equation from Octave Software.

```
 1 clc;
 2 close all;
 3 close;
 4
 5 x = [0.4,130,307,500,688,981,1410,1670,2000];
 6 y = [634,531,455,380,314,226,128,87,60];
 7
 8 p = polyfit(x, y, 2)
 9 v = polyval(p, x);
10
11 title("Graph of lux value vs resistance value")
12 xlabel("lux value")
13 ylabel("resistance value")
14 plot(x,y,"x","MarkerEdgeColor","black")
15 hold on
16 plot(x, v)
17 hold off
18 grid on;
```

In this code segment, all data was cleared when starting the new "Run" in line numbers 1,2,3. The 'x' values and 'y' values were added in line number 5 and 6. Then the "Polyfit" command was added in line number 8 to get the curve line according to the given coordinates. Then from line number 11 to 18, the graph was plotted with axis and title. Finally, the equation was made for getting the real lux values for given analog resistance values.

$$y = 0.0047205x^2 - 6.502373x + 2251.8445705 \quad \text{----------(eq:1)}$$

Then the real water level values was sent to the google sheet using above (eq:1) equation.

The following box was shown the code segment for getting data on light intensity of sun by using LDR.

```
1 #define LDRPIN 36
2
3 void setup() {
4   Serial.begin(115200);
5 }
6
7 void loop() {
8
9   int analogValueLDR = analogRead(LDRPIN);
10  float luxvalue = (0.00010213*(analogValueLDR*analogValueLDR))-
11 (0.4717*analogValueLDR)+590.3023;
12  Serial.print("Analog Value = ");
13  Serial.print(analogValueLDR);
14  delay(500);
```

The “VP (ADC1-0)” pin was defined as “LDRPIN” for getting the data from the sensor by line number 1.The serial monitor was started and the 115200 was set up as *boud* by line number 4. After that, the analog value of light resistance was got from the data input pin. The value was converted getting an analog value to “int” (as integer) with the name called “analogValueLDR” by line number 9. In line number 10, the “analogValueLDR” analog value was got and added as ‘x’ in equation 2 (eq:1). Then the value which giving from the equation was converted to float and the “luxValue” name was given. “Analog Value: ” name was passed in the serial monitor by line number 11. The integer value got from the data input pin was passed to the serial monitor by line number 12. Finally, the “delay” was set up to get some brake for the loop.

3.7 Code for getting the data from BMP180 sensor

In this part, the “Wire.h” library and “Adafruit_BMP085.h” library were used. The following box was shown the code segment for getting data on pressure from BMP180 Sensor.

```
1 #include <Wire.h>
2 #include <Adafruit_BMP085.h>
3
4 Adafruit_BMP085 bmp;
5
6 void setup() {
7  Serial.begin(115200);
8   if (!bmp.begin()) {
```

```
 9   Serial.println("Could not find the BMP180 sensor, check
10 connections!!!");
11   while (1) {}
12   }
13 }
14
15 void loop() {
16
17     float bmpPressure = bmp.readPressure();
18
19     Serial.print("Pressure = ");
20     Serial.print(bmpPressure);
21     Serial.println(" Pa");
22   delay(500);
23 }
```

The “Wire.h” library and “Adafruit_BMP085.h” library were included by line number 1 and line number 2. The “Adafruit_BMP085” was defined as “bmp” for getting the data from the sensor module by line number 4. The serial monitor was started and the 115200 was set up as *boud* by line number 7. After that, the “BMP180” sensor was started by line number 8. In line number8, if the data signal pin was disconnected, the message was printed "check the connection" in “Serial monitor”. Then “while” loop was started by line number 11 if it happened. Then the value of pressure was got from the “BMP180” sensor and the value was converted as “float” by line number 17. “Pressure: ” name was passed in serial monitor as strings typed by line number 19. The values of pressure were passed to the serial monitor by line number 20. Finally, the “ Pa” was passed to the serial monitor for printing the pressure SI unit with pressure value.

3.8 Code for getting the data from Rain Drop sensor

The following box was shown the code segment for getting data about rainy situations in a day.

```
1  #define rainAnalogPin 35
2  void setup() {
3    Serial.begin(115200);
4    pinMode(rainDigitalPin, INPUT);
5  }
6  void loop() {
7    int rainAnalogVal = analogRead(rainAnalogPin);
8    if(rainAnalogVal<=4000){
9        Serial.println("Light Rain");
```

```
10 }else if(rainAnalogVal<=3500){
11      Serial.println("Medium Rain");
12   }else if(rainAnalogVal<=2000){
13       Serial.println("Heavy Rain");
14   }else{
15       Serial.println("No Rain");
16   }
17 }
```

The “D35” pin was defined as “rainAnalogPin” for getting the data from the sensor module by line number 1. The serial monitor was started and the 115200 was set up as boud by line number 3. Then the ‘pinMode’ was defined as the input pin by line number 4. After that, the analog value of rain drop sensor was got from the data input pin. The value was converted getting an analog value to “int” (as integer) with the name called “rainAnalogVal” by line number 7. Then the “if” condition was set up for getting the details about the rain condition. The rain conditions were defined as “Light Rain”, “Medium Rain”, “Heavy Rain” and “No Rain”. Then the conditions were passed to the serial monitor.

3.9 Code for getting the data from Water Level sensor

3.9.1 Calibration the Water Level Sensor

In this part, the water level sensor, a ruler, thin paper layer, a pen, a stick tape, a cylinder cup, Octave Software and water were used.

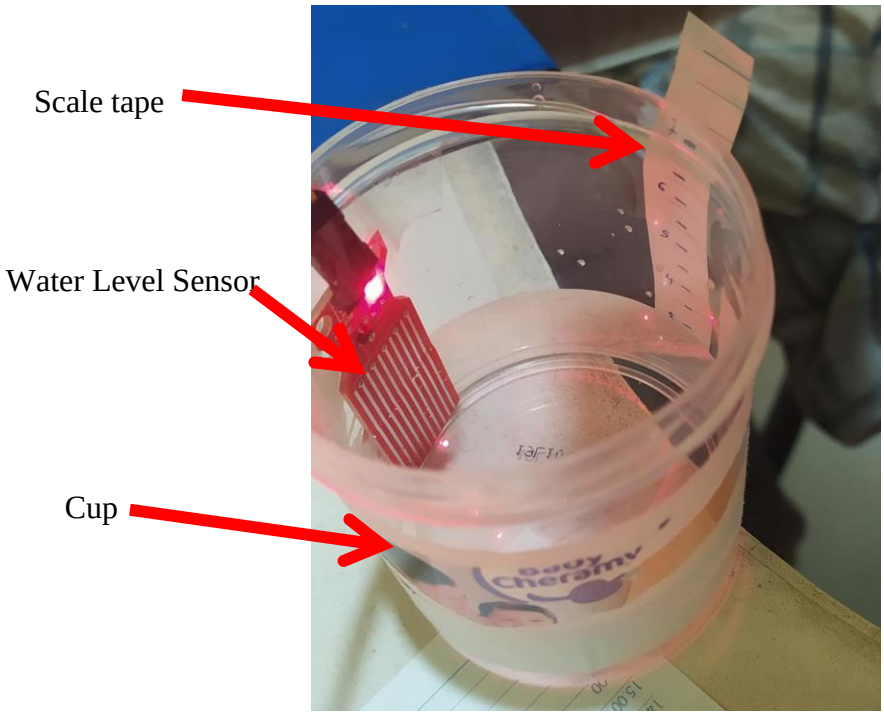


Figure 3-10: Figure of calibration the Water level Sensor

First the scale tape with scale number was made using ruler, pen and paper layer. Then the scale tape was attached outer side of cylinder cup using stick tape. Then the "Water Level sensor" was added into cup and wired. Then the following box was shown the code segment for getting data of water level in cup.

```
1 #define rainwaterLevelPin 32
2 void setup() {
3   Serial.begin(115200);
4   pinMode(rainwaterLevelPin, INPUT);
5 }
6 void loop() {
7   float rainwaterLevelOld = analogRead(rainwaterLevelPin);
8   Serial.print("Rain Level: ");
9   Serial.println(rainwaterLevel);
10  delay(1000);
11 }
```

The "D32" pin was defined as "rainWaterLevelPin" for getting the data from the sensor module by line number 1. The serial monitor was started and the 115200 was set up as boud by line number 3. Then the 'pinMode' was defined as the input pin by line number 4. After that, the analog value of rain drop sensor was got from the data input pin. The value was converted getting an analog value to "float" with the name called "rainWaterLevelOld" by line number 7. "Water Level: " name was passed in serial monitor as strings typed by line number 8. The values of water level were passed to the serial monitor by line number 9.

The code was uploaded. Then the water was added into cup and noted the analog value in the serial monitor and the value of the scale tape. In there, the water was added near the top of the sensor and got values. After that, the water was reduced and got the values. Then the median of values was calculated. Finally, the values were plotted using octave software. The following box was shown the code segment for plotting data for get the graph in Octave.

```
1 clc;
2 close all;
3 close;
4
5 x = [0,0.6,1.2,1.5,2,2.4,3,3.5,4];
6 y = [0,148,198,239,260,295,321,295,354];
```

```
 7
 8 p = polyfit(x, y, 1)
 9 v = polyval(p, x);
10
11 title("Graph of Analog value vs Water level")
12 xlabel("Water Level (mm)")
13 ylabel("Analog Value")
14 plot(x,y,"x","MarkerEdgeColor","black")
15 hold on
16 plot(x, v)
17 hold off
18 grid on;
```

In this code segment, all data was cleared when starting the new "Run" in line numbers 1,2,3. The 'x' values and 'y' values were added in line number 5 and 6. Then the "Polyfit" command was added in line number 8 to get the most fitted straight line according to the given coordinates. Then from line number 11 to 18, the graph was plotted with axis and title. Finally, the equation was made for getting the real water level values for given analog values.

$$y = 0.011303x - 0.62776 \quad \text{----------(eq:2)}$$

Then the real water level values was sent to the google sheet using above (eq:2) equation.

```
 1 #define rainwaterLevelPin 32
 2 void setup() {
 3   Serial.begin(115200);
 4   pinMode(rainwaterLevelPin, INPUT);
 5 }
 6 void loop() {
 7   float rainwaterLevelOld = analogRead(rainwaterLevelPin);
 8   float rainwaterLevel = (74.182*(rainwaterLevelOld)) + 84.431;
 9
10   Serial.print("Rain Level: ");
11   Serial.println(rainwaterLevel);
12   delay(1000);
13 }
```

The "D32" pin was defined as "rainWaterLevelPin" for getting the data from the sensor module by line number 1. The serial monitor was started and the 115200 was set up as boud by line number 3. Then the 'pinMode' was defined as the input pin by line number 4. After that, the analog value of rain drop sensor was got from the data input pin. The value was converted getting an analog value to "float" with the name called

“rainWaterLevel” by line number 7. In line number 8, the “rainWaterLevelOld” analog value was got and added as ‘x’ in equation 2 (eq:2). Then the value which giving from the equation was converted to float and the “rainWaterLevel” name was given. “Water Level: ” name was passed in serial monitor as strings typed by line number 10. The values of water level were passed to the serial monitor by line number 11.

3.10 Code for send the data from ESP32 to Google sheet

3.10.1 Code for add setting in google sheet

The following box was shown the code segment for receiving data from ESP32 to Google Sheet. It should be inserted into the position shown in Figure 3-6.

```
  function doGet(e) {
    Logger.log( JSON.stringify(e) );  // view parameters
    var result = "Ok"; // assume success
    if (e.parameter == "undefined") {
1     result = "No Parameters";
2   }
3   else {
4     var sheet_id = "1Nv0p9m9BLif_gv8G9vCqd4lrwkLNEH9ZUqZ0RlRUUvU";
5     var sheet = SpreadsheetApp.openById(sheet_id).getActiveSheet();
6     var newRow = sheet.getLastRow() + 1;
7     var rowData = [];
8     d=new Date();
9     rowData[0] = d; // Timestamp in column A
10    rowData[1] = d.toLocaleTimeString(); // Timestamp in column A
11
12    for (var param in e.parameter) {
13      Logger.log("In for loop, param=" + param);
14      var value = stripQuotes(e.parameter[param]);
15      Logger.log(param + ":" + e.parameter[param]);
16      switch (param) {
17        case "Temperature": //Parameter 1, It has to be updated in
18 Column in Sheets in the code, orderwise
19          rowData[2] = value; //Value in column C
20          result = "Written on column A";
21          break;
22        case "Humidity": //Parameter 2, It has to be updated in
23 Column in Sheets in the code, orderwise
24          rowData[3] = value; //Value in column D
25          result += " Written on column B";
26          break;
27        case "LDR": //Parameter 3, It has to be updated in Column
28 in Sheets in the code, orderwise
29          rowData[4] = value; //Value in column E
30          result += " Written on column C";
31          break;
32        case "Pressure": //Parameter 4, It has to be updated in
```

```
36 Column in Sheets in the code, orderwise
37           rowData[5] = value; //Value in column F
38           result += " Written on column D";
39           break;
40         case "Rain": //Parameter 2, It has to be updated in Column
41 in Sheets in the code, orderwise
42           rowData[6] = value; //Value in column G
43           result += " Written on column E";
44           break;
45         case "Wind": //Parameter 2, It has to be updated in Column
46 in Sheets in the code, orderwise
47           rowData[6] = value; //Value in column H
48           result += " Written on column F";
49           break;
50         case "WDir": //Parameter 2, It has to be updated in Column
51 in Sheets in the code, orderwise
52           rowData[6] = value; //Value in column I
53           result += " Written on column G";
54           break;
55         default:
56           result = "unsupported parameter";
57       }
58     }
59     Logger.log(JSON.stringify(rowData));
60     // Write new row below
61     var newRange = sheet.getRange(newRow, 1, 1, rowData.length);
62     newRange.setValues([rowData]);
63   }
64   // Return result of operation
65   return ContentService.createTextOutput(result);
66 }
67 function stripQuotes( value ) {
68   return value.replace(/^["']|['"]$/g, "");
69 }
```

In line number 1, the function was started. Then “View parameters” was started by line number 2. Then the “if” parameter was started by line number 4 for to execute when an irrelevant value is encountered. Then the line number 7, the “elese” function was started for execution when a data value arrives. The “Google Sheet Id” was added by line number 8. When the data arrives, directing the data to the columns and rows where it should go from line number 9 to 12. For rows where data is entered, when the data is entered, the data and time of that moment is entered by line number 12 to 14. Then when

entering data into Column and Rows, it was entered in String from line number 16 to 20. After that, the data related to the temperature was first referred to the relevant column by line number 23,24 and 25. In this way, columns were separated for entering the remaining data such as Humidity, LDR value, Pressure value, rain condition, rain volume and wind speed. finally by line number 61 and 62, the received data was inserted into the next row.

3.10.2 Code for send the data from ESP32

The following box was shown the code segment for sending data from ESP32 to Google Sheet.

```
1 #include "WiFi.h"
2 #include <HTTPClient.h>
3
4 const char* ssid = "WiFi SSID";
5 const char* password = "WiFi password";
6
7 String GOOGLE_SCRIPT_ID = "Google sheet ID";
8
9 void setup() {
10   delay(1000);
11   Serial.begin(115200);
12   delay(1000);
13
14   Serial.println();
15   Serial.print("Connecting to wifi: ");
16   Serial.println(ssid);
17   Serial.flush();
18   WiFi.begin(ssid, password);
19   while (WiFi.status() != WL_CONNECTED) {
20     delay(1000);
21     Serial.print(".");
22   }
23 }
```

The “WiFi.h” library and “HTTPClient.h” library were included by line number 1 and line number 2. Then two mutable pointers to an immutable character/string functions were defined as “ssid” and password” for entering the ssid and password of connected WiFi by line numbers 4 and 5. Then a “String” was defined as “GOOGLE_SCRIPT_ID” which must receive data from ESP32 by line number 7. Then the “Void Setup” was started and the serial monitor was started and the 115200 was set up as boud by line number 11. Then “Connecting to wifi: ” name and “ssid” were printed in Serial monitor by line number 15 and 16. Then the “Serial.flush” function was defined for waiting for the transmission of outgoing serial data to be complete by line number 17. After that, the WiFi was started and the ssid and password were checked as it is correct. Then the

“while” loop was defined for connecting the Wifi of ESP32 with WiFi router. For wait time, the dots were printed.

```
24 void loop() {
25   if (WiFi.status() == WL_CONNECTED) {
26     HTTPClient http;
27     String urlFinal = "https://script.google.com/macros/s/" +
28 GOOGLE_SCRIPT_ID + "/exec?read";
29     Serial.println("POST data to spreadsheet: ");
30     Serial.println(urlFinal);
 8         HTTPClient http;
 9         http.begin(urlFinal.c_str());
10         http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
11         int httpCode = http.GET();
12         Serial.print("HTTP Status Code: ");
13         Serial.println(httpCode);
14         String payload;
15         if (httpCode > 0) {
16             payload = http.getString();
17             Serial.println("Payload: "+payload);
18         }
19           http.end();
20     }
21   delay(1000);
22 }
```

Then the void loop was started. In line number 2, the connection of WiFi was checked using “if” function. Then the “HTTPClient” library was called by line number 3. In line number 4, the “url” of the google sheet and must sending data were added. Then the “Post data to sheet” was printed by Serial monitor after sending the data from ESP32 to Google Sheet. Then the “url” was printed. Then the function of “HTTPClient” library was called and set upped by line number from 8 to 18. Finally, the “http” was ended by line number 19.

3.11 Creating and settings up ThingSpeak

In this part, the "ThingSpeak" and Arduino IDE were used. Firstly, the web browser was opened and went to https:// thingspeak.com/channels/ web address. Then the "ThingSpeak" was opened. Then the new channel was created clicking on "new Channel". The below dialogue box was opened.

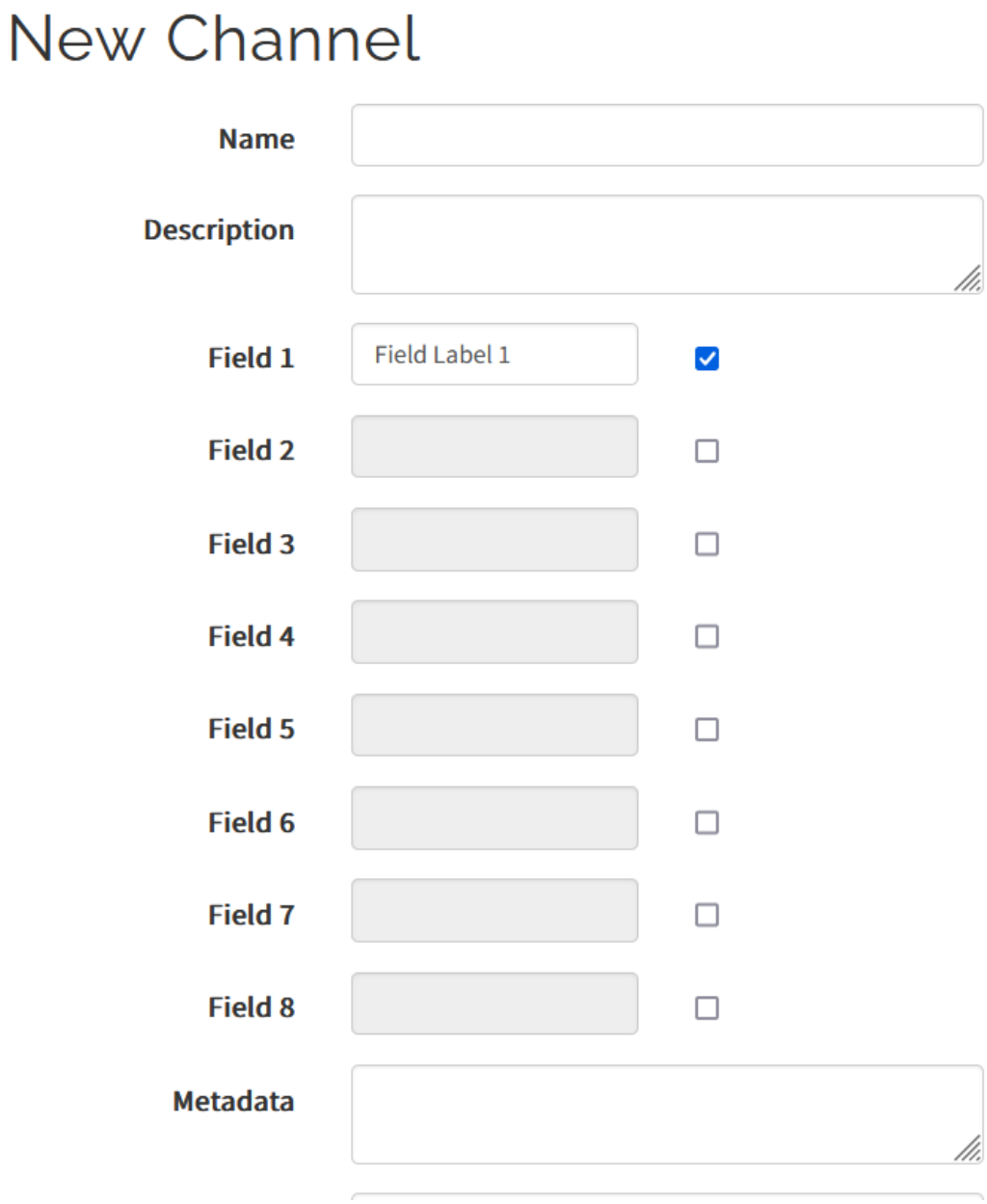


Figure 3-11: Figure of channel setting dialogue box of "ThingSpeak

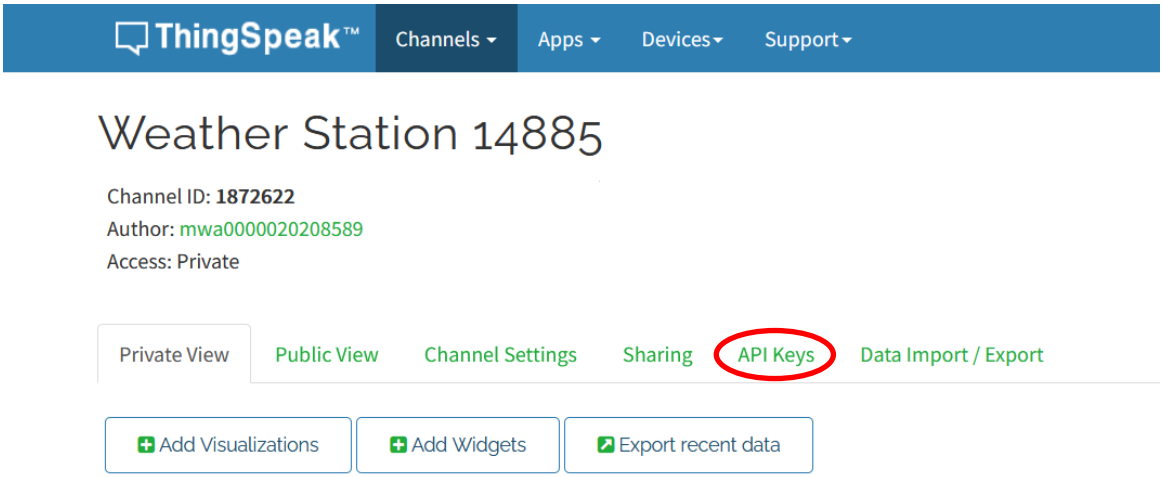


Figure 3-12: Figure of web page view of ThingerSpeak

Then the "API Keys" word which has middle of red ellipse was clicked and go to "API Keys" page.

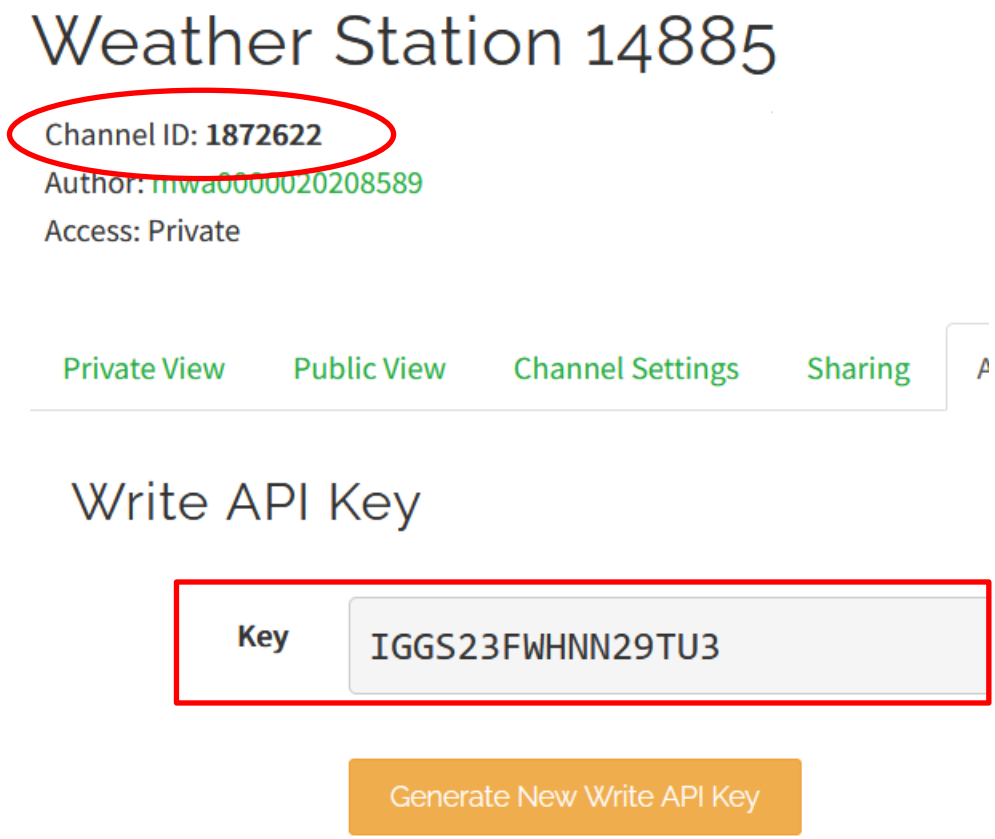


Figure 3-13: Figure of API Keys page of ThingSpeak

Then the “channel ID” numbers which has middle of red colour ellipse was copied and the “Key” which has middle of red colour box was copied. Then open the “Arduino IDE” and create a code segment for sending the data from “ESP-32 to “ThigSpeak”. The code was following box.

```
 1 #include "ThingSpeak.h"
 2 unsigned long thingsPeakID = 1872622;
 3 const char *writeAPIKey = "IGGS23FWHNN29TU3";
 4 WiFiClient client;
 5
 6 void setup() {
 7   ThingSpeak.begin(client);
 8
 9 void loop() {
10   ThingSpeak.setField(1,dhtTemperature);
11   ThingSpeak.setField(2,dhtHumidity);
12   ThingSpeak.setField(3,pressure);
13   ThingSpeak.setField(4,luxvalue);
14   int respons1 = ThingSpeak.writeFields(thingsPeakID,writeAPIKey);
15 }
```

The “ThingSpeak.h” library was included by line number 1. Then the “ThingSpeak ID” was written in line number 2 as “thingsPeakID” using ‘unsigned long’ type. Then the “API Key” was added as “writeAPIKey” using ‘const char’ (Constant Character). Then the “WiFiClient” was called “client” in line number 4. Then the “ThingSpeak” was begun with line number 7. Then the fields were set to the “ThingSpeak” for analysing the data in line numbers from 10 to 14. The numbers were equelled to the field which was created in “ThingSpeak”.

3.12 Creating a PCB for Circuit and Creating a Stand for it

In this part, the EasyEDA software was used. First the ESP-32 Devkit VI micro computer unit waw added into new sheet. Then the two of four headers, two of three headers, one of two headers, one of five headers were added into sheet. Then the all headers connected as following figure.

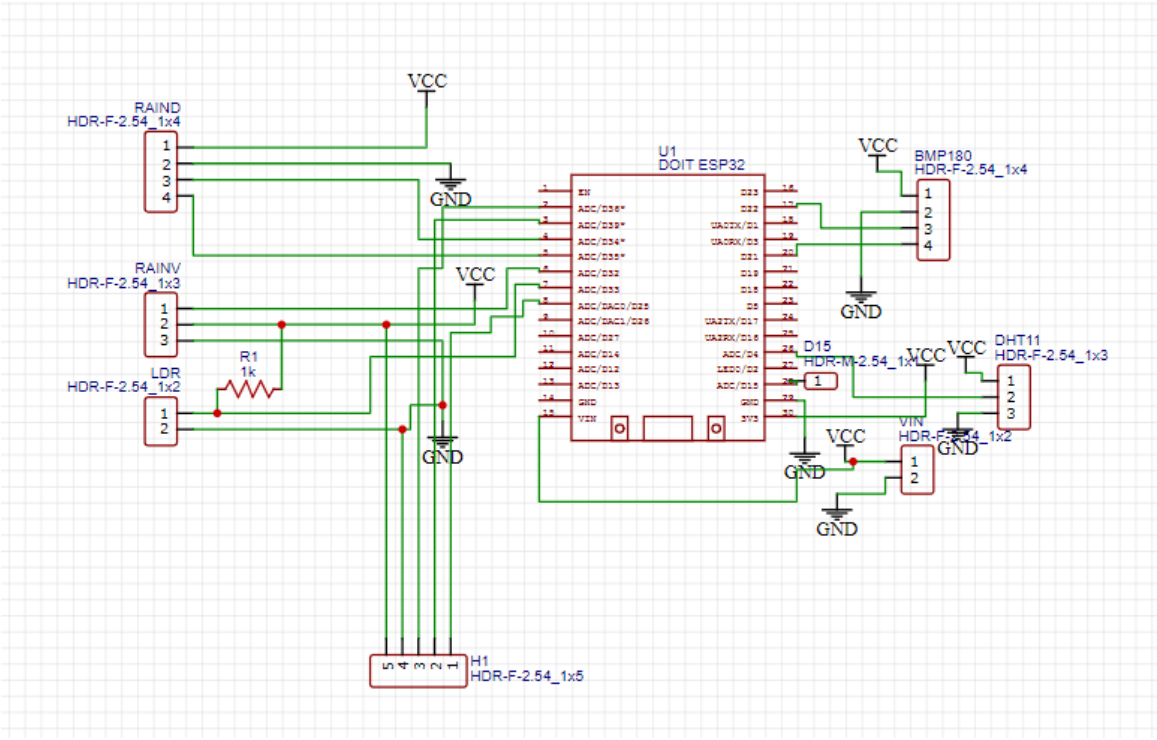


Figure 3-14: figure of Circuit design for creating the PCB

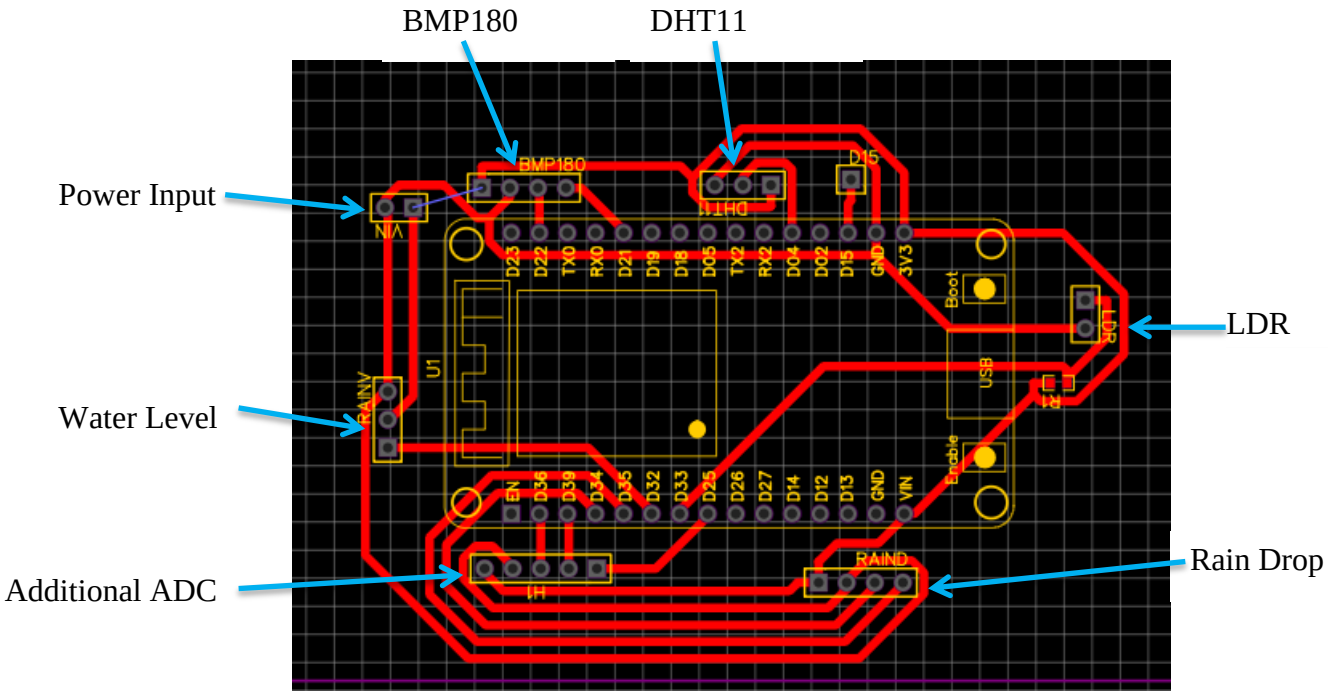


Figure 3-15: PCB Schematic for Weather circuit

The PCB is set up as in Figure 3-14 and only the ESP-32 is connected to it. Sensors are connected to the circuit using jumper wires.

Then the stand was created for attaching the weather station. In this part a old broom with a steel handle, a lithium battery for giving the power into circuit, piece of wood, four screw nails with nut, a drill, a cylinder cup, some piece of half inch PVC pipe and jumper wires were used.

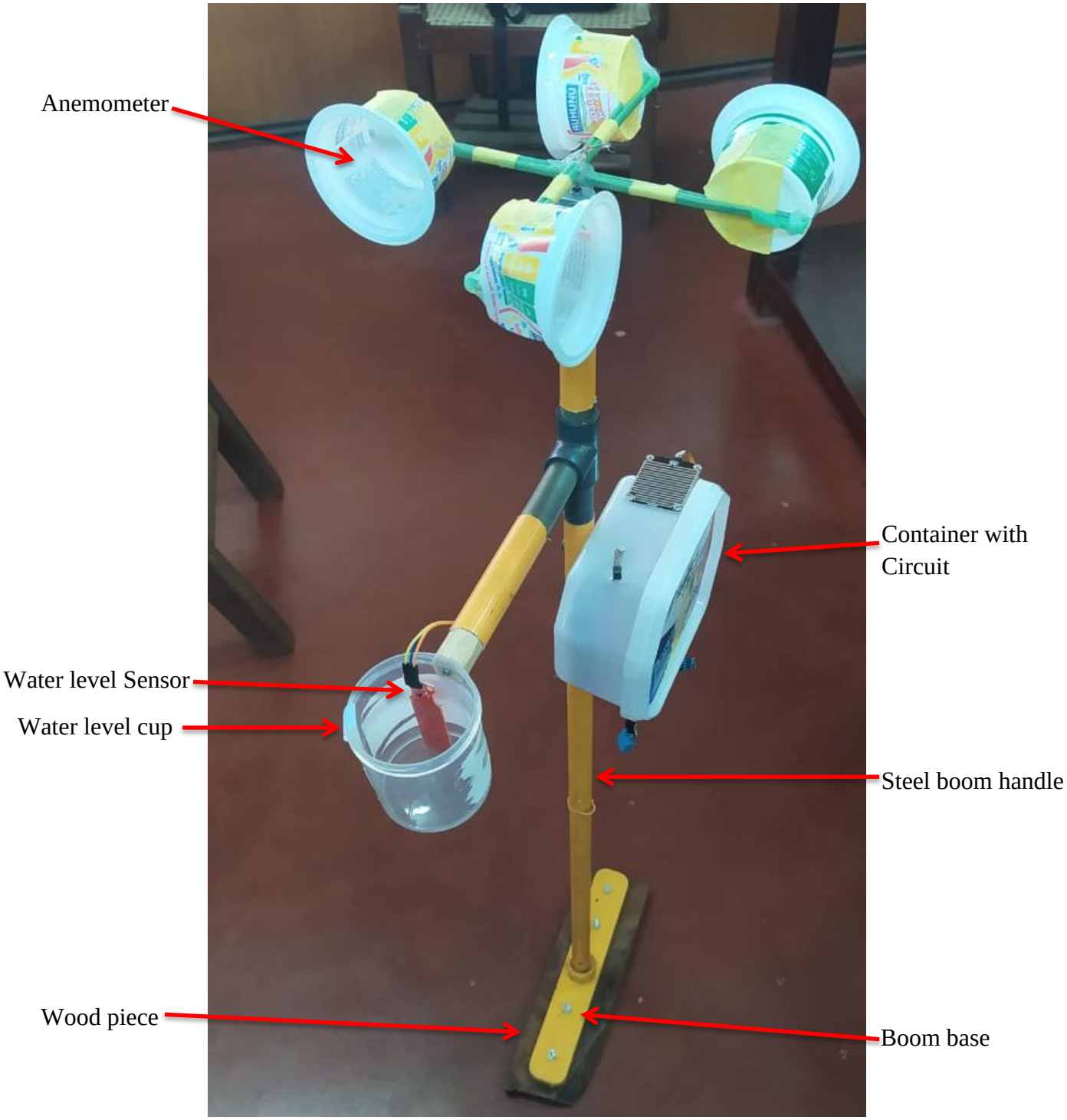


Figure 3-16: Figure of Complete installation after support is made

The layout is set up as shown in Figure 3-16. There, the circuit board, battery and sensors are connected as shown in Figure 3-17.

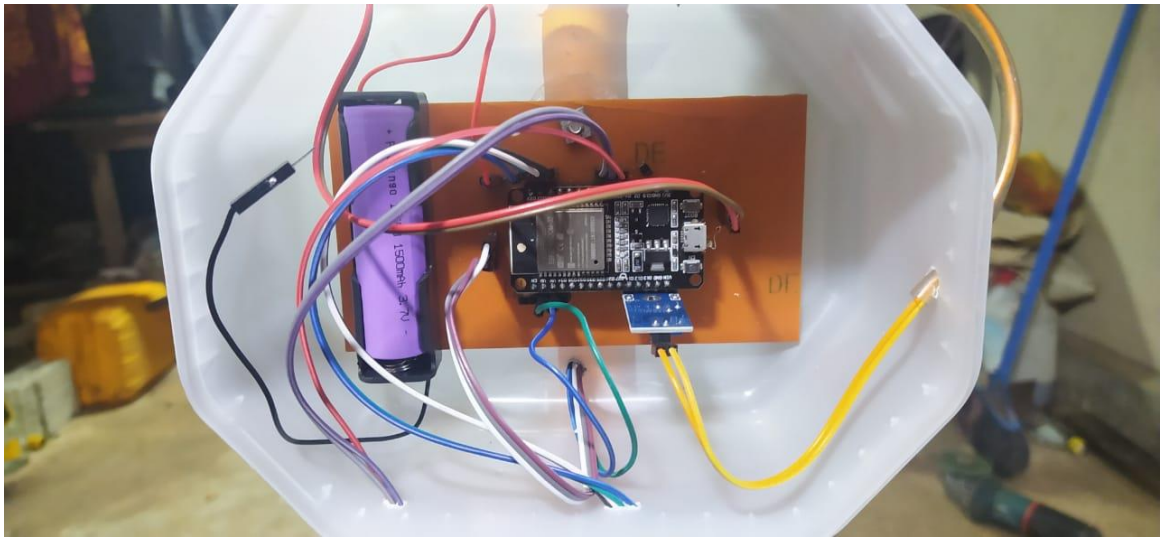


Figure 3-17: Figure of how the circuit is connected to the box

For this part, a disposable ice container and a screw and two nut about two inches long were used. First of all, the container is firmly attached to the support using screws. A hole was then drilled in the top of the PCB and connected to the rest of the pin.

3.13 Final View of ThingSpeak

Finally, the ThingSpeak output view after all the settings were set was as follows.

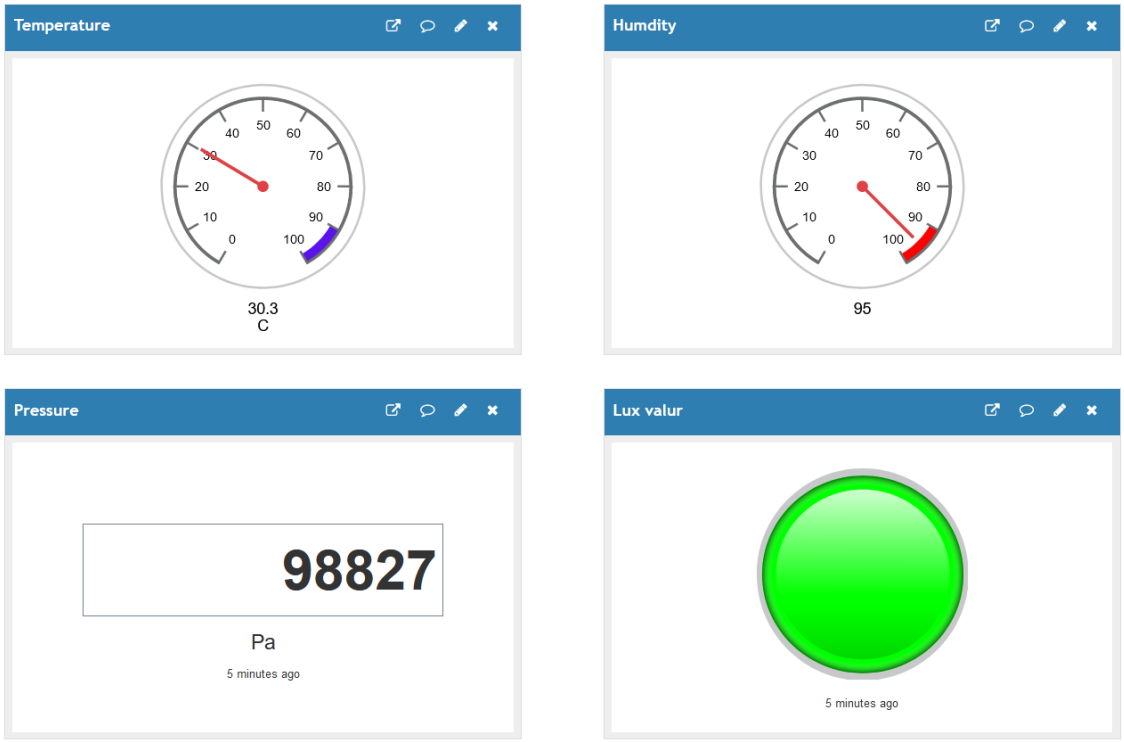


Figure 3-18: Final output view of ThingSpeak

A "Gauge" type display was used to indicate Temperature, Humidity. Also a "Numeric Display" was used to indicate the pressure value. A "Lamp Indicator" was used to indicate the light intensity provided by the LDR.

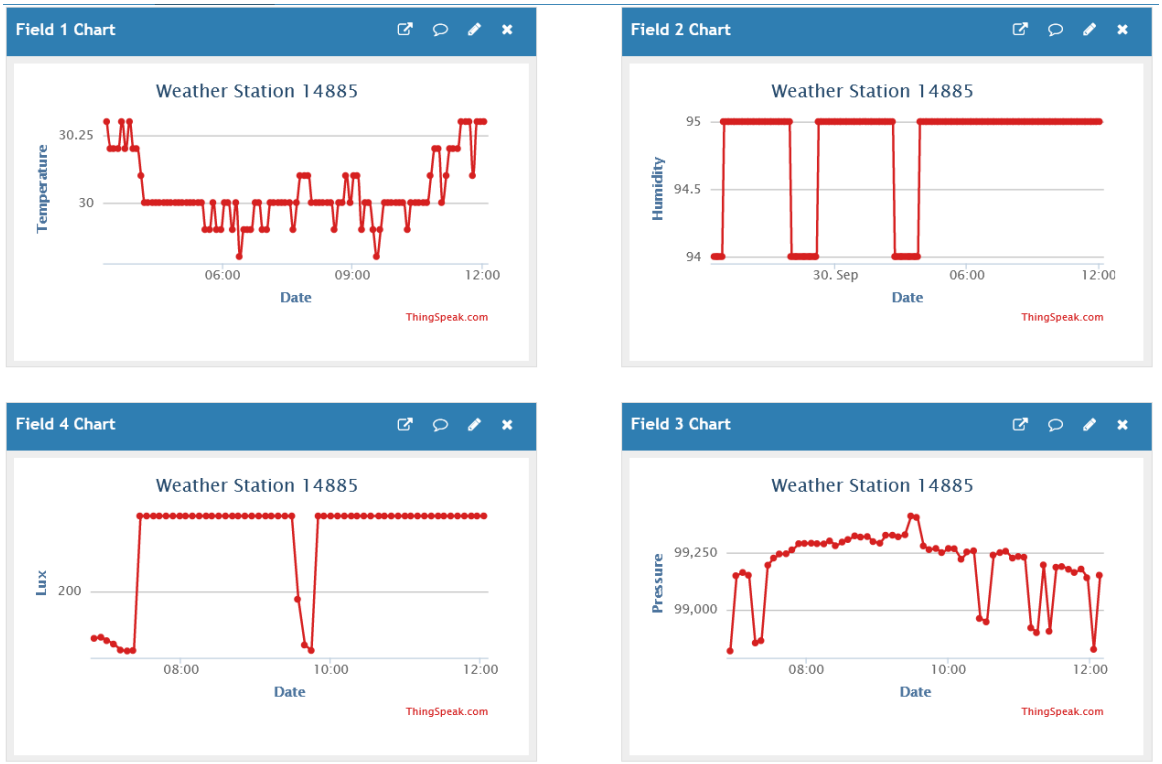


Figure 3-19: Figure of ThingSpeak Graph output

Later, graphs were used to show all the Humidity, Temperature, Light intensity and pressure values. All these charts are set to update live.

4 RESULTS AND ANALYSIS

4.1 Results of Sensor calibration

4.1.1 Result of LDR Calibration

Table 4-1: Table of lux values with its resistance values

Lux Value	Resistance Value
0.4	634
130	531
307	455
500	380
688	314
981	226
1410	128
1670	87
2000	60

According the table 4-1, during the night the resistance peak is high. When it becomes bright, it gradually decreases.

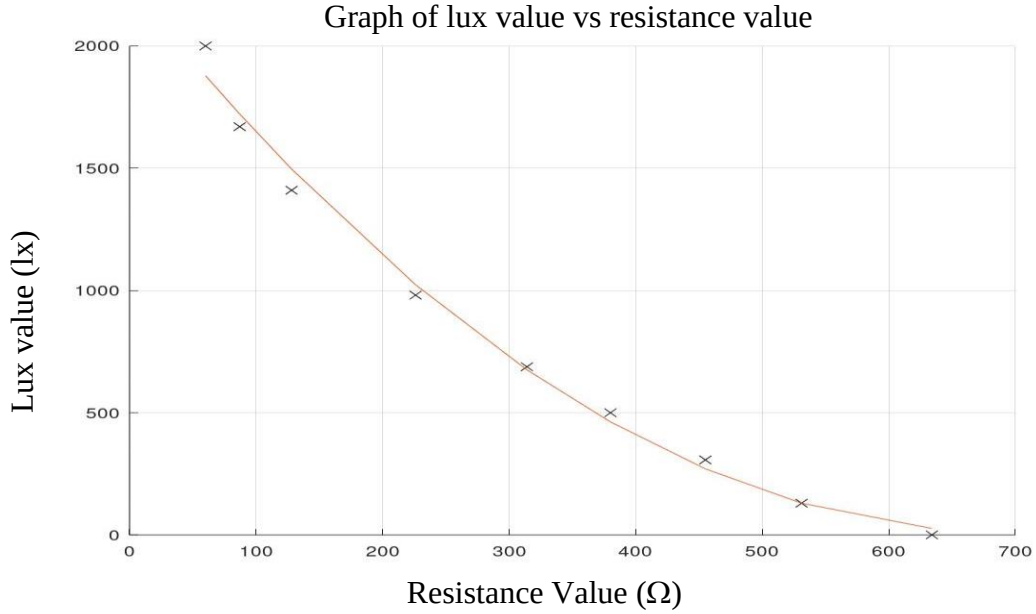


Figure 4-1: Graph of lux values versus resistance values

According to the figure 4-1, the graph type was the curve. During the night the resistance peak is high. When it becomes bright, it gradually decreases. The gradient was "-6.502373" and the intersection was "2251.8". The coefficient of the square term was "0.0047205". The following equation was constructed from this graph.

$$y = 0.0047205x^2 - 6.502373x + 2251.8445705 \quad \text{----------(eq:1)}$$

4.1.2 Results of Water Level sensor calibration

Table 4-2: Table of water level with its analog value and its median

Water level (±0.01)	Analog Value increase	Analog Value decrease	Median
0	0	10	5
0.5	131	165	148
1	170	226	198
1.5	220	258	239
2	259	281	270
2.5	289	301	295
3	291	338	314.5
3.5	315	349	332
4	354	354	354

According the table 4-2, the analog value gradually increased as the water was increased and decreased correspondingly as the water was decreased.

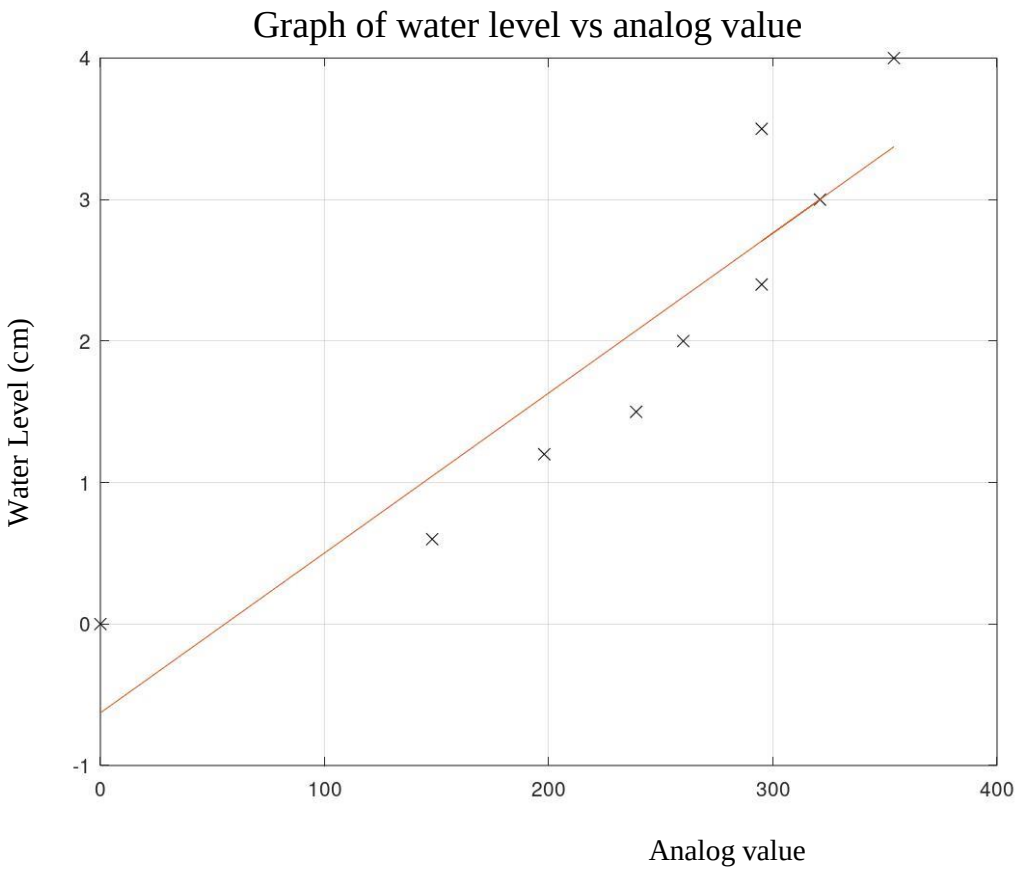


Figure 4-2: Graph of water level versus its analog value

According to the figure 4-2, the graph type was the straight line. The analog value gradually increased as the water was increased and decreased correspondingly as the water was decreased. The gradient was "0.0113" and the intersection was "-0.6278". The following equation was constructed from this graph.

$$y = 0.011303x - 0.62776 \quad \text{----------(eq:2)}$$

4.2 Result of the individual part of Weather Stations

In this section, in the individual project, the temperature, humidity, Pressure and intensity of Sun data values were obtained and the data was analysed. The time delay was 5 minutes. Also, Water Level sensor and Rain Drop sensor were connected. But since there was no rain on the days tested, the values were recorded as 0 ml and "No Rain".

4.2.1 Temperature Result for individual part

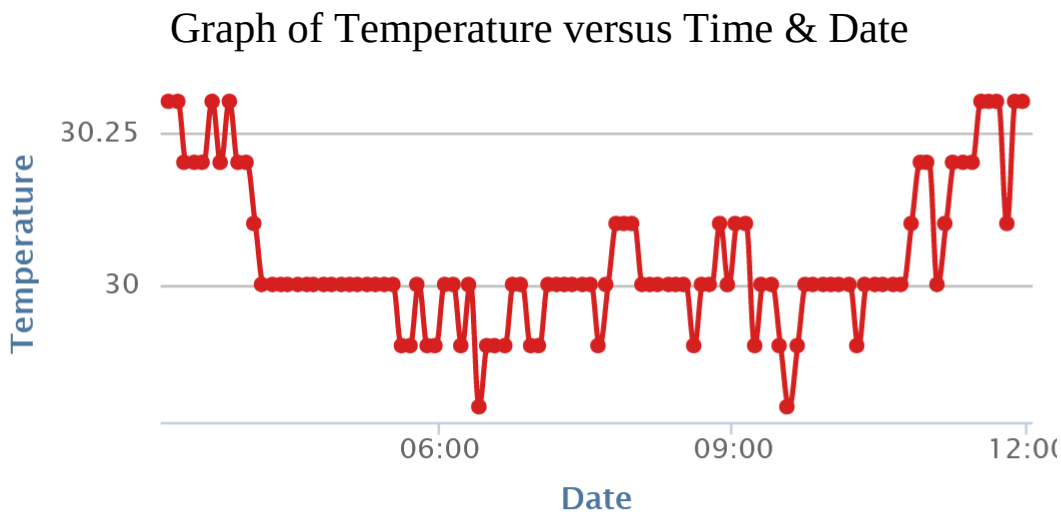


Figure 4-3: Figure of Temperature variation graph

According to the picture above, the temperature in Rajagiriya area was around 30.25 ^{0}C from 3.00 am to 4.00 am in the morning, and 30 ^{0}C from 4.00 am to 5.50 am. From 6.00 am to 10.30 am it varied between 29.75 ^{0}C and 30.25 ^{0}C, and after 10.30 am the temperature gradually increased to near 30.5 ^{0}C.

4.2.2 Humidity Result for individual part

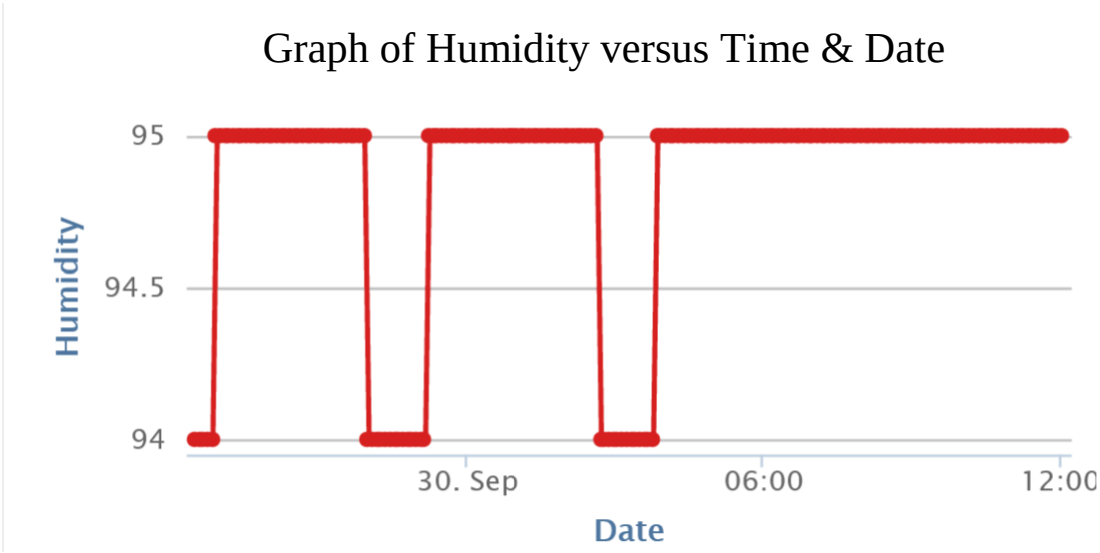


Figure 4-4: Figure of Humidity variation graph

According to the above picture, the data obtained from 9 pm on September 29 to 12 noon on September 30 was graphed in Rajagiriya area. During the last 15 hours, the humidity value has been varying between 94% and 95%. Most of the time it was at the 95% level. As such, the water concentration in the atmosphere in that area is very high.

4.2.3 Pressure Result for individual part

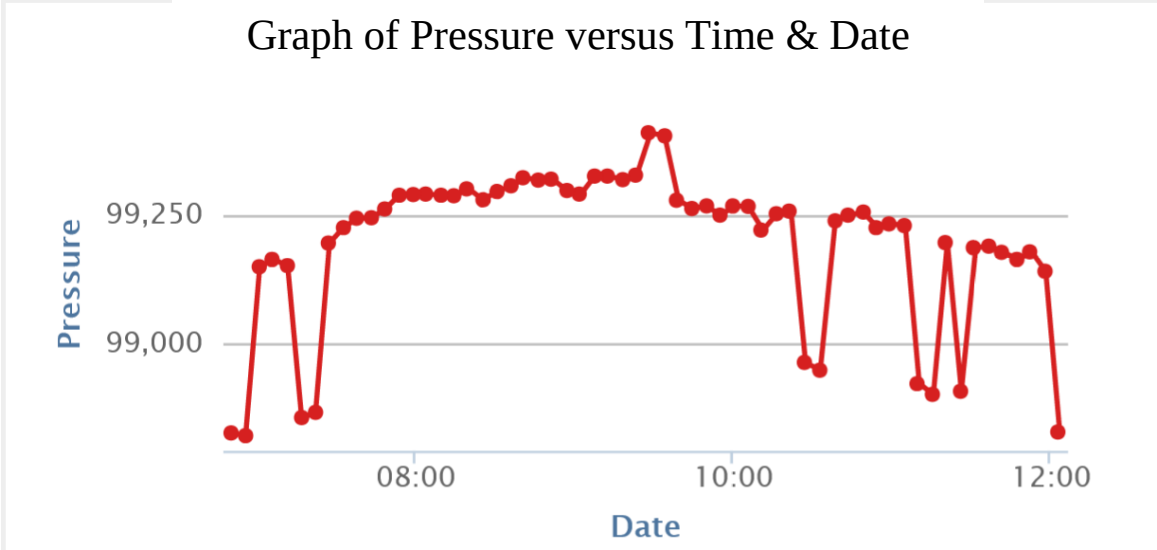


Figure 4-5: Figure of Pressure variation graph

According to the above picture, the data obtained from 7 am to 12 noon on September 30 in Rajagiriya area was graphed. During the last 5 hours, the pressure value has been varying between 98 750 Pa and 99 500 Pa. From 7.00 am to 8.00 am in the morning it was between 98 750 Pa and 99 250 Pa and from 8.00 to 11 most of the time it was between 99 250 Pa and 99 500 Pa. From 11.00 am to 12.00 am again it remained fluctuating between 99 250 Pa and 99 000 Pa.

4.3 Result of the group part of Weather Stations

In this section, in the group project, the temperature and humidity data values of ten difference areas were obtained and the data was analysed.

4.3.1 Temperature and Humidity result for Kiribathgoda Area

Figure 4-6: Figure of Temperature variation graph for Kiribathgoda area

According to the Figure 4-6, the temperature in Kiribathgoda area was around 27 ^{0}C from 3.00 to 3.30 in the morning, and it increased to 29.5 ^{0}C at 3.30. Again from 3.40 to 4.15 it has come down to 27 ^{0}C and by 4.30 the temperature has decreased to 26 ^{0}C. Then the temperature gradually increased until around 5.45 am and from 5.30 am to 8.30 am the temperature was varying around 29 ^{0}C.

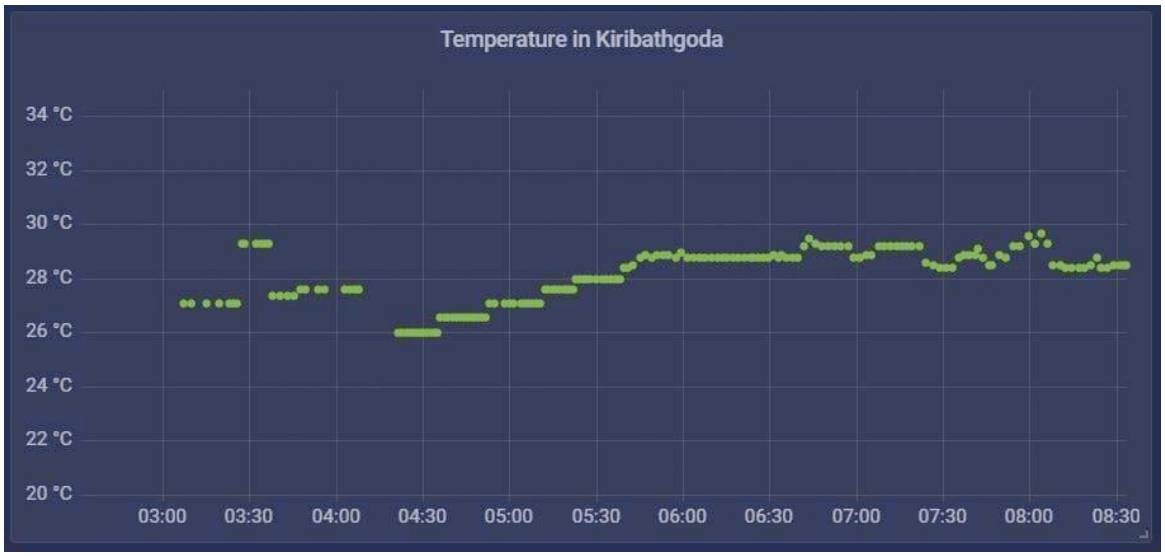


Figure 4-7: Figure of Humidity variation graph for Kiribathgoda area

According to the Figure 4-7, the humidity value in Kiribathgoda area from 3.00 am to 3.30 am is around 85%, and at 3.30 am it has reduced to 75%. Again from 3.40 to 4.15 it has increased to 85% and by 4.30 the humidity has decreased to 70%. After that, the humidity value has gradually increased till around 5.45 and from 5.30 to 8.30 the humidity value has been varying around 75%.

4.3.2 Temperature and Humidity result for Wijerama Area

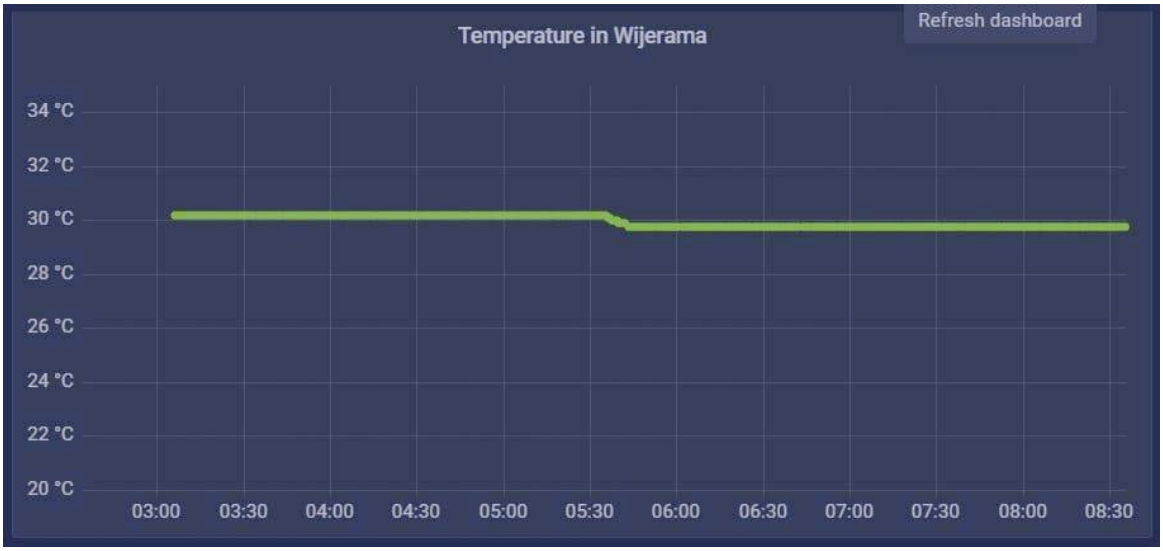


Figure 4-8: Figure of Temperature variation graph for Wijerama area

According to the picture above, the temperature in Wijerama area was around 30.1^{0}C from 3.00 am to 5.40 am, and it increased to 29.8 ^{0}C at 5.40 am and the temperature remained the same until around 8.30 am.

Figure 4-9: Figure of Humidity variation graph for Wijerama area

According to the above picture, the humidity value in Wijerama area was around 71% from 3.00 am to 5.40 am, and it increased to 72% at 5.40 am and the same humidity value remained until around 8.30 am.

4.3.3 Temperature and Humidity result for Gampaha Area

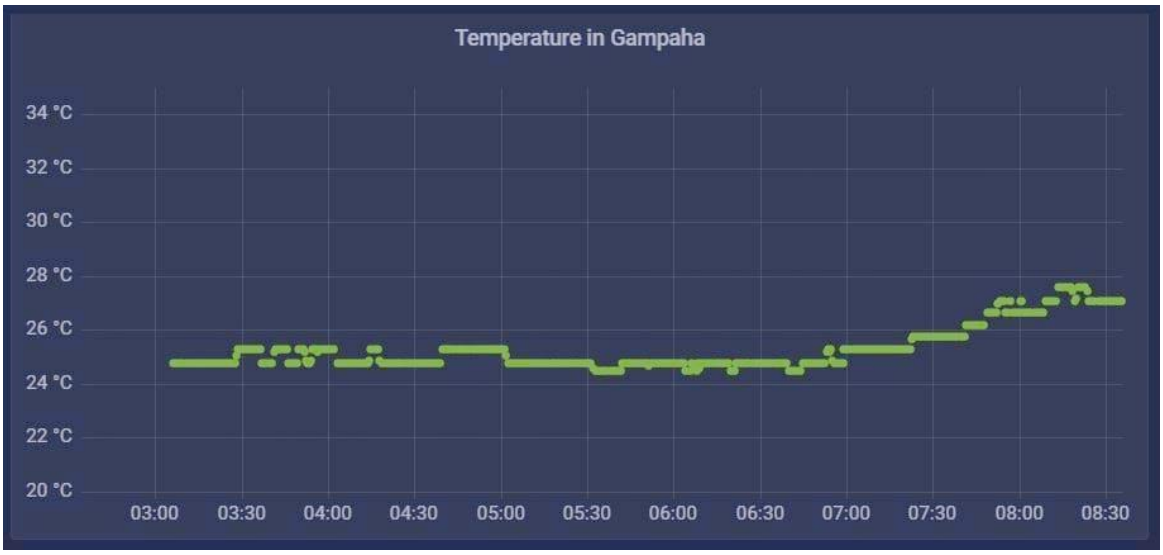


Figure 4-10: Figure of Temperature variation graph for Gampaha area

According to the above picture, the temperature in Gampaha area from 3.00 am to 7.30 am varied between 24 ^{0}C and 26 ^{0}C, and after 7.30 the value gradually increased and by 8.30 it was 28 ^{0}C.

Figure 4-11: Figure of Humidity variation graph for Gampaha area

According to the above picture, the humidity value in Gampaha area varied between 87% and 90% from 3.00 am to 7.45 am, after 7.45 am the value gradually decreased to 85% and at 8.15 am the humidity value decreased to 83%. Again by 8.30 there has been a slight increase to 85%.

4.3.4 Temperature and Humidity result for Rajagiriya Area

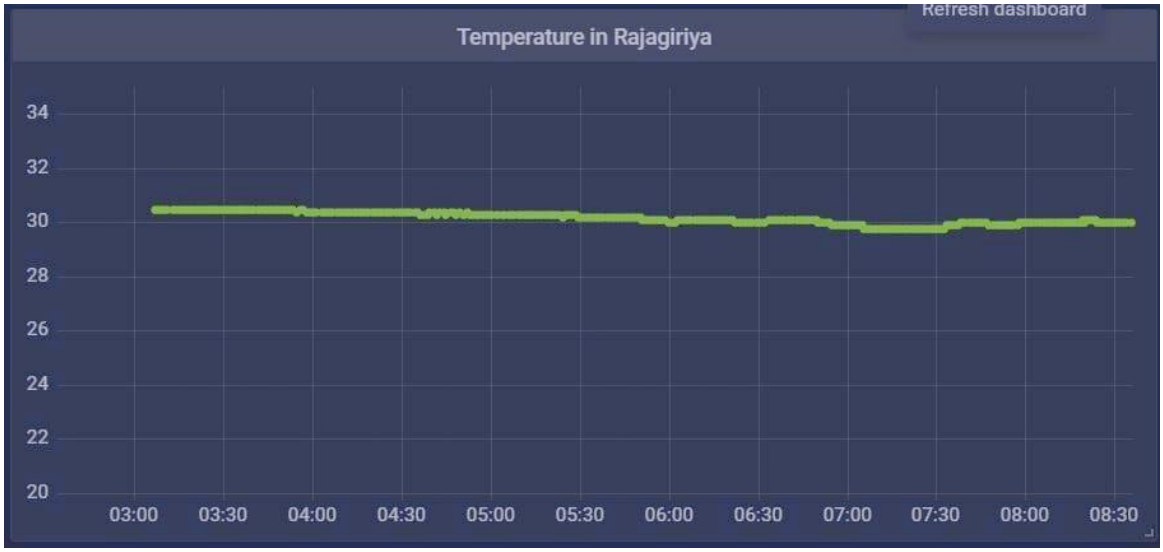


Figure 4-12: Figure of Temperature variation graph for Rajagiriya area

According to the above image, 31 ^{0}C was present in Rajagiriya area from 3.00 am to 5.00 am, showing a slight decrease after 5.00 am. By 7.30, the temperature had reached a minimum of 29.8 ^{0}C, and again the temperature rise slightly and by 8.30 it was at the limit of 30 ^{0}C. Temperatures have been around 30 ^{0}C throughout the entire period.

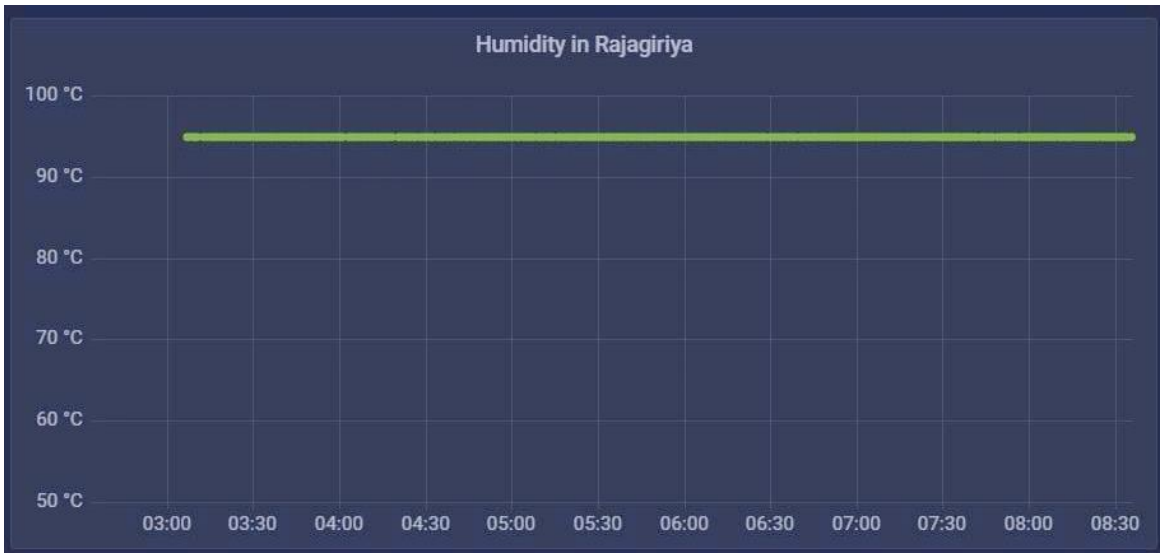


Figure 4-13: Figure of Humidity variation graph for Rajagiriya area

According to the above picture, the humidity value has remained at a constant level of 95% from 3.00 am to 8.30 am in Rajagiriya area. It implies that the concentration of water droplets in the air has been at the highest value during that period.

4.3.5 Temperature and Humidity result for Maradana Area

Figure 4-14: Figure of Temperature variation graph for Maradana area

According to the picture above, the temperature in Maradana area remained at a constant value of 29.3 ^{0}C from 3.00 am to 8.30 am.

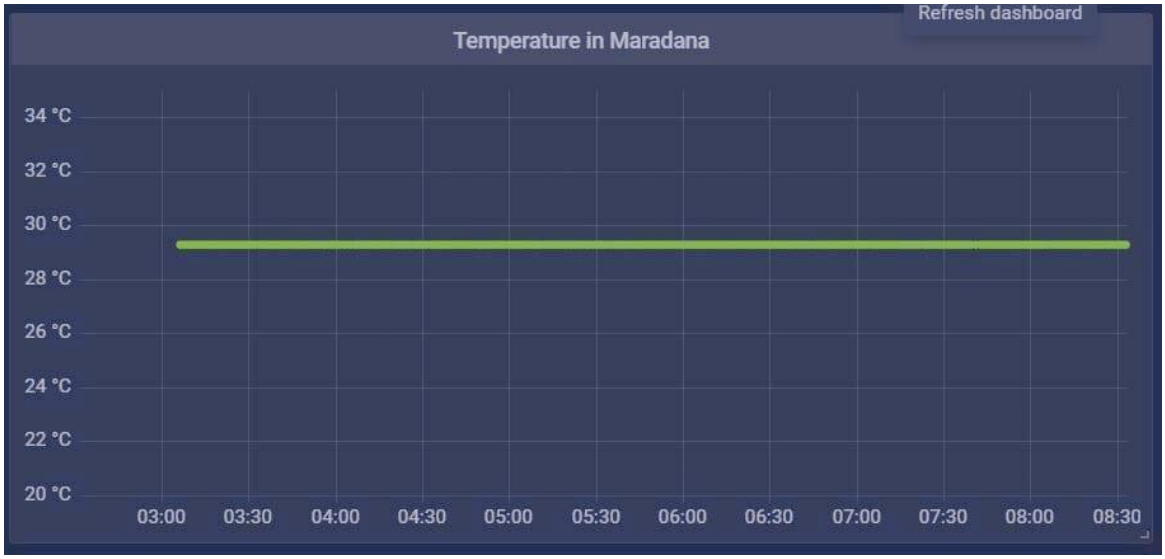


Figure 4-15: Figure of Humidity variation graph for Maradana area

According to the above picture, the humidity value has been varying between 75% and 80% in Maradana area from 3.00 am to 7.40 am. After that, the highest humidity value of 80% was recorded from 7.40 to 8.10. By 8.30 again, the humidity has reduced to 79%.

4.3.6 Temperature and Humidity result for Kuliyapitiya Area

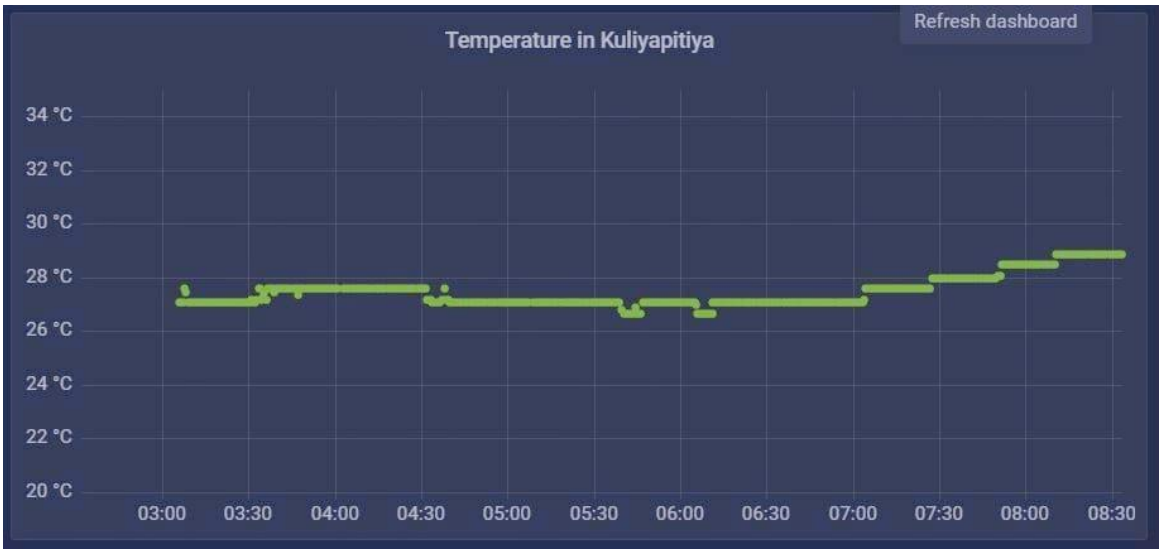


Figure 4-16: Figure of Temperature variation graph for Kuliyapitiya area

According to the picture above, the temperature in Kuliapitiya area has been varying between 26 ^{0}C and 28 ^{0}C from 3.00 am to 7.30 am. Then at around 7.30 the temperature in the area has been around 28 ^{0}C. After that the temperature gradually increased from 7.30 to 8.30 and by 8.30 the temperature reached 28.8 ^{0}C.

Figure 4-17: Figure of Humidity variation graph for Kuliyapitiya area

According to the above picture, the humidity value has been varying between 82% and 85% in Kuliapitiya area from 3.00 am to 4.45 am. Later, at around 4.45, the humidity value of the area has been around 85%. The humidity value has continued till 5.45 and around 5.45 the humidity value has reached 81%. Again from 6.00 to 6.40 the humidity value remained at 80% and at 6.40 again it increased to 81%. At 7.00, the highest humidity value was recorded as 84%. After 7.30, the humidity gradually decreased and by 8.30, the humidity was around 82%.

4.3.7 Temperature and Humidity result for Kadawatha Area

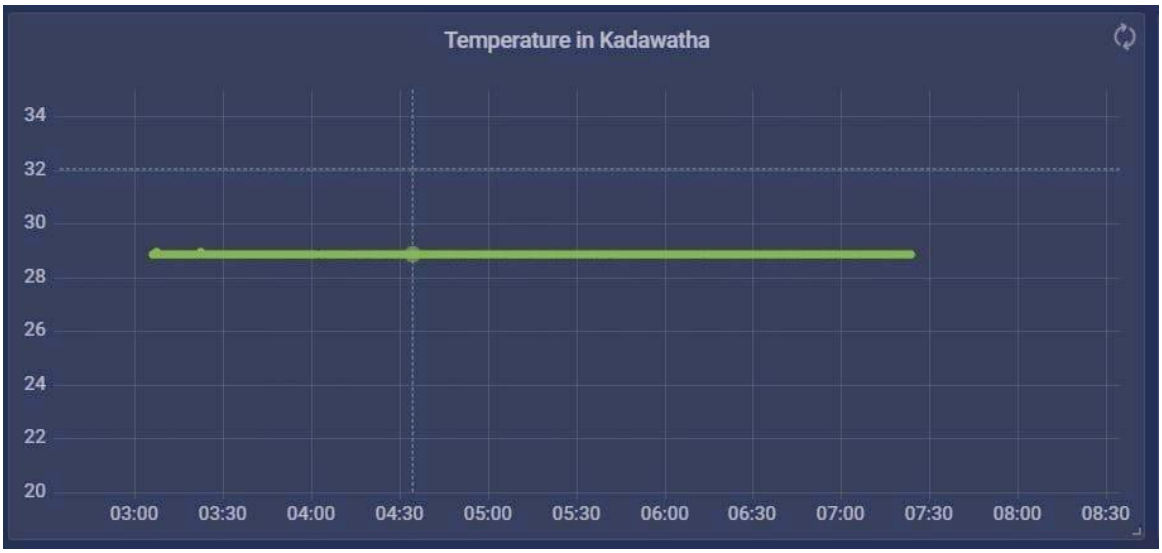


Figure 4-18: Figure of Temperature variation graph for Kadawatha area

According to the picture above, the temperature in Kadawatha area remained at a constant value of 29.3 ^{0}C from 3.00 am to 7.30 am. After 7.30 am, the data uploading was stopped.

Figure 4-19: Figure of Humidity variation graph for Kadawatha area

According to the above picture, the humidity value has been varying between 75% and 80% in Kadawatha area from 3.00 am to 7.40 am. After that, the highest humidity value of 80% was recorded from 7.40 to 8.10. By 8.30 again, the humidity has reduced to 79%.

4.3.8 Temperature and Humidity result for Melsiripura Area

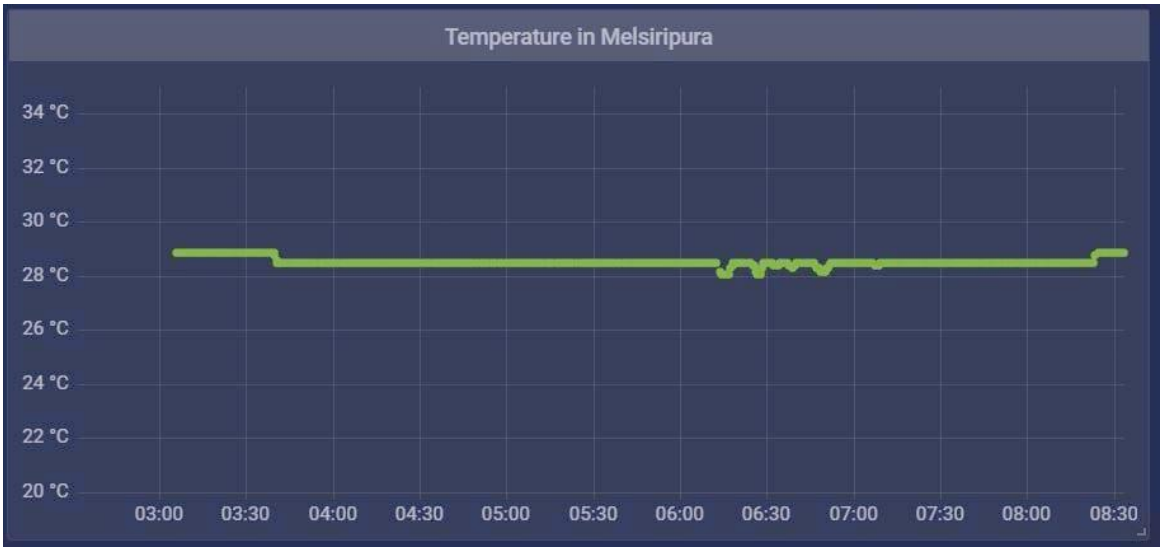


Figure 4-20: Figure of Temperature variation graph for Melsiripura area

According to the picture above, the temperature in Kuliapitiya area was 28.2 ^{0}C from 3.00 am to 3.40 am and 28.1 ^{0}C from 3.40 am to 6.15 am. Then till 6.50 the temperature fluctuated between 28.1^{0}C and 28 ^{0}C and again from 6.40 to 8.20 it remained at a constant value of 28.1. By 8.30 am the temperature has reached 28.2 ^{0}C again.

Figure 4-21: Figure of Humidity variation graph for Melsiripura area

According to the above picture, the humidity value in Melsiripura area was around 71% from 3.00 am to 6.00 am, and it increased to 72% at 5.50 am and the same humidity value remained until around 8.30 am.

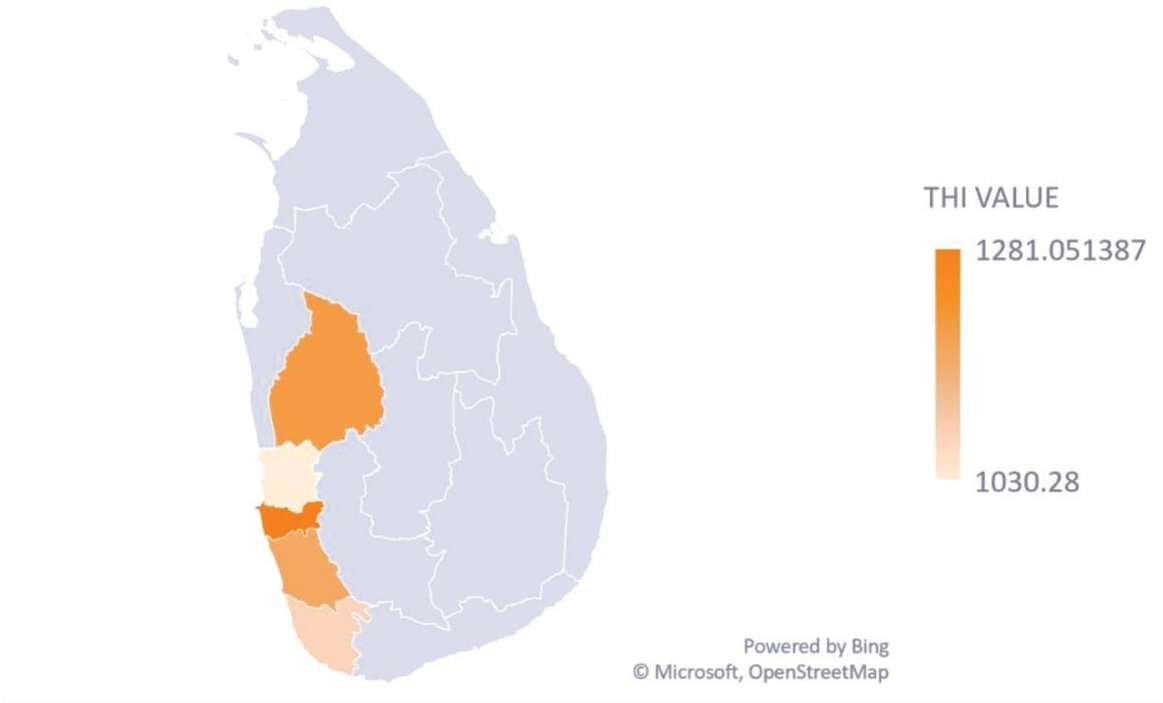


Figure 4-22: Figure of Temperature and Humidity index map for Sri Lanka (Source: Microsoft, OpenStreetMap)

According to figure 4-3, the five districts where temperature and humidity data were collected during the team project are shown. Those districts include Colombo, Gampaha, Kalutara, Galle and Kurunegala districts.

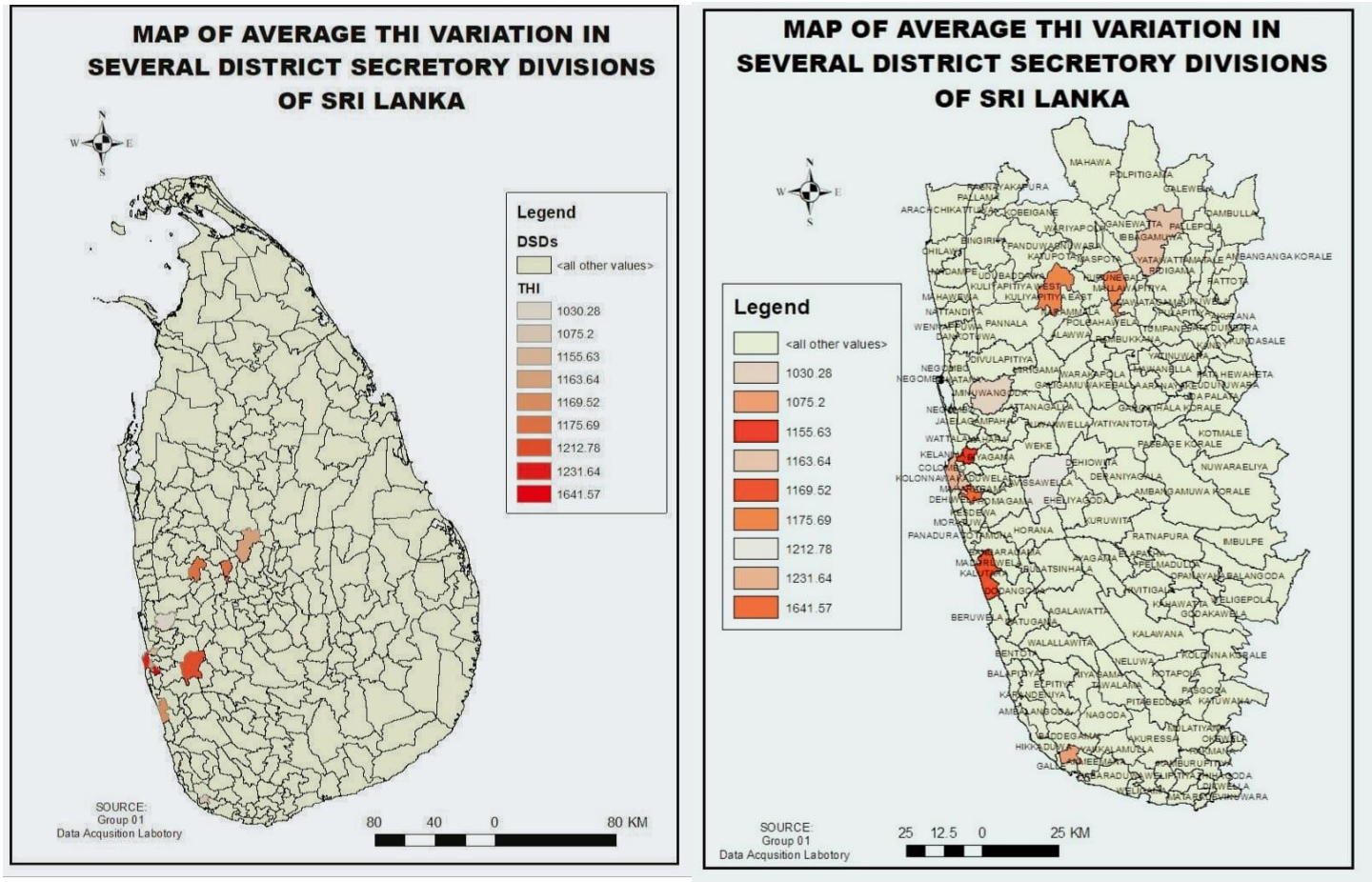


Figure 4-23: (a) Figure of map of THI variation in Sri Lanka, (b) Figure of map of THI variation in District Secretory divisions (Source: Microsoft, OpenStreetMap)

The map belonging to the district secretariats where the temperature and humidity data were collected during the team project is shown. There, Figure 4-4 (a) shows the relevant areas from the whole of Sri Lanka and Figure 4-4 (b) shows the relevant region in close proximity. These figures show the average Temperature Humidity Index (THI) values. The highest THI values were obtained from Maradana, Kalutara and Rajagiriya and their value is about 1641.57. The second highest values are reported from Kadavata, Kuliapitiya and Kurunegala areas and the value is around 1175. Somewhat minimum values were obtained from Kolonnawa, Ibbagamuwa areas. Its value is around 1163.64. The minimum value was obtained from Gampaha area and the value is around 1030.

5 DISCUSSION

In this project, the main hypothesis was that the sensor is more accurate. The “DHT-11” and "BMP180" sensors were supported for the hypothesis. When the several setups were placed near each other, the *temperature, humidity,* and *pressure* values among them were got as same values. But the “Rain Water Level” sensor wasn’t supported for the hypothesis. When the calibration, the various values with large different were gave by the sensor. The "LDR" sensor was supported for the hypothesis at day time. At night, the wrong values were given by the sensor due to the light rays of road lights and the home lights of the city area being affected by the sensor. Therefore, the hypothesis for the "LDR" sensor wasn't supported in the city area at night. The "Rain Drop" sensor was supported for the start of 'Light Rain'. But it was become 'Medium Rain' level although the speed of the rain did not change when time goes to up. Then it has become 'Heavy Rain' level as more time passes. Although light rain was received, over time the "Rain Drop" sensor board read as 'Medium Rain' due to the decrease in its resistance value due to the accumulation of more water drops. As the wind speed was got from the only analog value given from the motor drive without calibrating. The analog value of wind speed supported the hypothesis. But calibration was more difficult because it had a limited range of methods. A properly calibrated anemometer could not be found there and it could not be calibrated when travelling in a vehicle because the effect of the wind coming from outside could not be measured. The other hypothesis was the power-saving concept. The "Deep Sleep" mode was uploaded into the "ESP32" but it failed because the power was got by all other sensors at every time. Although the team project assumed that all data would be received at the same time slot, it was different time slots for different regions due to the weakness of internet communication speed in those areas.

Temperatures hovered around 30 degrees Celsius from 9 am to 10 pm. It was close to 31 ^{0}C in the afternoon and evening. It was around 29 ^{0}C from midnight to morning. And the humidity value has remained above 90 throughout the day. But according to Google weather forecast, the temperature and humidity values have changed at times. It can be mainly due to the location of the setup near the city and near the main road. The excessive sunlight received on the roads during the afternoon is emitted as heat during the night, thus the humidity and temperature values are recorded as high values. The light intensity value was approximate same according to the 'meteoblue' website during the afternoon and it changed during the night due to the light from the road and houses. A little rain was received and the "Water Level" sensor accuracy was reduced so slightly different values were shown. Although there was light rainfall, the "Water Drop" sensor first reported it as 'Light Rain' and after some time it was reported as 'Medium Rain'. The pressure value was equal to the value of Google weather forecast. Since the anemometer was not calibrated, its values could not be compared with the ‘Google Weather Forecast’. During the team project, temperature and humidity were obtained from ten different areas, and the data was graphed and mapped for each area. There, all the data is the same as Google Weather Forecast.

This project describes how anyone can easily and cheaply build a small weather station and how to get and analyse weather data. By getting the weather data in this way, the user can get a rough understanding of the weather and because the data is available on the internet, the people who need it can easily see the weather condition of that area. The data of temperature, humidity, air pressure, the light intensity of the sun, rain level and rain situation were got by in this project and sent to the “Google sheet” and “ThingSpeak” for analysing for easy understanding of the people.

If the Wi-Fi connection is disconnected for a long time, the weather data may be missed at times. Also, it was observed that even though the weather data was obtained in the nearest 5 minutes, there was no big difference between those time intervals. Therefore, in the case of Wi-Fi disconnection with a time interval of 5 minutes, the data is temporarily saved by the flash memory in the ESP32 only for a maximum period of 6 hours and 40 minutes. Therefore, by further increasing the deep sleep time, data can be saved for a longer time than without a Wi-Fi connection. Also, a delay time circuit can be made and connected by an NPN transistor to power up the sensors only when needed. This allows the weather station to run on battery for a longer period of time. Also, if the anemometer can be calibrated using the correct equipment, the wind speed can also be obtained. Also, the support stand for the installation of this circuit fell to the ground when a strong wind. Therefore, it can be prevented by placing a heavy material (stone) on its bottom or by increasing the length of the legs and fixing the legs in four directions. Also, the pressure sensor was the most accurate sensor module that could measure temperature, even sea level height.

6 CONCLUSION

A day if the day will seem like a rainy day, then we can plan to do work inside. Otherwise, it will be like a sunny day, and then we can go outside and do our work. Therefore people should use the weather information to become life easier. If someone needs to know about the weather forecast, must collect weather data. Therefore, a weather data acquisition system should be developed. A data base system need for save weather data and a user-friendly interface must need for presenting the weather information. Therefore, the weather data acquisition system, data saving data base system and user-friendly interface was developed throughout this project.

A weather forecast is a statement saying what the weather will be like the next day or for the next few days. It is using the analysis of temperature, wind speed, wind direction, humidity, air pressure and rain of the previous day and then gives the weather forecast for the next days. In this project, a large weather station was made using several of single mini weather stations and then the data was collected using mini weather stations. The DHT 11 temperature sensor gave the data of temperature and humidity, the Rain Drop and Water level sensor gave the data of rain, the BMP180 sensor gave the data of air pressure, the LDR sensor gave the data of sunlight and the ESP 32 was used and sent the data to a google sheet. The data of several single mini weather stations send to the google sheet like this. Then the data analyst can analyse the data and give a statement about the weather forecast. Finally, “ThingSpeak” was used for analysing the forecast data.

Here, it is easy to plan the day's work by using Temperature, Humidity and Rain situation data among the weather data. If the humidity value is high, it is clear that the water vapour concentration in the surrounding air is high. Hence there is more tendency to sweat on such days. This is because the ratio of water droplets in the surrounding environment is high, so the sweat does not evaporate. Also, if the temperature is high or there is cold weather, a clothing plan can be prepared based on that information. Also, in case of rainy conditions, the day's work can be started with advance preparation. And if the sun is in a high period, different cream can also be chosen to protect from the sun.

Generally, many websites that offer weather forecasts analyse previous data and issue weather forecasts for the next few hours or days. There they carry out this process by storing the data they have acquired for a long time. Furthermore, various businesses also display different products from time to time based on this weather data. Examples include sun cream and cold coats.

Therefore, if several small-scale weather stations can be set up, to obtain data and present weather information related to a certain area, that data and information about weather will bring great convenience to the residents of that area and also to the tourists who come to visit that area. And the data from these small-scale weather stations is more valuable than weather information from other websites. This is because the data obtained from more data collection centres covering the area is more appropriate and accurate. As such, these small-sized weather stations provide better weather information to the public.

7 WORKS CITED

components101, 2019. *components101*. [Online] Available at: www.components101.com [Accessed 17 09 2022].

Electronics, M., 2022. *Mouser*. [Online] Available at: www.mouser.com [Accessed 13 09 2022].

lastminuteengineers.com, 2022. *Last Minute Egineers.* [Online] Available at: www.lastminuteengineers.com [Accessed 15 09 2022].

Instruments, N., 2022. *NI.* [Online] Available at: https://www.ni.com/en-us/shop/labview.html [Accessed 1 8 2022].

Oxford, 2019. *encyclopedia.* [Online] Available at: www.encyclopedia.com [Accessed 1 8 2022].

APPENDIX

- Link of Google Sheet :

https://docs.google.com/spreadsheets/d/1Nv0p9m9BLif_gv8G9vCqd4lrwkLNEH9ZUqZ0RlRUUvU/edit?usp=sharing

- Link of ThingSpeak Channel feed

https://api.thingspeak.com/channels/1872622/feeds.json?results=2

- Link of ThingSpeak Channel field

https://api.thingspeak.com/channels/1872622/fields/1.json?results=2

- The Code uploaded to the ESP-32

```
#include "DHT.h"          //Library for DHT11 Tempersture and
humidty Sensor
#include "WiFi.h"         //Library for WiFi
#include <HTTPClient.h>   //Library for WiFi
#include <Wire.h>         //Library for BMP180
#include <Adafruit_BMP085.h>  //Library for BMP180
#include "ThingSpeak.h"
#include "SPIFFS.h"     //library for save data local memory
#include <vector>        //library for save data local memory

#define DHT11PIN 4      //Define pin 4 is DHT11 signal input pin
#define rainAnalogPin 35  //Define pin 35 is rainAnalog signal input
pin
#define rainDigitalPin 34   //Define pin 34 is rainDig signal input
pin
#define rainwaterLevelPin 32    //Define pin 32 is rainwater level
measure signal input pin
#define LDRPIN 33     //Define pin 33 is LDR signal input pin
#define uS_TO_S_FACTOR 1000000  // Conversion factor for micro
seconds to seconds
#define TIME_TO_SLEEP  300        // Time ESP32 will go to sleep (in
seconds)

RTC_DATA_ATTR int bootCount = 0;

// WiFi DETAILS
const char* ssid = "Redmi Note 8";          // wifi SSID
const char* password = "12345678";    // Wifi password
```

```
29
30 // Google script ID
31 String GOOGLE_SCRIPT_ID = "AKfycby87NBaa55-
32 8paHMnfRA_YWFyuLzbf4VDs8OP_9cso9bg-WaD07GC3nxFgDHu3xUkV2nw";     //
33 Google shwwt id
34 unsigned long thingsPeakID = 1872622; // thingSpeak id
35
36 using namespace std; //"SPIFFS.h library
37
38 const char *writeAPIKey = "IGGS23FWHNN29TU3"; // thingSpeak API
39 WiFiClient client;
40
41 int count = 0;
42
43 DHT dht(DHT11PIN, DHT11);
44 Adafruit_BMP085 bmp;
45
46 void print_wakeup_reason(){
47   esp_sleep_wakeup_cause_t wakeup_reason;
48
49   wakeup_reason = esp_sleep_get_wakeup_cause();
50
51   switch(wakeup_reason)
52   {
53     case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by
54 external signal using RTC_IO"); break;
55     case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by
56 external signal using RTC_CNTL"); break;
57     case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by
58 timer"); break;
59     case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused
60 by touchpad"); break;
61     case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP
62 program"); break;
63     default : Serial.printf("Wakeup was not caused by deep sleep:
64 %d\n",wakeup_reason); break;
65   }
66 }
67 void usedcharactors(){
68
69   }
70 void listAllFiles()
71 {
72   File root = SPIFFS.open("/");
73   File file = root.openNextFile();
74   while (file) {
75     Serial.print("FILE: ");
76     Serial.println(file.name());
```

```
    file = root.openNextFile();
  }
}

void setup() {

  delay(1000);
  Serial.begin(115200);
  delay(1000);

  SPIFFS.begin(true);
  File tempfile;
  File humidityfile;


  dht.begin();  //Start DHT11

  char i = 0;
  char j = 0;
  char k = 0;
  char g = 0;
  char h = 0;
  char t=0;
  char StartupEmptypoint;
  char n = 80; //offline recorded array lenth
  char WifiConnectWaitingTime=20;
  //offline log arrays
  String temparray[n];
  String humidarray[n];
  String pressurearray[n];
  String ldrarray[n];

  delay(1000);

 if (!bmp.begin()) {
    Serial.println("Could not find the BMP180 sensor, check
connections!!!");
    while (1) {}
  }
   //Rain Sensor
   pinMode(rainDigitalPin, INPUT);
   pinMode(rainwaterLevelPin, INPUT);

    // connect to WiFi
    Serial.println();
    Serial.print("Connecting to wifi: ");
    Serial.println(ssid);
    Serial.flush();
```

```
125     WiFi.begin(ssid, password);
126 
127         float dhtHumidity = dht.readHumidity();
128         float dhtTemperature = dht.readTemperature();
129         float pressure = bmp.readPressure();
130         int analogValueLDR = analogRead(LDRPIN);
131 
132     while (WiFi.status() != WL_CONNECTED) {
133       delay(500);
134       Serial.print(".");
135 
136       k++;
137       if (k > WifiConnectWaitingTime) { // if waiting time
138         Serial.println("\ndata recording Offline...");
139         if(!(SPIFFS.exists("/tempc.txt")) &&
140 !(SPIFFS.exists("/humidity.txt")) &&
141 !(SPIFFS.exists("/pressure.txt")) && !(SPIFFS.exists("/ldr.txt"))){
142           File tempfile = SPIFFS.open("/tempc.txt", FILE_WRITE);
143           File humidityfile = SPIFFS.open("/humidity.txt",
144 FILE_WRITE);
145           File pressfile = SPIFFS.open("/pressure.txt", FILE_WRITE);
146           File ldrfile = SPIFFS.open("/ldr.txt", FILE_WRITE);
147 
148           Serial.println("file write in write mode");
149           // log to fiel
150           tempfile.println(String(dhtTemperature));
151           humidityfile.println(String(dhtHumidity));
152           pressfile.println(String(pressure));
153          ldrfile.println(String(analogValueLDR));
154 
155           tempfile.close();
156           humidityfile.close();
157           pressfile.close();
158           ldrfile.close();
159           }else{
160             File tempfilesecond = SPIFFS.open("/tempc.txt","a" );
161             File humidityfilesecond = SPIFFS.open("/humidity.txt",
162 "a");
163             File pressfilesecond = SPIFFS.open("/pressure.txt","a"
164 );
165             File ldrfilesecond = SPIFFS.open("/ldr.txt", "a");
166             tempfilesecond.println(String(dhtTemperature));
167             humidityfilesecond.println(String(dhtHumidity));
168             pressfilesecond.println(String(pressure));
169             ldrfilesecond.println(String(analogValueLDR));
170             Serial.println("file write in append mode");
171             tempfilesecond.close();
172             humidityfilesecond.close();
```

```
173             pressfilesecond.close();
174             ldrfilesecond.close();
175           }
176       k = 0;
177       break;// when process is done leave while() loop
178       }
179     }
180    ThingSpeak.begin(client); //assign the wifi into thingspeak
181
182        //Increment boot number and print it every reboot
183   ++bootCount;
184   Serial.println("Boot number: " + String(bootCount));
185
186   //Print the wakeup reason for ESP32
187   print_wakeup_reason();
188
189   esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
190   Serial.println("Setup ESP32 to sleep for every " +
191 String(TIME_TO_SLEEP) +
192   " Seconds");
193
194
195 }
196
197 void loop() {
198
199   char i = 0;
200   char j = 0;
201   char k = 0;
202   char g = 0;
203   char h = 0;
204   char t=0;
205   char StartupEmptypoint;
206   char n = 80; //offline recorded array length
207   char WifiConnectWaitingTime=25;
208   //offline log arrays
209   String temparray[n];
210   String humidarray[n];
211   String pressarray[n];
212   String ldrarray[n];
213
214   float dhtHumidity = dht.readHumidity();
215   float dhtTemperature = dht.readTemperature();
216   float pressure = bmp.readPressure();
217   int analogValueLDR = analogRead(LDRPIN);
218   int rainAnalogVal = analogRead(rainAnalogPin);
219   int rainDigitalVal = digitalRead(rainDigitalPin);
220   float rainwaterLevelOld = analogRead(rainwaterLevelPin);
```

```
221
222    if(isnan(dhtHumidity) || isnan(dhtTemperature)){
223     Serial.println(F("Failed read DHT11 sensor!!!"));
224     return;
225     }
226
227   //calibrated equation for Water Level messurement. EQ; y = 74.182x
228 + 84.431;
229   float rainwaterLevel = (74.182*(rainwaterLevelOld)) + 84.431;
230
231   //calibrated equation for analog to lux converter for LDR. EQ; y'
232 = 0.00010213x^2 - 0.4717x + 590.3023;
233   float luxvalue = (0.00010213*(analogValueLDR*analogValueLDR))-
234 (0.4717*analogValueLDR)+590.3023;
235
236   ThingSpeak.setField(1,dhtTemperature);
237   ThingSpeak.setField(2,dhtHumidity);
238   ThingSpeak.setField(3,pressure);
239   ThingSpeak.setField(4,luxvalue);
240   int respons1 = ThingSpeak.writeFields(thingsPeakID,writeAPIKey);
241
242
243
244
245      if (WiFi.status() == WL_CONNECTED) {
246         HTTPClient http;
247
248             //if there exists spiff files they should upload first
249 to the google sheet
250         if ((SPIFFS.exists("/tempc.txt")) &&
251 (SPIFFS.exists("/humidity.txt"))) {
252
253           File tempfile1 = SPIFFS.open("/tempc.txt");
254           File humidityfile1 = SPIFFS.open("/humidity.txt");
255           File pressfile1 = SPIFFS.open("/pressure.txt");
256           File ldrfile1 = SPIFFS.open("/ldr.txt");
257           vector<String> v1;
258           vector<String> v2;
259           vector<String> v3;
260           vector<String> v4;
261
262           while (tempfile1.available()) {
263             v1.push_back(tempfile1.readStringUntil('\n'));
264           }
265
266           while (humidityfile1.available()) {
267             v2.push_back(humidityfile1.readStringUntil('\n'));
268           }
```

```
269           while (pressfile1.available()) {
270             v3.push_back(pressfile1.readStringUntil('\n'));
271           }
272 
273           while (ldrfile1.available()) {
274             v4.push_back(ldrfile1.readStringUntil('\n'));
275           }
276 
277           tempfile1.close();
278           humidityfile1.close();
279           pressfile1.close();
280           ldrfile1.close();
281     //retrieve log data to array s
282           for (String s1 : v1) {
283             temparray[i] = s1;
284             i++;
285           }
286 
287           for (String s2 : v2) {
288             humidarray[j] = s2;
289             j++;
290           }
291 
292           for (String s3 : v3) {
293             pressarray[g] = s3;
294             g++;
295           }
296 
297           for (String s4 : v4) {
298             ldrarray[h] = s4;
299           }
300           while (t <= n) {
301             if(temparray[t]==0 && humidarray[t]==0 &&
302 pressarray[t]==0 && ldrarray[t]==0){
303                 Serial.println("\nArrays are empty....");
304                 StartupEmptypoint=t;
305                 break;
306               }else{
307                 if (WiFi.status() == WL_CONNECTED) {
308                   HTTPClient http;
309                   Serial.println("offline data uploading... ");
310                   Serial.print("temp :");
311                   Serial.print(temparray[t]);
312                   Serial.print("\t");
313                   Serial.print("humid :");
314                   Serial.print(humidarray[t]);
315                   Serial.print("\n");
316                   Serial.print("Press :");
```

```
317                   Serial.print(pressarray[t]);
318                   Serial.print("\t");
319                   Serial.print("ldr :");
320                   Serial.print(ldrarray[t]);
321                   Serial.print("\n");
322                   String url = "https://script.google.com/macros/s/"
323 + GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(temparray[t])
324 + "&Humidity=" + String(humidarray[t])+"&Pressure=" +
325 String(pressarray[t]) + "&LDR=" + String(ldrarray[t]);
326                   // httprequest(String(tempC), String(humi),
327 GOOGLE_SCRIPT_ID);
328                   Serial.println("Making a request");
329                   http.begin(url.c_str());
330
331 http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
332                   int httpCode = http.GET();
333                   String payload;
334                   if (httpCode > 0) { //Check for the returning code
335                     payload = http.getString();
336                     Serial.println(httpCode);
337                     Serial.println(payload);
338                   }
339                   else {
340                     Serial.println("Error on HTTP request");
341                   }
342                   http.end();
343                 }
344
345               t++;
346
347
348               }
349           }
350           if (t == n || t==StartupEmptypoint) {//remove files
351               Serial.println("Before remove");
352               listAllFiles();
353               SPIFFS.remove("/tempc.txt");
354               SPIFFS.remove("/humidity.txt");
355               Serial.println("After remove");
356               listAllFiles();
357               i = 0;
358               j = 0;
359             }
360         }else{
361
362             if(rainAnalogVal<=4000){
363               Serial.print("Temperature: ");
364               Serial.print(dhtTemperature);
```

```
365               Serial.println("°C ");
366               Serial.print("Humidity: ");
367               Serial.println(dhtHumidity);
368               Serial.print("Pressure = ");
369               Serial.print(pressure);
370               Serial.println(" Pa");
371               Serial.print("Lux Value = "); // LDR
372               Serial.println(luxvalue);
373               Serial.print("Rain Level: ");
374               Serial.println(rainwaterLevel);
375               delay(100);
376               String url = "https://script.google.com/macros/s/" +
377 GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(dhtTemperature)
378 + "&Humidity=" + String(dhtHumidity)+ "&Pressure=" +
379 String(pressure)+"&LDR=" + String(luxvalue)+ "&WDIr=" +
380 String(rainwaterLevel)+ "&Rain=Light_Rain";
381               Serial.println("Making a request");
382               http.begin(url.c_str());
383
384 http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
385               int httpCode = http.GET();
386               String payload;
387               if (httpCode > 0) {
388                 payload = http.getString();
389                 Serial.println(httpCode);
390                 Serial.println(payload);
391               }
392               else {
393                 Serial.println("Error on HTTP request");
394               }
395               http.end();
396             }
397
398
399           else if(rainAnalogVal<=2000){
400               Serial.print("Temperature: ");
401               Serial.print(dhtTemperature);
402               Serial.println("°C ");
403               Serial.print("Humidity: ");
404               Serial.println(dhtHumidity);
405               Serial.print("Pressure = ");
406               Serial.print(pressure);
407               Serial.println(" Pa");
408               Serial.print("Lux Value = "); // LDR
409               Serial.println(luxvalue);
410               Serial.print("Rain Level: ");
411               Serial.println(rainwaterLevel);
412               delay(100);
```

```
413               String url = "https://script.google.com/macros/s/" +
414 GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(dhtTemperature)
415 + "&Humidity=" + String(dhtHumidity)+ "&Pressure=" +
416 String(pressure)+"&LDR=" + String(luxvalue)+ "&WDIr=" +
417 String(rainwaterLevel)+ "&Rain=Midium_Rain";
418               Serial.println("Making a request");
419               http.begin(url.c_str()); //Specify the URL and
420 certificate
421
422 http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
423               int httpCode = http.GET();
424               String payload;
425               if (httpCode > 0) { //Check for the returning code
426                 payload = http.getString();
427                 Serial.println(httpCode);
428                 Serial.println(payload);
429               }
430               else {
431                 Serial.println("Error on HTTP request");
432               }
433               http.end();
434             }
435
436
437           else if(rainAnalogVal<=3500){
438               Serial.print("Temperature: ");
439               Serial.print(dhtTemperature);
440               Serial.println("°C ");
441               Serial.print("Humidity: ");
442               Serial.println(dhtHumidity);
443               Serial.print("Pressure = ");
444               Serial.print(pressure);
445               Serial.println(" Pa");
446               Serial.print("Lux Value = "); // LDR
447               Serial.println(luxvalue);
448               Serial.print("Rain Level: ");
449               Serial.println(rainwaterLevel);
450               delay(100);
451               String url = "https://script.google.com/macros/s/" +
452 GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(dhtTemperature)
453 + "&Humidity=" + String(dhtHumidity)+ "&Pressure=" +
454 String(pressure)+"&LDR=" + String(luxvalue)+ "&WDIr=" +
455 String(rainwaterLevel)+ "&Rain=Heavy_Rain";
456               Serial.println("Making a request");
457               http.begin(url.c_str()); //Specify the URL and
458 certificate
459
460 http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
```

```
461               int httpCode = http.GET();
462               String payload;
463               if (httpCode > 0) { //Check for the returning code
464                 payload = http.getString();
465                 Serial.println(httpCode);
466                 Serial.println(payload);
467               }
468               else {
469                 Serial.println("Error on HTTP request");
                  }
                  http.end();
                }


              else{
                  Serial.print("Temperature: ");
                  Serial.print(dhtTemperature);
                  Serial.println("°C ");
                  Serial.print("Humidity: ");
                  Serial.println(dhtHumidity);
                  Serial.print("Pressure = ");
                  Serial.print(pressure);
                  Serial.println(" Pa");
                  Serial.print("Lux Value = "); // LDR
                  Serial.println(luxvalue);
                  Serial.print("Rain Level: ");
                  Serial.println(rainwaterLevel);
                  delay(100);
                  String url = "https://script.google.com/macros/s/" +
GOOGLE_SCRIPT_ID + "/exec?"+"Temperature=" + String(dhtTemperature)
+ "&Humidity=" + String(dhtHumidity)+ "&Pressure=" +
String(pressure)+"&LDR=" + String(luxvalue)+ "&WDIr=" +
String(rainwaterLevel)+ "&Rain=No_Rain";
                  Serial.println("Making a request");
                  http.begin(url.c_str()); //Specify the URL and
certificate

http.setFollowRedirects(HTTPC_STRICT_FOLLOW_REDIRECTS);
                  int httpCode = http.GET();
                  String payload;
                  if (httpCode > 0) { //Check for the returning code
                    payload = http.getString();
                    Serial.println(httpCode);
                    Serial.println(payload);
                  }
                  else {
                    Serial.println("Error on HTTP request");
                  }
```

```
            http.end();
          }
        }
    }
  //count++;
     WiFi.disconnect();

     Serial.println("WiFi Disconnectiong....");
     delay(1000);

    Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}
```