Weather and Environment Monitoring System

Emerging Technologies Final Project

02/19/2024

by

Abdurrahman Aliyu Gambo

191203024

CPE - 541 Emerging Technologies in Computer Engineering

Department of Computer Engineering

Nile University of Nigeria

Contents

1 Introduction

1.1 IoT an Emerging Technology

The Internet of Things (IoT) is an emerging technology that provides heightened communication between devices through the utilization of inter-networking based protocols and technologies. IoT revolutionizes the world through the its ability to seamlessly connect many devices together, facilitating unparalleled communication between even low-powered edge devices in a manner that was previously not possible with regards to scale, flexibility and universal accessibility. IoT continues to transform the world providing access to edge device creation and data collection that works together with Big Data and Artificial Intelligence to create smarter, more interconnected, and more useful systems and devices. IoT allows everyday objects to connect to the internet gather and transmit data, as well as receiving commands and acting upon them (actuators). This project aims to utilize the principles of IoT, and existing frameworks/technologies to create a weather and environmental monitoring system. Data will be collected through sensors, aggregated, and forwarded to a cloud based host which will present the data in an easily accessible dashboard.

1.2 Technological Background

IoT blends staggering amounts of information from various nodess and devices together, naturally this has required particularly performant and adequate protocols. MQTT which stands for Message Queue Telemetry Transport, is a widely used and extremely popular protocol in the IoT space. MQTT's major strengths come from its particularly light-weight publish-subscribe architecture. Making it extremely useful in low-resource scenarios, which for IoT devices is particularly common. MQTT is supported as it provides resource cognizant, lightweight performance, in a scalable and reliable solution [1].

There are however protocols of different levels heavily involved in the IoT space that are not MQTT. Some notable examples include:

- Bluetooth Low Energy (BLE): Short-range, low-power connectivity for wearables and smart home devices (Originated in Bluetooth 4.0) [2].
- LoRaWAN: Long-range, low-power network solution for geographically dispersed devices (provides an ideal alternative to LANs in less through-put demanding applications) [3].
- NB-IoT: A cellular network technology designed for massive machine-type communication.
- OPC UA: Industrial oriented communication protocol for machine-to-machine data exchange.
- Zigbee: A Mesh network protocol for building automation and industrial applications.

2 Methodology

2.1 Overview

The development of the weather system consists of three main phases. These phases are the Hardware Implementation, Firmware Implementation, and Cloud Host Data Visualization phases. The development of the project largely followed the phases in the aforementioned order chronologically.

2.1.1 Objective

Prior to the development of the system, key system requirements were specified. Namely the system must categorize and capture these key parameters.

Local Weather Data

- Relative Humidity
- Pressure (Hecto-Pascals)
- Temperature (Degrees Celsius)
- Wind Speed (miles or kilometers per hour)

Environmental Quality Data

- General Air Quality
- Carbon Monoxide Levels
- Smoke and Gas Levels

These parameters will be captured through the use of a variety of sensors which will be connected to a NodeMCU32S Microcontroller that will then capture and transmit the collected sensor data to a host utilizing the MQTT protocol over the internet through the use of the NodeMCU32S' onboard WiFi Module.

2.1.2 Components

The sensors used to capture the relevant data points are as follows:

- **BMP 180:** The BMP 180 is a multipurpose temperature, pressure, and altitude measurement sensor. It functionally performs the task or a barometer, thermometer, and altimeter in one package.
- **DHT 11:** The DHT 11 is a temperature and humidity sensor and is primarily responsible for the capture of humidity data as well as serving as a means of temperature reading redundancy alongside the BMP 180.

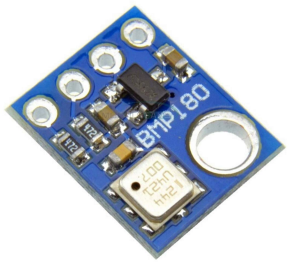

Figure 2.1: BMP-180 Pressure Sensor

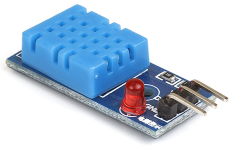

Figure 2.2: DHT11 Temperature and Humidity Sensor

- **MQ-2:** The MQ-2 is chemical gas sensor used to detect the presence of smoke, alcohol, hydrogen, as well as gaseous hydrocarbons such as LPG, i-butane, propane, and methane.

- **MQ-7:** The MQ-7 similar to the MQ-2 is also a chemical gas sensor, but differs in that it is used to monitor carbon monoxide levels instead.
- **MQ-135:** The MQ-135 is similar to the other MQ sensors with the exception that its primary focus is on the monitoring of the general air quality within an environment for pollutants such as ammonia, nitrogen, sulfur, and carbon dioxide, as well as smoke.

Figure 2.3: MQ2 Smoke and Gas

Figure 2.4: MQ7 Carbon Monoxide

Figure 2.5: MQ135 Air Quality

- **Anemometer:** The Anemometer used in this project was largely created out of a standard computer fan, functioning as a generator when connected to the anemometer body. When the anemometer spins an current is induced, the magnitude of the voltage of this signal is then passed to the microcontroller where it is extrapolated to arrive at the wind speed in m/s, km/h, mph. This is further discussed in Section 2.2.4.

Figure 2.6: Anemometer Top View

Figure 2.7: Anemometer Side Profile

- **NodeMCU32S:** The NodeMCU32S is an ESP-32 based microcontroller created by AI Thinker [4]. It features WiFi connectivity along with all the features of a typical ESP with regards to interfacing with devices. Crucial to this project are the ADCs for analog inputs, and the I2C connectivity to interface with the BMP180. Version 1.3 is used in this project.

2.2 Data Collection

2.2.1 Sensor Interfacing

The collection of sensory data achieved by hardware component of the system involved connected the respective sensors to the NodeMCU. For environmental monitoring the 3 gas based sensors were connected specifically to their analog output. This allowed the MCU to gather continuous and specific information as to the output levels. The alternative would be to use the digital output pins, though this would only provide

Figure 2.8: AI Thinker NodeMCU32S

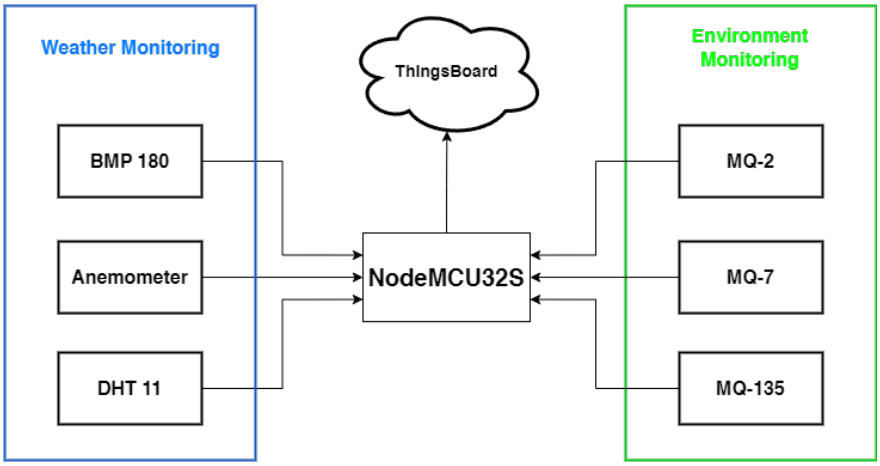

Figure 2.9: Weather and Environmental Monitoring System Block Diagram

a toggle as to whether or not the threshold has been reached, this threshold is controlled by a potentiometer on the back of the modules. As a result the analog approach proved superior as A) it provides constant data as to the precise level and B) Threshold limits can be implemented in software either at the level of the microcontroller, or even in the data aggregation stage/alarm stage at the cloud host side. A sample connection is shown in Figure 2.10.

ADC1 (GPIO Pins 32,33,34,35,36,39) was used as ADC2 often can not be used while WiFi is on [5]. The MQ2, MQ7, and MQ135 had their analog outputs connected to GPIO pins 35, 34, and 36 respectively. The DHT11 with its data line outputs to GPIO 17, whereas the BMP communicates to the MCU through I2S, with SCL, and SDA being on the default pins 22, and 21 respectively. Finally the fan inputs were used as the anemometer inputs (see Section 2.2.4). These were connected to pins 39 and 33.

2.2.2 Data Conversion

The nature of the communicated data, is done so with regards to the range of the ADC, in the NodeMCU this is a 12-bit ADC meaning the potential output value range is 2^{12} or 4096. this means that 0 to 4095 are values extracted across a range of 0 to 3.3V which is the logic level of ESP32s. For analog data particularly this means that the range from 0 to 4095 is the output from the sensors, and as such a necessary conversion from the raw readings to a standard unit is required. For pressure, humidity, and temperature readings gathered from the DHT11 and the BMP180 libraries tasked with interfacing with the sensors additionally provide the conversion to standard units (assuming the sensors do not already provide data in these units). As for the Environmental monitoring sensors, given the raw analog output, to convert the raw readings to usable values a conversion within the microcontroller firmware was necessary. The typical relation used to ascertain concentration values in PPM (parts per million) by a sensor like the MQ7 involves comparing the

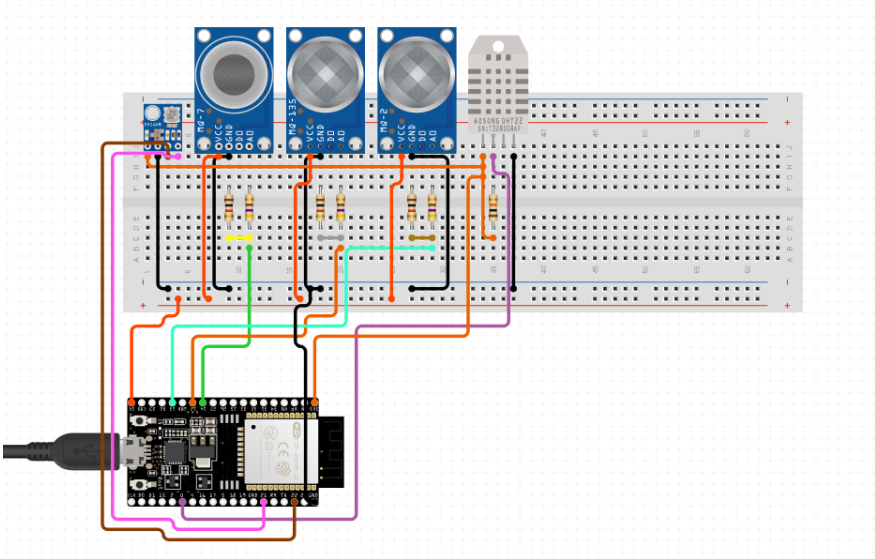

Figure 2.10: Hardware Sample Prototype Connection

voltage created by changes in the resistive load, where the ratio between the resistance R_0 under normal or "clean air" conditions and the current condition is utilized to quantify the concentration. A chart outlining the sensitivity of the MQ7 is shown in Figure 2.11.

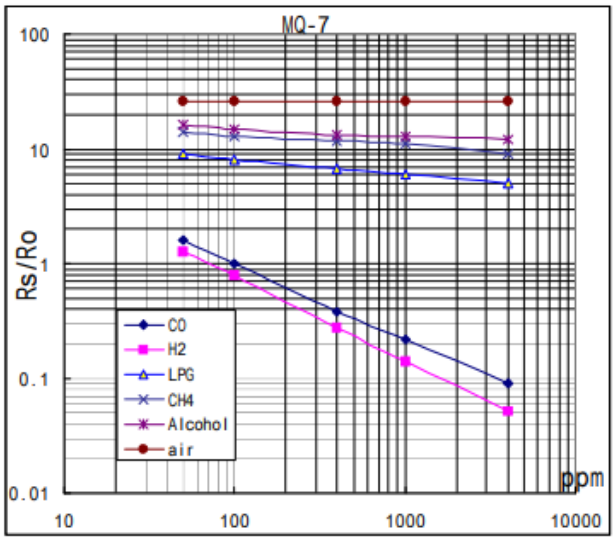

Figure 2.11: MQ7 Sensitivity Characteristics

From this data eventually a relationship between the sensor resistance, and the PPM can be established [6]. The relationship is shown in Equation 1.

$$ppm = (1538.46\frac{R_S}{R_2})^{-1.709} \tag{1}$$

2.2.3 Circuit Overview

The complete circuit diagram can be found in Figure 2.12. Of particular note is that the 3 Gas sensors, each are connected to a potentiometer a 4 pin output port, as well as a Texas Instruments LM393 which is a Dual

Channel Operational Amplifier. The LM393 amplifies the Gas Modules signal output before outputting it to the microcontroller. The circuit diagram for all 3 sensors as is seen in Figures 2.12 and 2.13, functionally outlines the construction of the Sensor Modules. The Anemometer in contrast is simply referred to with two pins, AnemHigh and AnemLow. The BMP 180 is connected to SDA and SCL, care is taken to pull up the two lines when no message is being sent (4.7kΩ resistors).

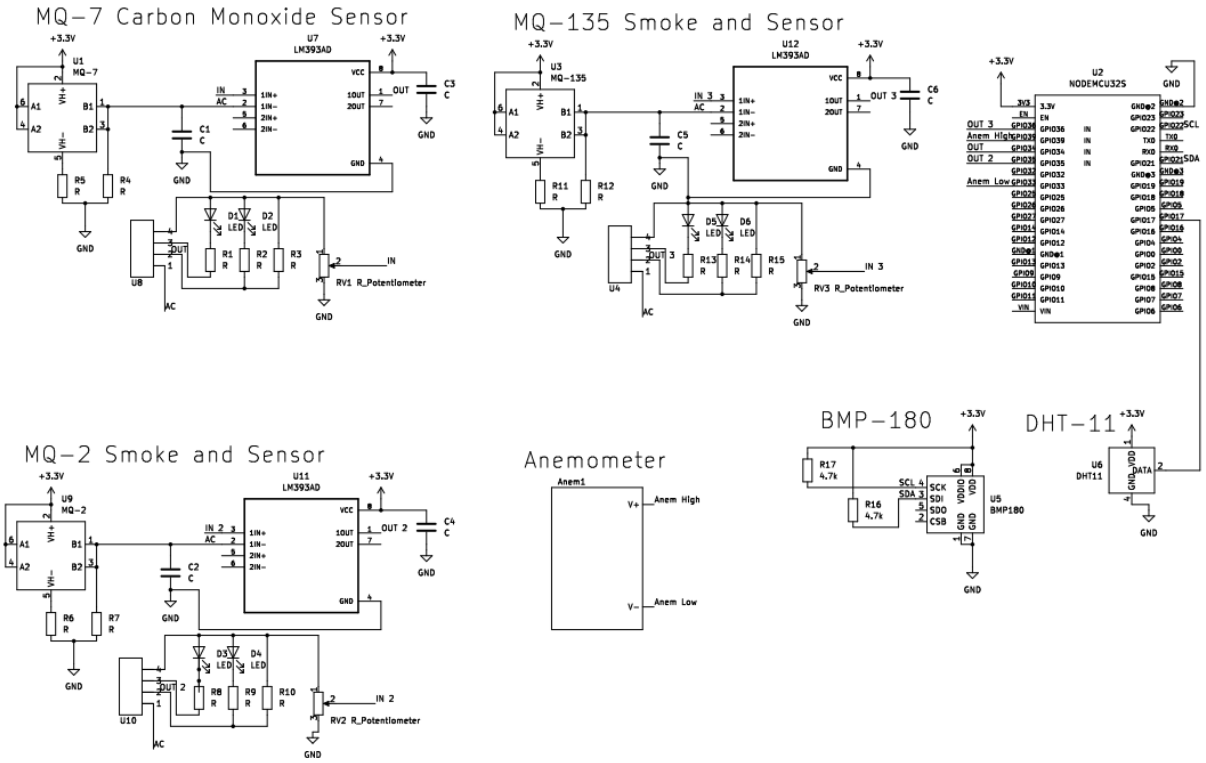

Figure 2.12: Weather and Environment Monitoring System Full Hardware Diagram

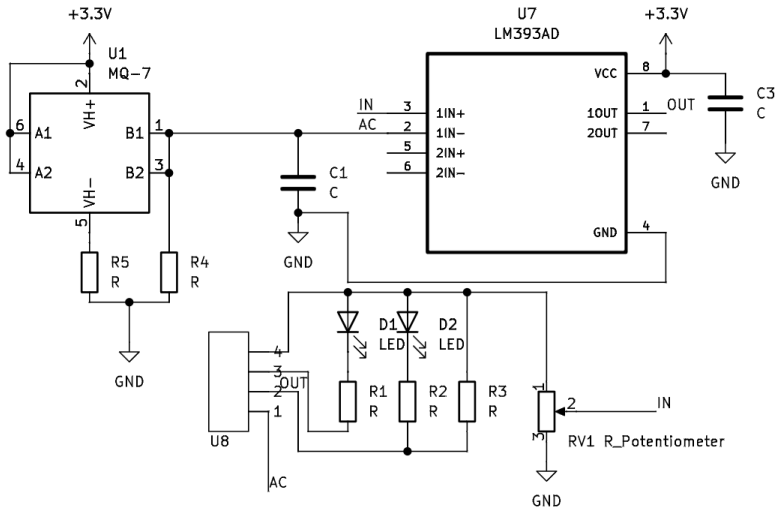

Figure 2.13: Gas Sensor Sub-Circuit Diagram

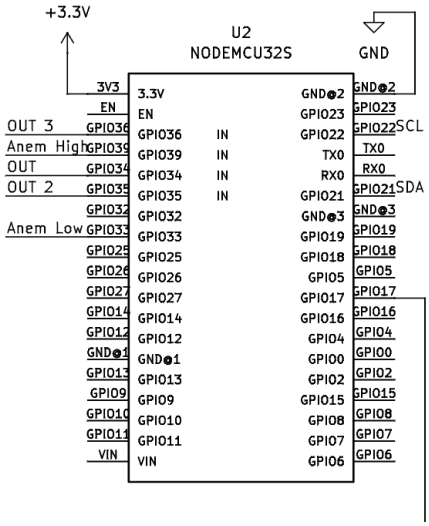

Figure 2.14: NodeMCU Sub-Circuit Diagram

2.2.4 Anemometer Creation

The anemometer was created from a low-power computer fan, as well as a fan like structure created from household materials. The working principle is based on electromagnetism. A computer fan is in essence a motor that converts electrical energy into mechanical energy [7]. The motor consists of windings and magnets that allow current to induce a magnetic field that causes the magnets (and by extension the fan blades to spin). By reversing this we arrive at the principle of a generator in which mechanical movement which creates a moving magnetic field, induces current in a coiled wire. As such by instead utilizing the fan inputs as output, we can expect feedback current from the motors [8]. Usually for most applications this is an undesired phenomena, but in this case it is leveraged to create the anemometer. Beyond simply gaining feedback current, it is important to be able to quantify the scale of the values read from the ADC GPIO pins connected to the anemometer. This was achieved through simple interpolation between two data points. Crucially to convert the raw readings into any measurement of speed (mph, kmph, or m/s), a means of gathering the rotations per time period was a necessity. Then by utilizing the length of the anemometer arms, the distance travelled can be determined. The anemometer was spun at a uniform speed over an certain interval between 10-15 seconds in which a slow motion video at 1/8 of real time speed, was used to capture a video of the spinning anemometer. This video was then later reviewed, the number of rotations counted, and the output value from the ADC recorded. The rotations and the video duration where then used to capture rotations per second data utilizing the relation in Equation 2. Where (Rotation time = total video duration/8).

$$T_R = \frac{T_{Duration}}{8}$$

$$\frac{R_{Total}}{T_R} = R_S \tag{2}$$

Finally to get the speed, the rotations per second R_S were multiplied with the distance travelled per rotation, this is shown in Equation 3.

$$Speed = 2\pi r(R_S) \tag{3}$$

Where r the radius of the circle made when the anemometer spins, or the distance between the anemometer edge and the central spindle.

Using the readings in Table 1. We can interpolate and find a general formula to represent different values. The relation is estimated to be linear and as a result the slope can be determined from the the readings at 120 and 160 through standard slope calculation means. The y-intercept is 90 as a result and so a linear

Table 1: Interpolation Table

Anemometer Radius = 0.15 meters			
ADC Readings	**Rotations Per Second**	**Meters Per Second**	**Kilometers Per Hour**
90	0	0	0
120	3.2	3.02	10.872
160	3.48	3.3	11.880

relationship can be made. Firstly the slope can be derived as the difference of 3.8 and 3.02 (independent variable - speed of anemometer/wind) over the difference of 160-120 (ADC Results - dependent variable). The intercept is known as at 0 m/s the ADC due to noise will still read 90.

$$y = mx + b$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\therefore y = \frac{160 - 120}{3.3 - 3.02}x + b$$

This means the linear function representing the speed in m/s with regards to the ADC value of the anemometer can be expressed in Equation 4 where W_S is Wind Speed in meters per second and ADC is the Anemometer output.

$$ADC = 142.857(W_S) + 90$$

$$\frac{ADC - 90}{142.857} = W_S \tag{4}$$

2.3 MCU Firmware

The firmware implemented on the NodeMCU32S was focused on 2 main goals. Firstly it was tasked with the collection of all sensor data as well as the post processing required to convert the values into standard units. Secondly it had to connect to a WiFi network and communicate to the host server through the use of the MQTT protocol. The development of the firmware was done with Arduino. A flowchart of the behaviour of the firmware is shown in Figure 2.15

2.3.1 Value Conversion

To accurately determine PPM values for the environmental sensors, 2 different codes were used. The first to isolate what the average "Clean Air Quality" resistance is (R_0). This was done by reading from the analog pins, connected to the three sensors after allowing them to adequately heat up. Given that the sensors use resistors in a voltage divider setup between the sensor resistor (internal resistor whose values change based on the detected chemical concentration), and the static resistor, the voltage of the sensor resistor which is the output of the system can be used to calculate the current sensor resistance. This is achieved through use of the Voltage Divider Rule (Equation 5) for the parallel connection of R4 and R5 in Figure 2.13.

$$V_S = V_{CC}\frac{R_2}{R_S + R_2} \therefore R_S = \frac{V_{CC}R_2}{V_S} \tag{5}$$

This was implemented in an Arduino Sketch in which the calculated R_S for each sensor was averaged over a range of values. These readings were then utilized as baselines to accurately determine the PPM. After averaging over 30 samples of 10,000 values the baselines for the three sensors were determined and are shown in Table 2.

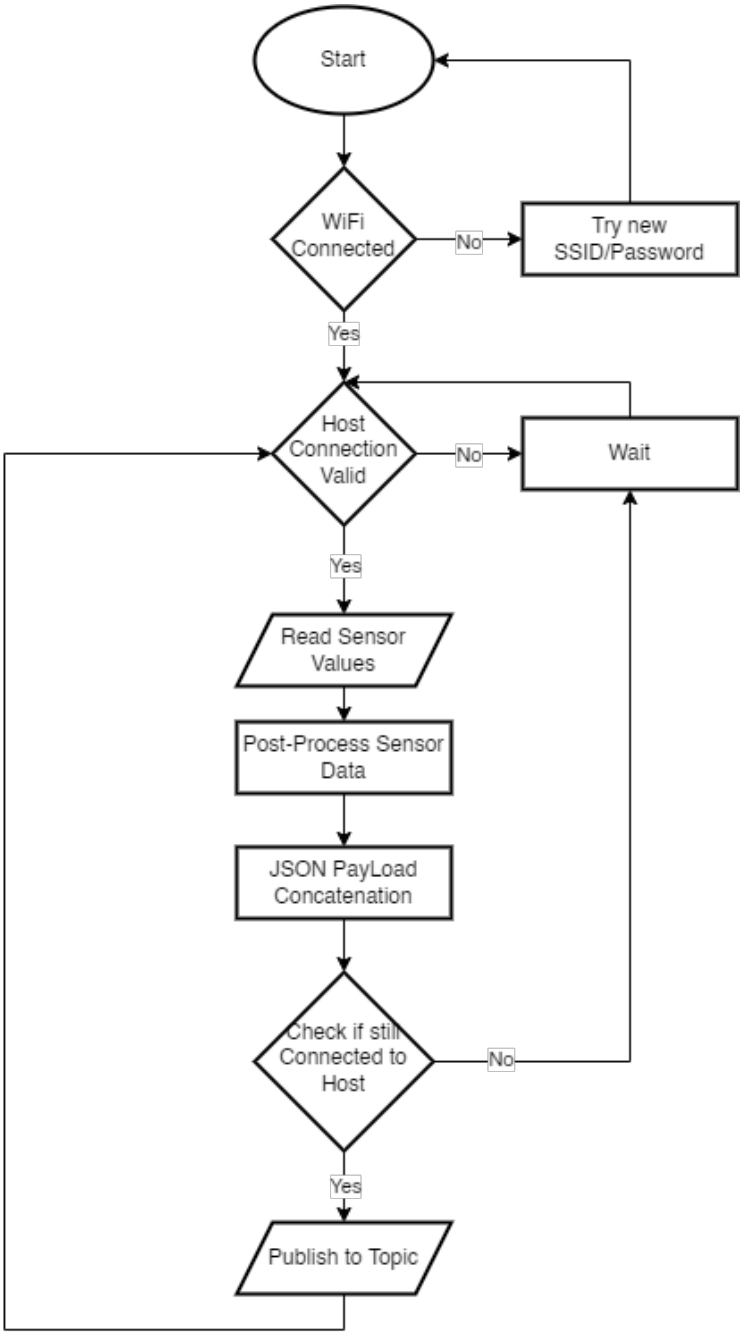

Figure 2.15: Weather and Environmental Monitoring System Firmware Flowchart

2.3.2 Code Implementation

As mentioned earlier the NodeMCU32S must perform 2 key tasks. A) Facilitate a connection and communication between itself as a gateway device, and the cloud host by connecting to a WiFi Network and B) Collecting and performing post processing on sensor data. Below is the setup including the libraries and the global variables utilized in the Arduino Sketch. The key libraries used in the creation of the Arduino Sketch are:

- **PubSubClient** (MQTT Functionality and Interfacing)

Table 2: Baseline Average Resistance (Ω)

Clean Air Quality Baselines	
Sensor	**Resistance in Ohms**
MQ2	6983.38
MQ7	2991.92
MQ135	2115.56

- **Adafruit BMP085 library** (BMP 180 Connection and Reading)
- **DHT Sensor Library by Adafruit** (DHT Connection and Reading)

```
#include <Adafruit_BMP085.h>
#include <Adafruit_Sensor.h>
#include <DHT_U.h>
#include <WiFi.h>
#include <PubSubClient.h>

//Anemometer
#define AnemPin 39
#define TotalAnemRead 2000

//Gas Sensors
#define MQ7Pin 34
#define MQ2Pin 35
#define MQ135Pin 36

//ThingsBoard Connection
#define MaxConnectAttempts 100
const char* ssid = "***";
const char* ssid2 = "********";
const char* password = "*********";
const char* password2 = "**********";
const char* mqtt_server = "demo.thingsboard.io";
const int mqtt_port = 1883;
WiFiClient espClient;
PubSubClient client(espClient);

//DHT11
static const int DHT_SENSOR_PIN = 17;
DHT_Unified dht(DHT_SENSOR_PIN, DHT11);

//BMP
Adafruit_BMP085 bmp;

//Value Holders
float gas, air, carbonmonoxide, temperature, pressure, humidity, windspeed, altitude = 0;
String jsonoutput;
```

The setup loop of the Arduino Sketch functionally initializes both the DHT and BMP objects, additionally

it runs the WiFi connection function, which loops continuously between the specified network SSIDs and Passwords until a successful connection is made.

```
void setup( )
{
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  dht.begin();
  if (!bmp.begin()) {
        Serial.println("Could not find a valid BMP085 sensor, check wiring!");
        while (1) {}
  }
}
```

The main loop of the Arduino Sketch is responsible for checking and maintaining a connection to the cloud host, calling sensor read functions, for all the different sensors, and finally publishing that data to the specified topic. The sensor read functions then update the global variables holding the sensor data to be transmitted. Next the ReturnJSON() function is called which then concatenates all the information into a JSON payload that the MQTT Broker will understand. The NodeMCU then published the information to the telemetry topic of the Cloud Host through MQTT. Finally a check is done to ensure that the connection to the host is still valid.

```
void loop( )
{
  if (!client.connected()) {
    reconnect();
  }

  // Read Speed
  ReadWindSpeed();
  delay(300);

  //Read Gas Sensors
  ReadGasSensors();
  delay(300);

  //Read DHT
  ReadDHT();
  delay(300);

  //ReadBMP
  ReadBMP();
  delay(300);

  client.publish("v1/devices/me/telemetry", ReturnJSON());
  client.loop();
}
```

During the determination of the baseline sensor resistance as shown in Table 2, the following program was used to acquire the necessary data.

```
void loop() {
  for(int i = 0; i <10000; i++){
```

```
  int MQ2sensorValue = analogRead(MQ2Pin);
  int MQ7sensorValue = analogRead(MQ7Pin);
  int MQ135sensorValue = analogRead(MQ135Pin);
  MQ2sensor_volt=(float)MQ2sensorValue/4096*3.3;
  MQ2RS_gas = ((3.3 * MQ2R2)/MQ2sensor_volt) - MQ2R2;
  MQ2R0 = MQ2RS_gas / 1;

  MQ7sensor_volt=(float)MQ7sensorValue/4096*3.3;
  MQ7RS_gas = ((3.3 * MQ7R2)/MQ7sensor_volt) - MQ7R2;
  MQ7R0 = MQ7RS_gas / 1;

  MQ135sensor_volt=(float)MQ135sensorValue/4096*3.3;
  MQ135RS_gas = ((3.3 * MQ135R2)/MQ135sensor_volt) - MQ135R2;
  MQ135R0 = MQ135RS_gas / 1;

  MQ135Avg += MQ135R0;
  MQ7Avg += MQ7R0;
  MQ2Avg += MQ2R0;

  }
  Serial.print("MQ2 R0: ");
  Serial.println(MQ2R0);
  Serial.print("MQ7 R0: ");
  Serial.println(MQ7R0);
  Serial.print("MQ135 R0: ");
  Serial.println(MQ135R0);
  delay(500);
}
```

2.4 Cloud Host

Taking full advantage of IoT, means distributing and analyzing the information gathered from all means of edge devices. For these purposes cloud hosting has become an invaluable tool for aggregating large amounts of data. In this system a cloud host is utilized to track and monitor sensory information in the form of a dashboard. The platform of choice is an open-source service called ThingsBoard which provides access to a host and dashboard creation functionality amongst many other features.

2.4.1 Gateway/Device Creation

The first requirement in the creation of a dashboard, is a means of collecting information. This is done through the creation of a device. In this case a gateway device labelled as the NodeMCU32 was created. This "device" would function as the data entry point by means of the MCU Firmware. Essentially the NodeMCU32S will connect to the host, with the credentials of this "device" and provide telemetry data. From the host perspective it simply receives the data, through MQTT over the internet from what it perceives to be the gateway device. Different device credentials can be set, from an authorization token to a simple username and password setup.

2.4.2 Dashboard Creation

Next using the variety of cards and premade tiles a dashboard was created to display and showcase the readings from the various different sensors. This provides anyone with access to the dashboard real-time, accurate data directly from the NodeMCU32S.

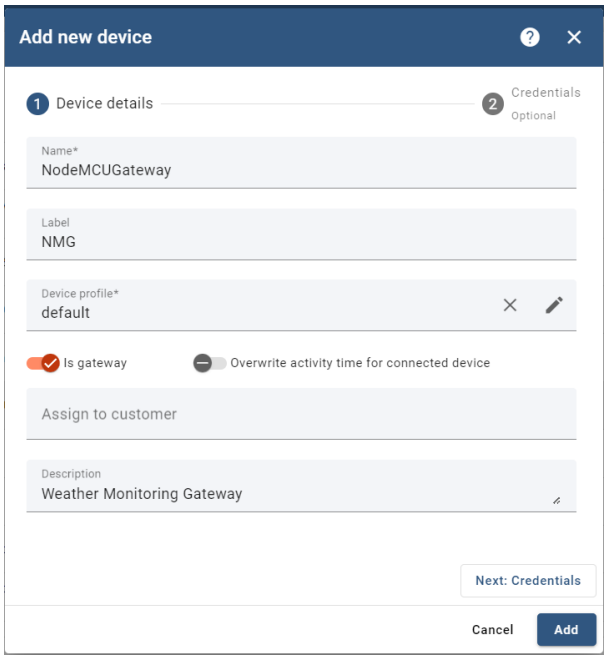

Figure 2.16: Device Creation Process

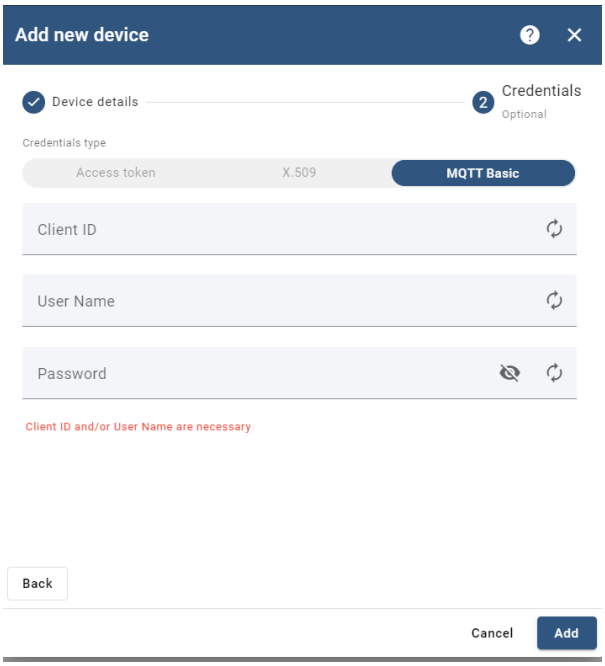

Figure 2.17: Device Credential Specification

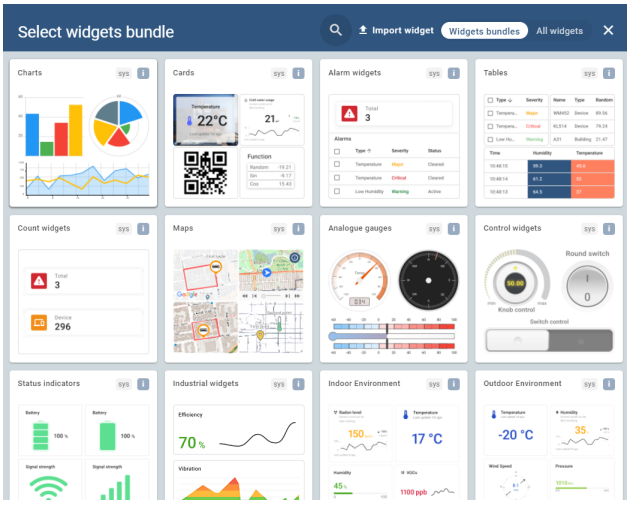

Figure 2.18: ThingsBoard Dashboard Widgets Bundles

3 Results

3.1 System Review

The entirety of the system successfully functioned and has been tested for hours on end with no hiccups. Data is monitored on the dashboard from anywhere with an internet access, and updates occur at 2 second intervals. The functioning dashboard can be seen below in Figure 3.19. The circuit implementation can be view in an image below as well (Figure 3.21).

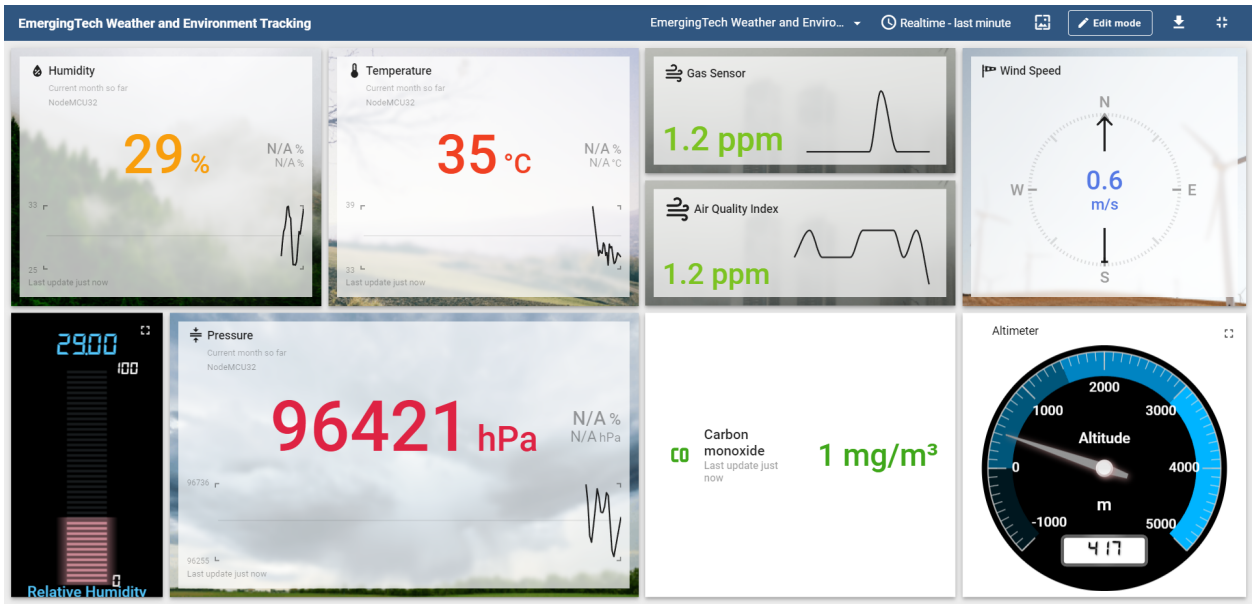

Figure 3.19: Weather and Environmental Monitoring System Dashboard

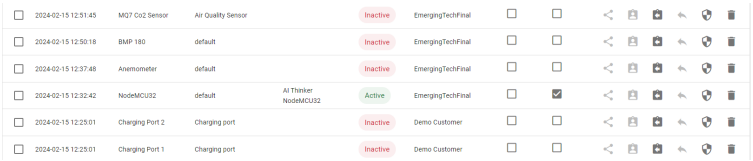

Figure 3.20: ThingsBoard Gateway Device Active

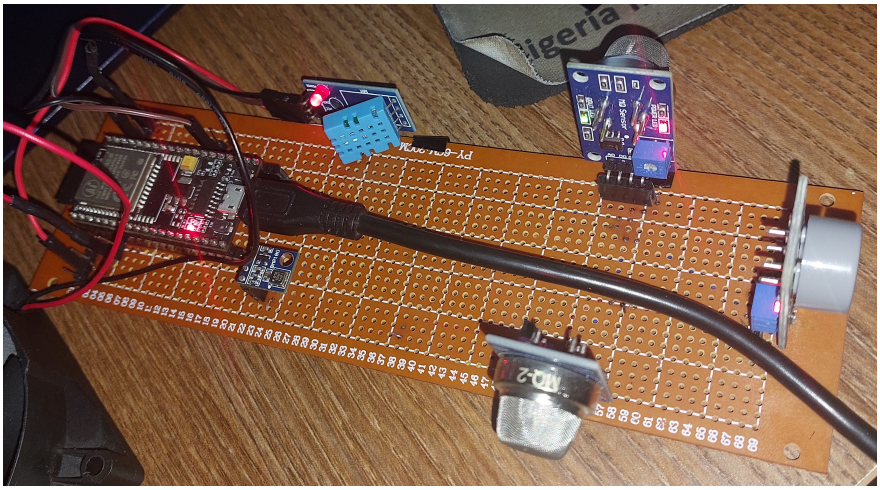

Figure 3.21: Weather and Environmental Weather Hardware System Implementation

3.2 Future Work

Some areas in which the system may be improved in the future include:

- Wind-Vane creation using rotary encoder: Currently the Anemometer only measures windspeed, and not direction. Creating a windvane by using a rotary encoder to ascertain its directional position would allow for more detailed information as to wind direction, and changes over time, as opposed to the scalar nature of wind-speed data.

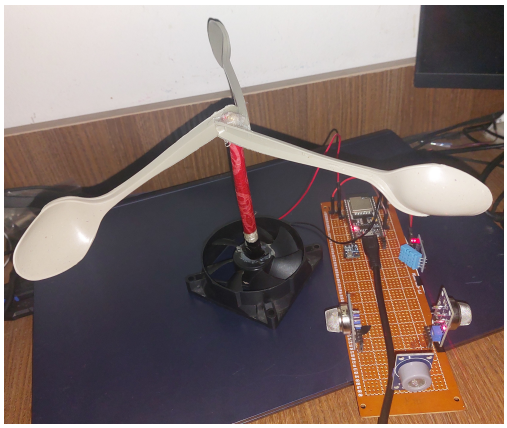

Figure 3.22: Weather and Environmental Weather Hardware System Implementation with Anemometer

- Heavier Analytics and Predictive Measurements on Collected data: Currently the dashboard functions as a means of monitoring and observing the data. This can be combined with actions to enable the system to toggle alarms, push notifications, among other responses. This however does not take advantage of the volume of data being pushed to the cloud in an informatics based approach. Using the vast data to create predictive models, or to generalize even more information from sensor data could be particularly useful.
- More Accurate Scaling for Gaseous Sensors and Anemometer, the scaling for the MQ2,7,135 could be improved by performing proper calibration with a dedicated gas sensor, ensuring that rough estimations could be avoided. The anemometer's model for determining wind speed has many flaws, and makes many assumptions, particularly in the application of a linear model. Better data collection, with high data points would be crucial to improve the accuracy of the results.
- Noise Reduction for Anemometer: Filtering and Amplification would prove to be a superior solution to the software based averaging approach currently employed with regards to reducing anemometer noise.
- Better Shape/Profile for Anemometer: The spoon-like shape of the anemometer is far from ideal, and struggles more with capturing passing wind, by adjusting the shape the overall capture rate in terms of mechanical motion would more closely resemble the actual wind motion.

4 Conclusion

In conclusion the project successfully utilized the principles of IoT, through usage of a cloud host, the MQTT protocol, and an embedded system design to seamlessly monitor weather and environment characteristics via an intuitive dashboard. Beyond the collection of digital output sensors including the BMP180 and the DHT11, analog sensors were used and their ADC values were mathematically converted into useful units. For the custom anemometer this involved some data collection, extrapolation, and assumptions, but was able to provide stable and usable readings. The creation of a dashboard using ThingsBoard allowed for clear, and effective data monitoring, completing the objective of the project.

5 References

[1] EMQXTeam, *What is the mqtt protocol and how does it work?* en. [Online]. Available: https://www.emqx.com/en/blog/the-easiest-guide-to-getting-started-with-mqtt.

[2] Bluetooth, *Bluetooth technology overview*, en-US. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

[3] LoraAlliance, *What is lorawan® specification*, en-US. [Online]. Available: https://lora-alliance.org/about-lorawan/.

[4] AIThinker. [Online]. Available: https://docs.ai-thinker.com/en/esp32-s_development_board.

[5] LastMinuteEngineers, en-US, Feb. 2022. [Online]. Available: https://lastminuteengineers.com/esp32-pinout-reference/.

[6] R. Pelayo, *How to use the mq-7 carbon monoxide sensor*, en, Apr. 2018. [Online]. Available: https://www.teachmemicro.com/use-mq-7-carbon-monoxide-sensor/.

[7] B. Marshall and H.-G. Kristen, *How electric motors work*, en-us, Apr. 2000. [Online]. Available: https://electronics.howstuffworks.com/motor.htm.

[8] K. Sunil, *Running a brushed dc motor as a generator*, English, Dec. 2021. [Online]. Available: https://www.portescap.com/en/newsroom/whitepapers/2021/12/running-a-brushed-dc-motor-as-a-generator#:~:text=It%20may%20surprise%20design%20engineers,suitable%20for%20AC%20voltage%20applications..