



# Design & Innovation Project (DIP)

## Project Report

<<A PARTICLE DETECTOR FOR AIR QUALITY MONITORING>>

Project Group: <<E016>>

Supervisor: A/P Chan Pak Kwong

Name	Matriculation Number
Ho Kok Pin	U2020889C
Joe Cheng Kim Wei	U1922906G
Chua Jian Ming	U2023510F
Dhuwaragish Ravichandrakumar	U2020947D
Xu Jie	U2021616J
Chua Beng Choon	U2021375D
Kanishk Srivastav	U1922059H

**School of Electrical and Electronic Engineering  
Academic Year 2021/22  
Semester 2**

## **Acknowledgement**

Our Team would like to express special thanks and sincerest gratitude to our main supervisor, Associate Professor Chan Pak Kwong, project lab technician Madam Lim, and Mr Tay, and workshop supervisor for their continuous support and generous assistance which make this project possible & successful.

Assoc. Prof. Chan never misses our weekly meeting session to discuss the progress of the project and provide insightful ideas that could add value to our project objectives. Besides, his technical expertise broadens our approaching viewpoints toward complex problems which could possibly maximize the efficiency & reliability of the solution.

Nevertheless, continuous support by Madam Lim and Mr. Tay also aids the progress of the project where they always try their best to provide us with whatever tools or components that we need. In addition, they also try their best to accommodate our project requirements and make arrangements such that our project could be done in a safe environment.

Last but not least, we would also like to extend our gratitude to other anonymous individuals that helped us along the way.

## Table of Contents

<u>Chapter 1 : Purpose / Project Objectives</u>	1
<u>Chapter 2 : Project Summary &amp; Scope</u>	2
<u>Chapter 3 : Sensors</u>	
3.1 MQ 7 Carbon Monoxide	3
3.2 MQ 3 Alcohol Sensor	6
3.3 MQ 135 Carbon Dioxide	10
3.4 PMS5003 PM1, PM2.5, PM10	12
<u>Chapter 4 : IoT Microcontroller &amp; Data Processing</u>	17
4.1 ESP32	17
<u>Chapter 5 : Power Harvesting &amp; Management</u>	
5.1 Overview of Power Management System	21
5.1.1 Solar Panel 6V 2W by Voltaic System	22
5.1.2 Solar Charger (BQ24074) by Adafruit	23
5.1.3 PowerBoost 500 Basic DC / DC converter by Adafruit	24
5.1.4 Lithium Ion Battery (LLP785060) 3.7V 2500mA	25
5.2 Power saving Module	26
5.3 Battery Voltage Discharge Test	28
<u>Chapter 6 : System &amp; Hardware Integration</u>	
6.1 Subsystems in the project	30
6.1.1 Sensors	30
6.1.2 Power sub-system	30
6.1.3 Air intake system	30
6.1.4 ESP32 microcontroller	30
6.2 Testing of working of individual components before integration	30
6.3 Addition of the power system	31
6.4 Schematic design	31
6.5 Final test on breadboard	32
6.6 Hardware integration into the enclosure box	32
<u>Chapter 7 : ThingSpeak IoT platform</u>	
7.1 Applications of Thingspeak	35
7.2 Alerts	37
7.3 Data Analysis & AI	40
7.4 Thingview	45

<u>Chapter 8 : Schedule</u>	46
<u>Chapter 9 : Cost</u>	47
<u>Chapter 10 : Outcome / Expected Outcomes</u>	48
<u>Chapter 11 :Reflection</u>	50
References	51
Appendix A - Project Members Contribution	
Appendix B - Full Code	

## CHAPTER 1 Purpose/ Project Objectives

In this 21st century, one of the most significant environmental risk factors for leading causes of death, such as heart disease, stroke, lung cancer, and other respiratory illnesses, is air pollution, and we expect to see more challenges & impact due to air quality. According to the World Health Organization (WHO), 9 out of 10 people worldwide breathe dirty air, which contributed to more than 7 million premature deaths since 2016 associated with air pollution, therefore there is an increasing concern for human health which is caused by the poor air quality today. [1]

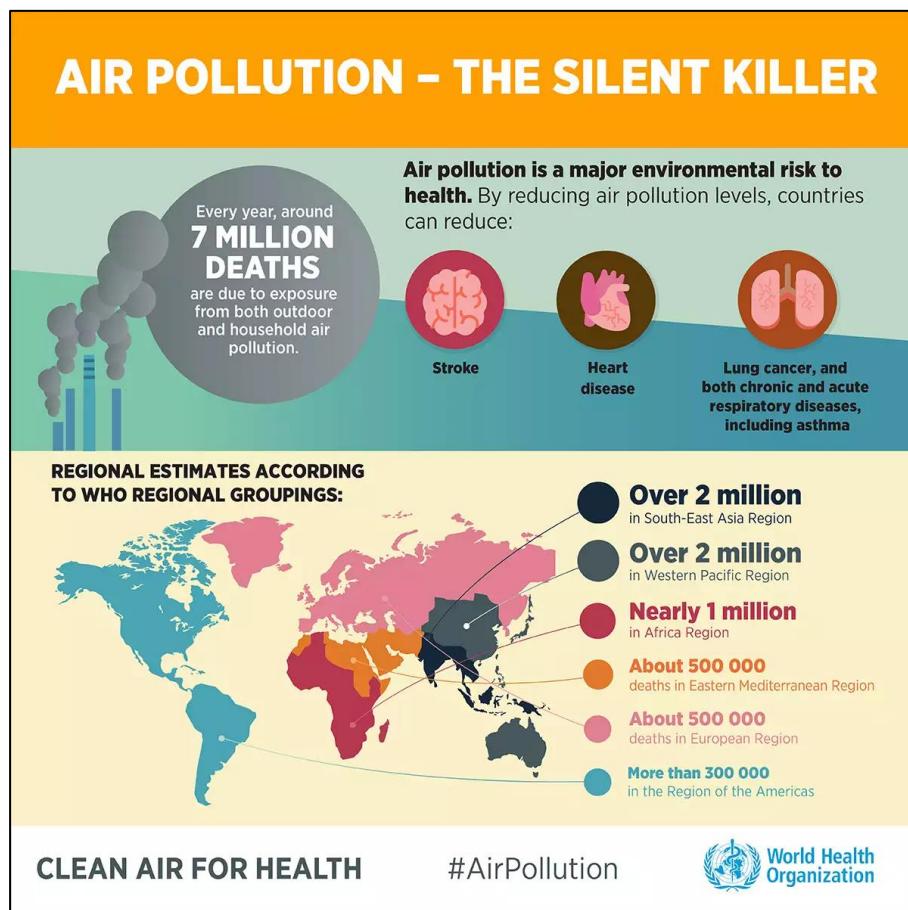


Figure 1.2

The purpose of this project is to design and construct a particle detector with a microcontroller system that deals with both hardware and software co-design for air quality monitoring. This includes creating the alert signal and air quality data, as well as providing a data processing interface with the Internet of Things (IoT) cloud. Furthermore, power management circuits with energy harvesting functions are required to allow the air detector to operate in a remote location with self-sustaining capability.

## **CHAPTER 2 Project Summary & Scope**

The project's primary goal is to construct a particle detector with a microcontroller for air quality monitoring which includes alert features and air quality data along with an IoT dashboard meant for the user to access and view the values and trend of the air particles in the air. In order to obtain the system to work remotely, a solar panel is integrated serving as the power harvesting mechanism which supports battery charging. The air particle detector will be monitored closely in the following few air parameters that can affect the air quality. Which are carbon dioxide, Carbon monoxide, Alcohol, fine particulate matter of 2.5 microns, and fine particulate matter of 10 microns.

However, there is still room for improvement on this project. Firstly, some of the sensors are very sensitive to the environment hence, the readings will fluctuate quite a bit. Next, as the solar panel that we used had a very small surface area, hence, the power harvested is not fully sustainable in the long run. Lastly, the current power consumption of the prototype is still considered. Moving forward to make the power consumption more efficient, it is possible to consider a different set of battery types and further enhance the power saving algorithm and mechanism.

This air particle detector for air quality monitoring will act as a decision-making support system for ambient air monitoring. The system will be equipped with 4 sensors namely, PMS5003, MQ-3, MQ-7, MQ-135, and other supporting components will be a mini-DC fan, PowerBoost, and Solar LIPO Charger, lithium-ion battery, and ESP32 microcontroller.

In our original planning stage, both the microcontroller Arduino UNO and ESP32 were being considered and used in the project. However, during the process after doing plenty of readings and researching, our group realized that we can just integrate the entire project using one microcontroller. Hence, we remove the usage of the Arduino UNO board and proceed with just the ESP32 module.

Moving further, the ESP32 will be used to integrate the sensors and to transmit the encoded sensor data to the IoT platform in which monitoring, visualization, analysis, and alert service will be executed. Environment reading will be taken every 10 minutes (Theoretically, the prototype works at a different interval) and when the sensor value received goes beyond the optimum range, we will alert the user via message to engage necessary safety protocols and measures to prevent any catastrophic consequences on human health.

## CHAPTER 3 Sensors

### 3.1. MQ-7 Carbon Monoxide Sensor



Figure 3.1.1: MQ-7 Sensor

Carbon monoxide is harmful when breathed in as it displaces oxygen and deprives your heart, brain, and other vital organs of the oxygen it needs, causing permanent brain damage and in the worst case, death. Thus, this sensor is included in our project.

#### Introduction

The MQ-7 sensor is highly sensitive to Carbon Monoxide(CO) gas and the sensitivity can be further adjusted via its potentiometer. The sensor is composed of a sensitive layer of tin dioxide( $\text{SnO}_2$ ) that has lower conductivity in clean air.

#### Working Principle

The sensor detection is through its sensitive layer of  $\text{SnO}_2$ . In clean air, donor electrons in  $\text{SnO}_2$  will react with oxygen, which is adsorbed and form a layer of oxide on the surface of  $\text{SnO}_2$ , this prevents electric current to flow. In the presence of CO, however, the surface density of the adsorbed oxygen decreases as it reacts with CO, thereby allowing electrons to flow freely through the sensor. The current flow will therefore give the equivalent readings of the measured CO levels.

## Connecting MQ-7 to ESP32

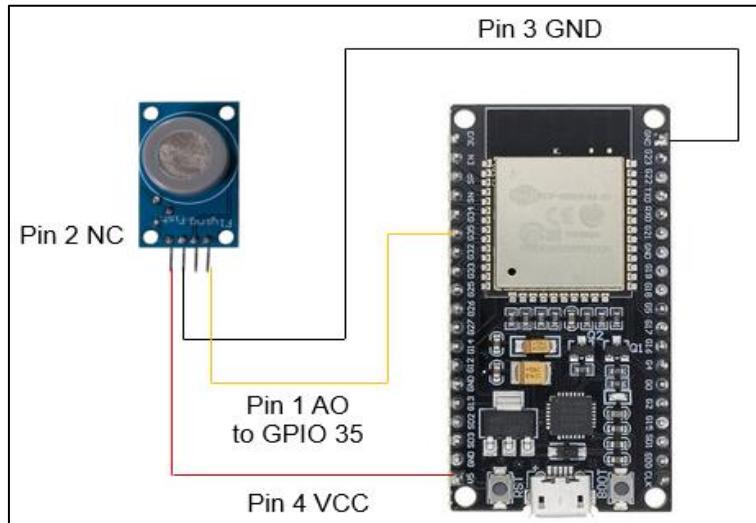


Figure 3.1.2: Connections between MQ-7 and ESP32

PIN1	Analogue Output	To ESP32 GPIO Pin 35
PIN2	Digital Output	Not Connected
PIN3	GND	Ground
PIN4	VCC	+5V

Figure 3.1.3: Pin Definition

## Calibration

First, the necessary library is downloaded to ensure our code would work, the MQ7Sensor library.

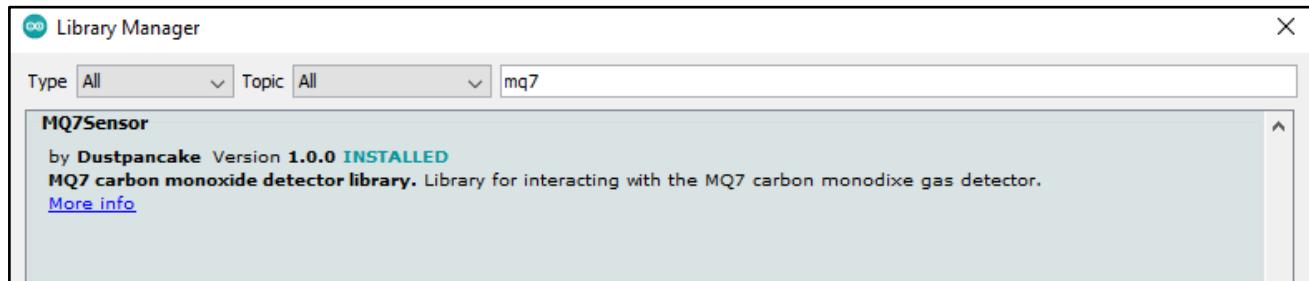
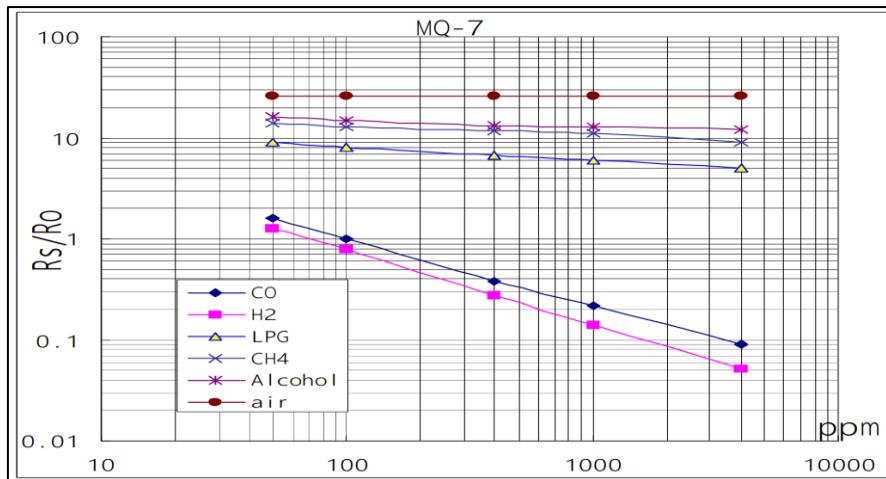


Figure 3.1.4: Library Installation

The pins are then specified and connected to the ESP32 and calibrate the potentiometer to ensure a correct value of RL is obtained and the sensitivity of the sensor is within the range specified in the data sheet.

Figure 3.1.5: Datasheet Graph

The sensor is then preheated for 48 hours to ensure that the sensors are stable and reliable before readings are taken for data visualization purposes.

symbol	Parameters	Technical parameters	Remark
$R_s$	Surface resistance Of sensitive body	2-20k	In 100ppm Carbon Monoxide
a (300/100ppm)	Concentration slope rate	Less than 0.5	$R_s$ (300ppm)/ $R_s$ (100ppm)
Standard working condition	Temperature $-20^\circ\text{C} \pm 2^\circ\text{C}$ relative humidity $65\% \pm 5\%$	RL:10K $\Omega \pm 5\%$	
Preheat time	No less than 48 hours	Detecting range: 20ppm-2000ppm carbon monoxide	

Figure 3.1.6: Datasheet Parameters

## Code

The readings are obtained using the library function to obtain the ppm values of CO in the air. The data is then sent to ThingSpeak and the serial monitor for further analysis.

```
#include "MQ7.h"
#define VOLTAGE 5
float CarbonMonoxide;

// init MQ7 device
MQ7 mq7(35, VOLTAGE);

void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial connection
  }

  Serial.println("Calibrating MQ7");
  mq7.calibrate(); // calculates R0
  Serial.println("Calibration done!");
}

void loop() {
  CarbonMonoxide= mq7.readPpm();
  Serial.print("CO\t"); Serial.println(CarbonMonoxide, DEC);
  ThingSpeak.setField(1, CarbonMonoxide);
}
```

Figure 3.1.7: Code

```

entry 0x400806b4
Connecting to WiFi..
Connected to the WiFi network
CO2    423
Alcohol 53
CO     13.4333200455
PM2.5   12
PM10    9
Voltage 0.00

```

Autoscroll  Show timestamp  Newline  115200 baud  Clear output

Figure 3.1.8: Example of Data Received

### 3.2 MQ-3 Alcohol Sensor

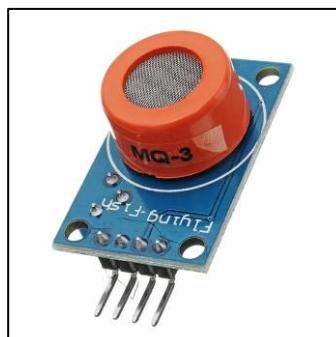


Figure 3.2.1: MQ-3 Sensor

#### Introduction

As inhaling alcohol is hazardous which would also cause a fire hazard when the concentration is too high. Hence, it would be useful for the alcohol sensor to be incorporated into our project.

#### Working Principle

MQ3 alcohol gas sensor is made by using SnO<sub>2</sub> material which has less conductivity in clean air. Whenever it comes close to nearby alcohol gas it starts conducting highly according to the gas concentration. The difference in the output voltage using a microcontroller will allow users to notice the presence of alcohol in the atmosphere.

## Connection

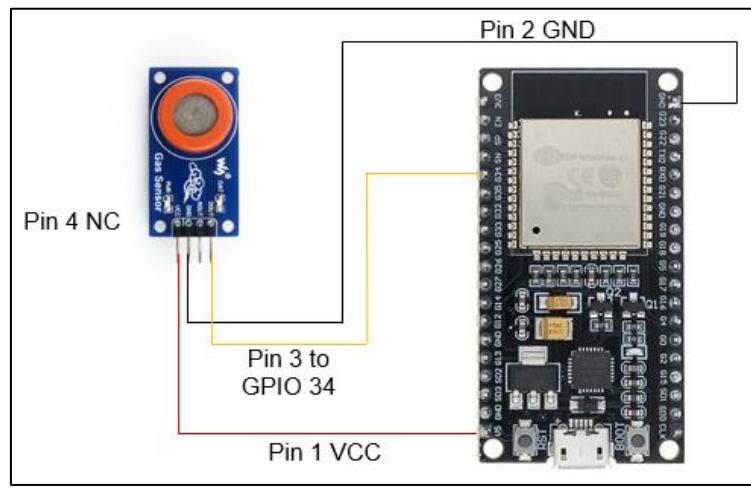


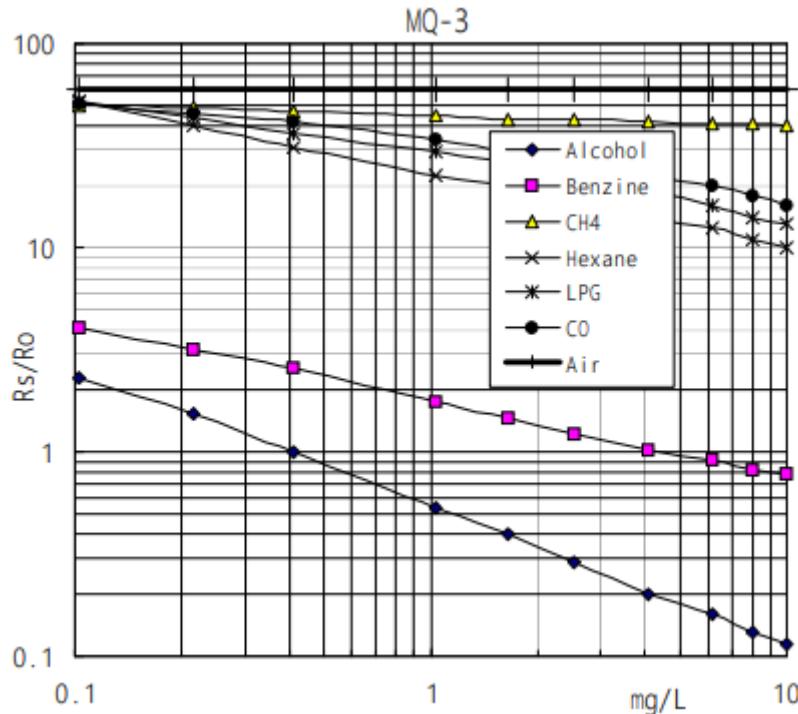
Figure 3.2.2: Connections between MQ-3 and ESP32

PIN1	VCC	+5V
PIN2	GND	Ground
PIN3	Analogue Output	To ESP GPIO Pin 34
PIN4	Digital Output	Not Connected

Figure 3.2.3: Pin Definition

## Calibration

The pins are then specified and connected to the ESP32 and calibrated with the variable resistor to obtain what is specified in the datasheet, an approximate 200ppm of alcohol in the air and a variable resistor resistance value of 200KΩ.

Figure 3.2.4: Datasheet Graph

The sensor is then preheated for 24 hours to ensure that the sensors are stable and reliable readings are taken for data visualization purposes.

Symbol	Parameter name	Technical parameter	Remarks
$R_s$	Sensing Resistance	$1M\Omega - 8M\Omega$ (0.4mg/L alcohol )	Detecting concentration scope: 0.05mg/L—10mg/L Alcohol
$\alpha$ (0.4/1 mg/L)	Concentration slope rate	$\leq 0.6$	
Standard detecting condition	Temp: $20^\circ C \pm 2^\circ C$ Humidity: $65\% \pm 5\%$	Vc: $5V \pm 0.1$ Vh: $5V \pm 0.1$	
Preheat time	Over 24 hour		

Figure: 3.2.5: Datasheet Parameters

After working with the sensors, we realize that the sensor will take 2 minutes for the readings to stabilize, hence a delay is included for the sensor to warm up before values are taken to improve the accuracy of our data.

## Code

```
float sensor_value;

void setup() {
    Serial.begin(115200);
    Serial.println("MQ3 is warming up");
    delay(120000); //2 min warm up time
}

void loop() {
    sensor_value = analogRead(34);
    Serial.print("Sensor Value: ");
    Serial.print(sensor_value);
    Serial.print(" ppm\n");
    delay(2000);
}
```

Figure 3.2.6: Code

The readings are obtained using an `analogRead()` function to obtain the ppm values of alcohol in the air. The data is then sent to ThingSpeak and the serial monitor for further data visualization and analysis purposes.

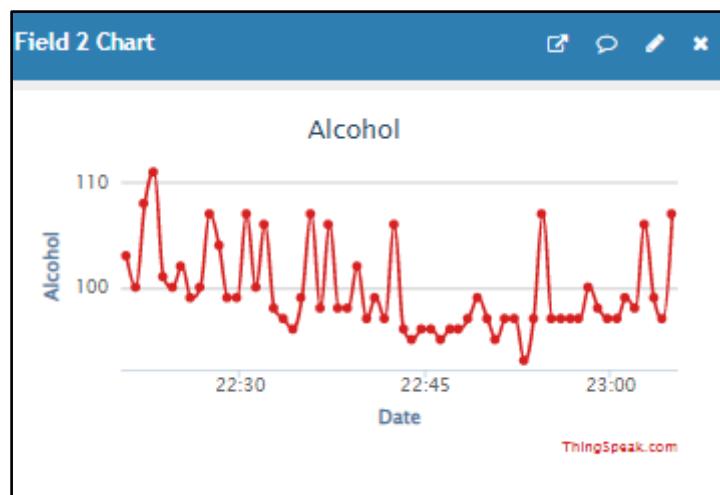


Figure 3.2.7: Example of Data Received at ThingSpeak

### 3.3 MQ-135 Carbon Diode Sensor

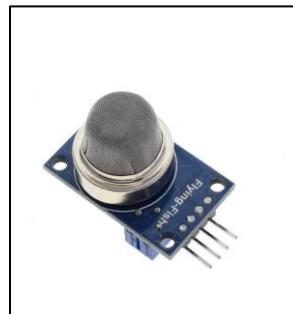


Figure 3.3.1: MQ-135 Sensor

#### Introduction

CO<sub>2</sub> levels in the air are a strong indication of pollution levels. In areas that are not ventilated properly, CO<sub>2</sub> levels can rise up to 1000ppm which is unhealthy.

#### Working Principle

The MQ-135 sensor is a metal oxide gas sensor that utilizes SnO<sub>2</sub>. The resistance of the sensor changes depending on different gas concentrations in the air. The sensor needs to be calibrated to detect the concentration of a specific gas, in this case, the CO<sub>2</sub>.

#### Connection

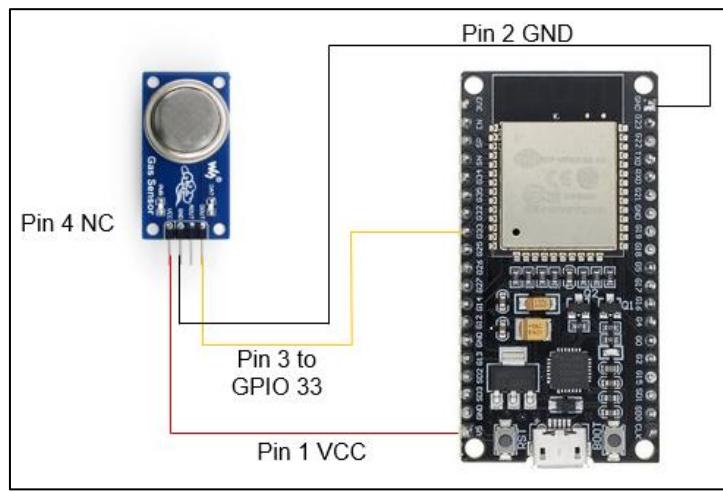


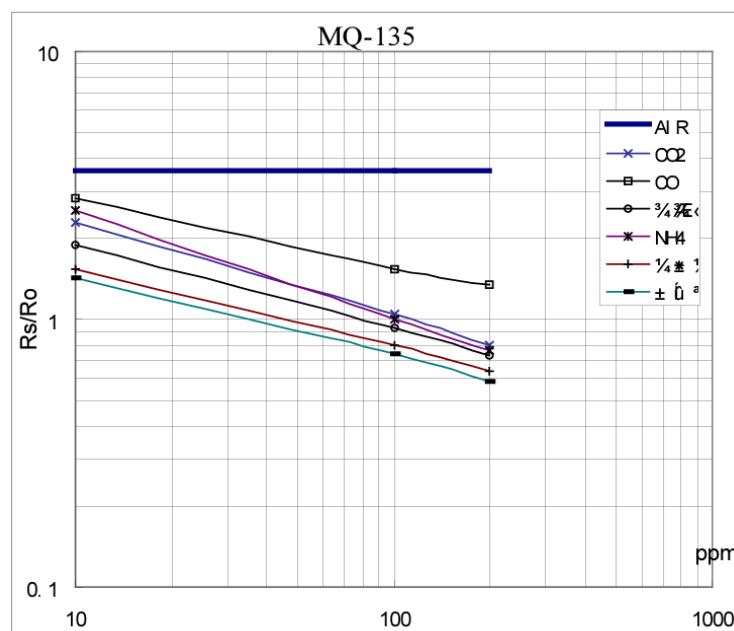
Figure 3.3.2: Connections between MQ-135 and ESP32

PIN1	VCC	ESP V5
PIN2	GND	GND 1
PIN3	AOUT	ESP GPIO 33
PIN4	DOUT	NC

Figure 3.3.3: Pin Definition

## Calibration

For MQ135 there was no pre-existing library available for calibration. Hence the calibration of MQ135 had to be done from scratch. For the sake of calibration, the CO<sub>2</sub> in the lab was approximated to be 450ppm when the sensor was calibrated. The R<sub>0</sub> values were calculated by finding the current and the voltage passing through the sensor. The R<sub>s</sub> values are the ones that kept changing.

Figure 3.3.5: Datasheet Graph

Graph for ratio of R<sub>s</sub>/R<sub>0</sub> and ppm concentration

Once the concentration was found R<sub>s</sub> was calculated with the formula:

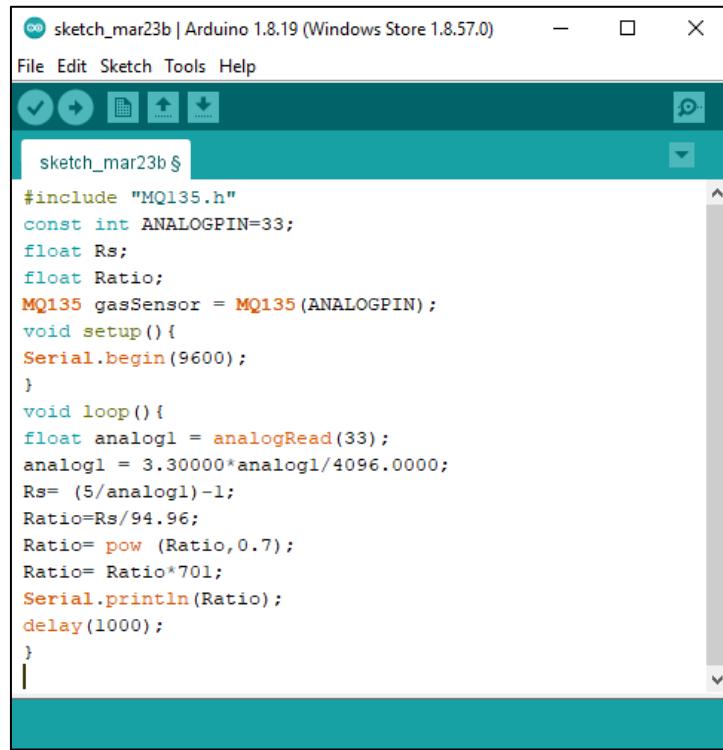
$$R_s = (V_{cc}/V_{RL}-1) \times R_L$$

Which converts to the formula:

$$(5V/(sensorValue * (3.3/4096.0))-1)*RL$$

The constant multiplied with the sensor value to convert the sensor value read by the ESP32 into the right voltage. Once we managed to find R<sub>s</sub> at 400ppm of CO<sub>2</sub>, we could use the spec sheet's graph to extrapolate the R<sub>s</sub>/R<sub>0</sub> values obtained from the sensor to find the respective concentration of CO<sub>2</sub> in the air. The interesting thing to note here is that as CO<sub>2</sub> concentration increases the output voltage of the MQ135 sensor decreases hence it behaves slightly differently from the other MQ sensors.

## Code



The screenshot shows the Arduino IDE interface with a sketch named "sketch\_mar23b". The code is as follows:

```
#include "MQ135.h"
const int ANALOGPIN=33;
float Rs;
float Ratio;
MQ135 gasSensor = MQ135(ANALOGPIN);
void setup(){
  Serial.begin(9600);
}
void loop(){
  float analogl = analogRead(33);
  analogl = 3.30000*analogl/4096.0000;
  Rs= (5/analogl)-1;
  Ratio=Rs/94.96;
  Ratio= pow (Ratio,0.7);
  Ratio= Ratio*701;
  Serial.println(Ratio);
  delay(1000);
}
```

Figure 3.3.6 Code used to calibrate the input sensor value to detect concentration of CO<sub>2</sub>

### **3.4. PMS5003 PM2.5, PM10 Sensor**

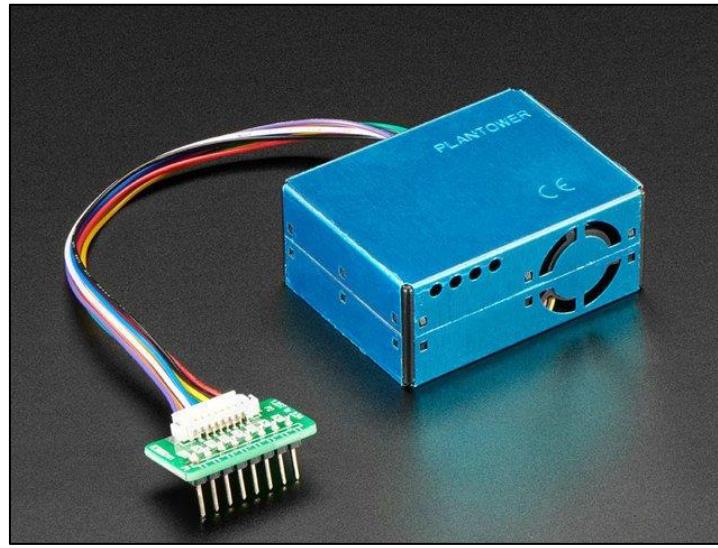


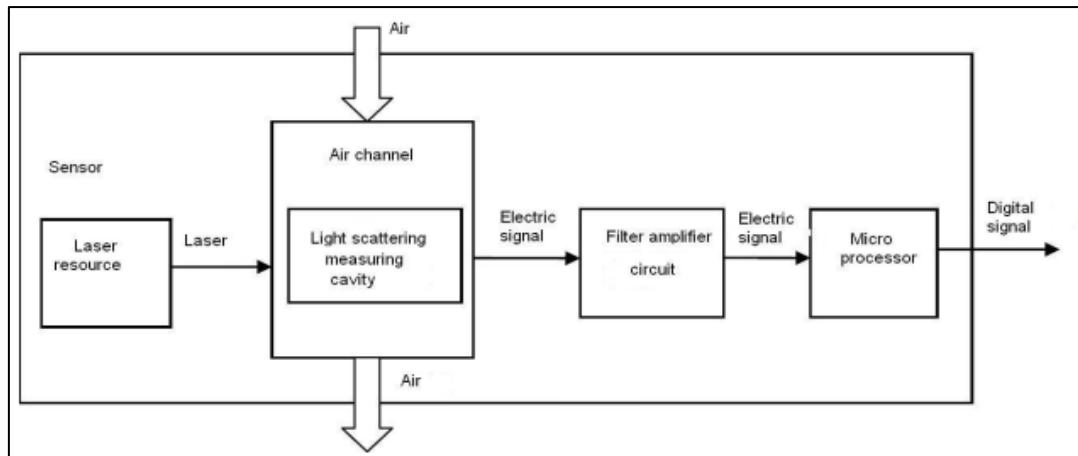
Figure 3.4.1: PMS5003 Sensor

## Introduction

PMS5003 is a digital and universal particle concentration sensor. PMS5003 senses particulates of sizes PM2.5 and PM10. This means that the sensor is able to sense inhalable particle matters that are 2.5 micrometers and 10 micrometers respectively.

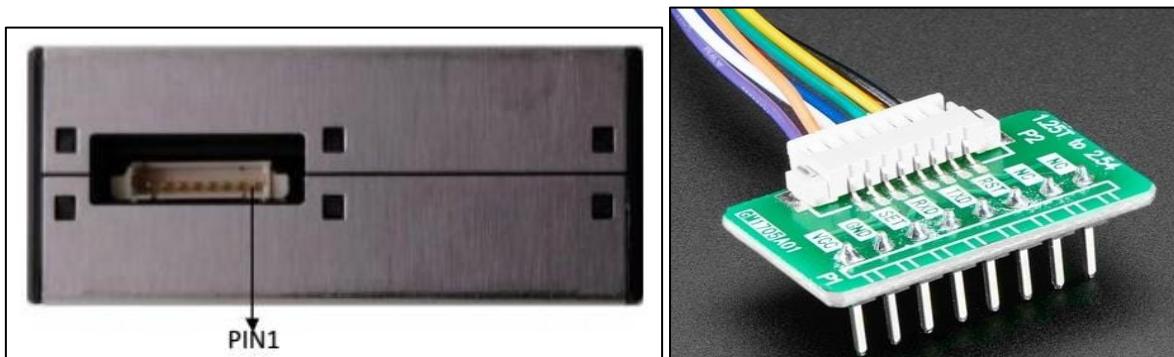
## Working Principle

This sensor uses laser scattering to radiate suspending particles in the air and collects the scattering light to plot the curve of the scattering light change with time. The microprocessor will then calculate the number of particles with different diameters per unit volume.

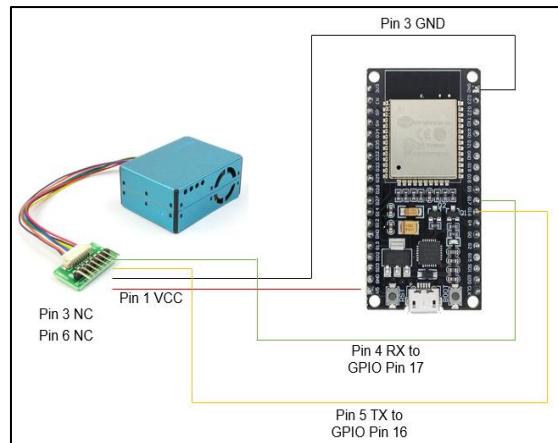


*Figure 3.4.2: Workflow*

## Connecting PMS5003 to ESP32



*Figure 3.4.3 and Figure 3.4.4: PMS5003 Pins*

Figure 3.4.5: Connections between PMS5003 and ESP32

PIN1	VCC	+5V
PIN2	GND	Ground
PIN3	SET	Not Connected
PIN4	RXD	To ESP32 GPIO Pin 17
PIN5	TXD	To ESP32 GPIO Pin 16
PIN6	Reset	Not Connected

Figure 3.4.6: Pin Definition

## Code

First, the necessary library, the EspSoftwareSerial library is installed to ensure our code will work.

Figure 3.4.7: Library Installation

Next, the pins are specified and connected to ESP32, the variables and the size of particulate matter that we want to get a reading of.

```
#include <SoftwareSerial.h>
SoftwareSerial pmsSerial(16, 17);

struct pms5003data {
    uint16_t framelen;
    uint16_t pm10_standard, pm25_standard, pm100_standard;
    uint16_t pm10_env, pm25_env, pm100_env;
    uint16_t particles_03um, particles_05um, particles_10um, particles_25um, particles_50um, particles_100um;
    uint16_t unused;
    uint16_t checksum;
};

struct pms5003data data;
```

Figure 3.4.8: Initialization Code

Referring to the PMS5003 datasheet, the default baud rate for the sensor is 9600 bits per second, with a 32-byte frame that starts from 0x42. This means that it will read 32 bytes worth of data, with every 2 bytes representing different sizes of particles. For example, Data2 refers to PM2.5 with a concentration unit of  $\mu\text{g}/\text{m}^3$  and Data 3 refers to PM10 with a concentration unit of  $\mu\text{g}/\text{m}^3$ .

```
boolean readPMSdata(Stream *s) {
    if (! s->available()) {
        return false;
    }

    // Read a byte at a time until we get to the special '0x42' start-byte
    if (s->peek() != 0x42) {
        s->read();
        return false;
    }

    // Now read all 32 bytes
    if (s->available() < 32) {
        return false;
    }

    uint8_t buffer[32];
    uint16_t sum = 0;
    s->readBytes(buffer, 32);
```

Figure 3.4.9: Function Code

Checksum ensures that data is being read accurately. If the data is accurate, the particulate matter concentrations will be displayed on the serial monitor and to be sent over to the IoT cloud.

```

// get checksum ready
for (uint8_t i=0; i<30; i++) {
    sum += buffer[i];
}

if (sum != data.checksum) {
    Serial.println("Checksum failure");
    return false;
}
// success!
return true;
}

void loop() {
    if (readPMSdata(spmsSerial)) {
        (data.pm10_standard);
        (data.pm25_standard);

        PM25 = data.pm25_standard;
        PM10 = data.pm10_standard;
        Serial.print("PM2.5\t"); Serial.println(PM25, DEC);
        Serial.print("PM10\t"); Serial.println(PM10, DEC);
        ThingSpeak.setField(4, PM25);
        ThingSpeak.setField(5, PM10);
    }
}

```

Figure 3.4.10: Checksum Code

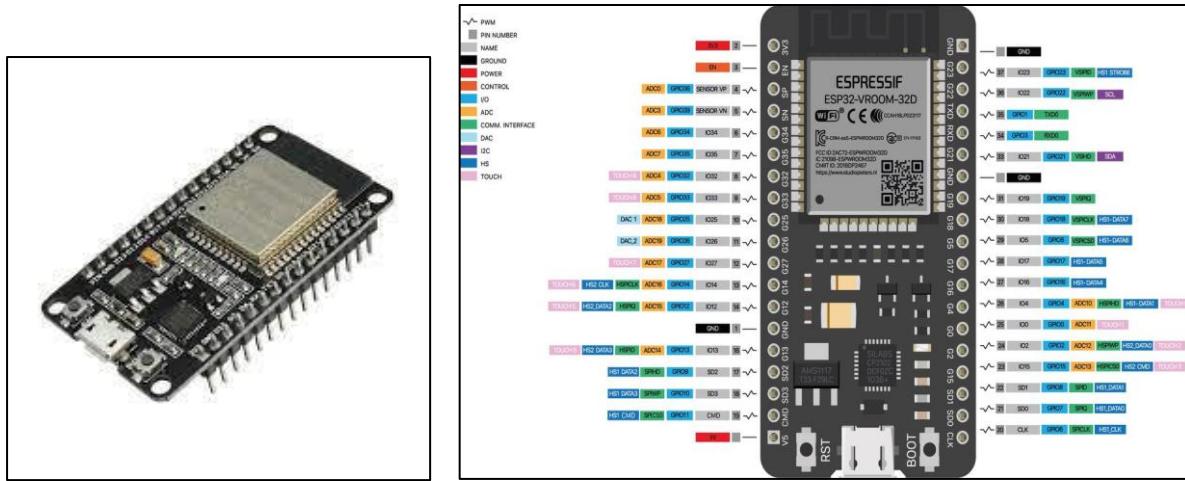
The PMS5003 sensor is able to read PM2.5 and PM10 values which are used by the NEA website(<https://www.haze.gov.sg/>) to provide the average PSI reading for air quality every hour. Below is the table to relate the readings that we have obtained from PMS5003 to real-world applications of PSI readings.

Index Category	PSI	24-hr PM <sub>2.5</sub> ( $\mu\text{g}/\text{m}^3$ )	24-hr PM <sub>10</sub> ( $\mu\text{g}/\text{m}^3$ )	24-hr SO <sub>2</sub> ( $\mu\text{g}/\text{m}^3$ )	8-hr CO ( $\text{mg}/\text{m}^3$ )	8-hr O <sub>3</sub> ( $\mu\text{g}/\text{m}^3$ )	1-hr NO <sub>2</sub> ( $\mu\text{g}/\text{m}^3$ ) <sup>a</sup>
Good	0 – 50	0 – 12	0 – 50	0 – 80	0 – 5.0	0 – 118	-
Moderate	51 – 100	13 – 55	51 – 150	81 – 365	5.1 – 10.0	119 – 157	-
Unhealthy	101 – 200	56 – 150	151 – 350	366 – 800	10.1 – 17.0	158 – 235	1130
Very Unhealthy	201 – 300	151 – 250	351 – 420	801 – 1600	17.1 – 34.0	236 – 785*	1131 – 2260
Hazardous	301 – 400	251 – 350	421 – 500	1601 – 2100	34.1 – 46.0	786 – 980*	2261 – 3000
	401 – 500	351 – 500	501 – 600	2101 – 2620	46.1 – 57.5	981 – 1180*	3001 – 3750

Figure 3.4.11: PM values to PSI values

## CHAPTER 4 IoT Microcontroller & Data Processing

### 4.1 ESP32



*Figure 4.1: ESP32 Pin configuration*

The ESP32 is a family of low-cost, low-power system-on-a-chip microcontrollers that include built-in Wi-Fi and dual-mode Bluetooth. The integration of Wi-Fi, Bluetooth, and Bluetooth LE guarantee that our modules can be used for a wide range of applications, making them genuinely adaptable. Furthermore, the ESP32 Series modules operate in a wide temperature range of -40°C to 105°C, making them ideal for commercial application development.

#### Additional Features of ESP32:

- **Processors:**
  - CPU: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz
  - Ultra-low-power (ULP) co-processor
- **Memory:** 320 KiB RAM, 448 KiB ROM
- **Peripheral interfaces:**
  - Timers and Watchdog
  - Real-Time Clock
  - ADC and built-in Sensors
  - Digital to Analog Converter (DAC)
  - Touch Sensor
  - Pulse Counter
  - Pulse Width Modulation (PWM)
- **Security:**
  - IEEE 802.11 standard security features
  - Secure boot
  - Flash encryption

- **Power management:**

- Internal low-dropout regulator
- Individual power domain for RTC
- 5 µA deep sleep current
- Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

We decided to choose the ESP32 microcontroller for our Internet of Things(IoT) project instead of the basic Arduino board after comparing their performance and features.

		<b>Uno</b>	<b>ESP32</b>
<b>General</b>	Dimension	2.7" x 2.1"	2" x 1.1"
	Pricing	\$20-23	\$10-12
<b>Connectivity</b>	I/O Pins	14	36
	PWM Pins	6	16
	Analog Pins	6	Up to 18*
	Analog Output Pins (DAC)		2
<b>Computing</b>	Processor	ATMega328P	Xtensa Dual Core 32-bit LX6 microprocessor
	Flash Memory	32kB	4MB
	SRAM	2kB	520kB
	EEPROM	1kB	-
	Clock Speed	16MHz	Up to 240 MHz
	Voltage Level	5V	3.3V
	USB Connectivity	Standard A/B USB	Micro-USB
<b>Communication</b>	Hardware Serial Ports	1	3
	SPI (Serial Peripheral Interface)	Yes (1x)	Yes (4x)

	Support		
	CAN (Controller Area Network) Support	No	Yes
	I2C (Inter-Integrated Communication)	Yes (1x)	Yes (2x)
<b>Additional</b>	WiFi	-	802.11 b/g/n
	BlueTooth	-	v4.2 BR/EDR & BLE
	Touch Sensors	-	10

The sensors will be connected to the Esp32 microcontroller analog pins and will be programmed through the Arduino IDE to collect all the measured sensor data collectively and send it to the IoT backend (ThingSpeak) which can be viewed in real-time at the dashboard with appropriate visualizations.

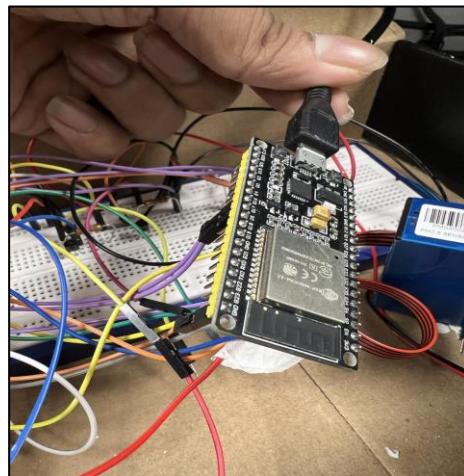


Figure 4.2: ESP32 Prototype Connection

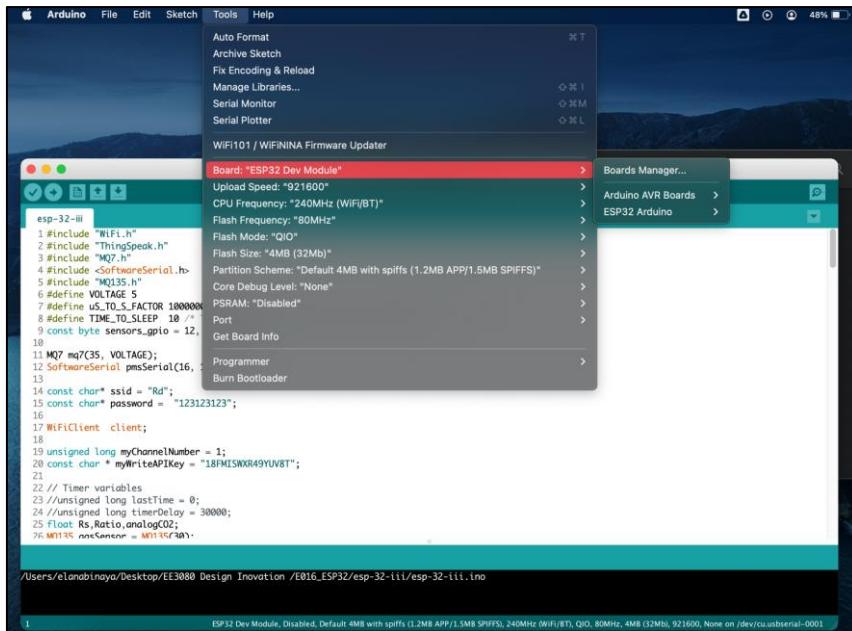
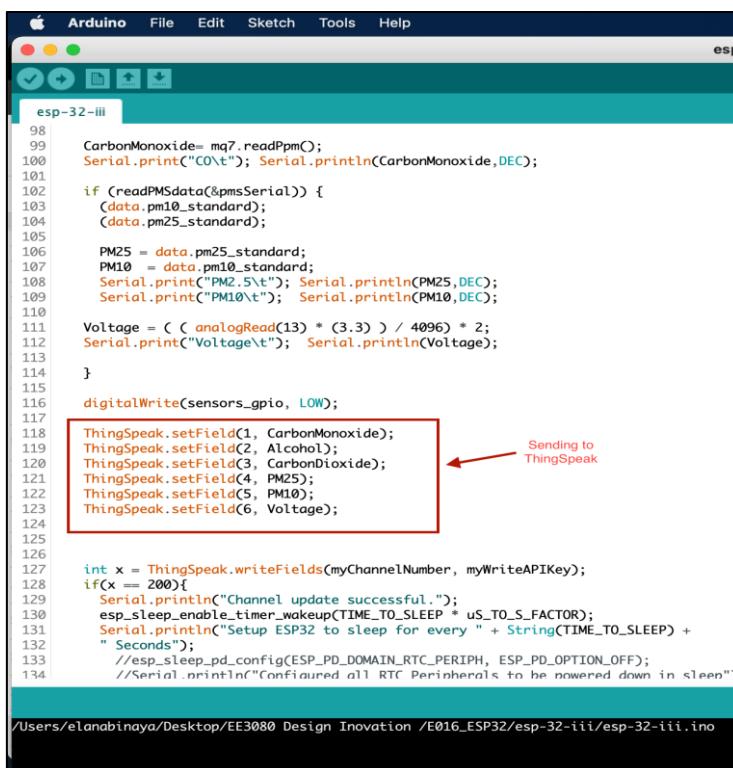


Figure 4.3: Arduino IDE: Board - -> ESP32 Dev Module



*Figure 4.4: Arduino IDE: Set of sensor readings to be sent to ThingSpeak*

## CHAPTER 5 Power Harvesting & Management

### 5.1 Overview of Power Management System

With the goal of having a portable air monitoring system that is able to work remotely, it is required for it to work without a power outlet. With this in mind, the team has designed a power harvesting system that is able to supplement the power usage of the device to increase operation time and reliability during outdoor operations. The components used in this system are as follows.

- 6V 2W Solar Panel by Voltaic System
- Solar Charger (BQ24074) by Adafruit
- PowerBoost 500 Basic DC / DC boost converter by Adafruit
- Lithium Ion Polymer Battery (LLP785060)
- Voltage Divider Block

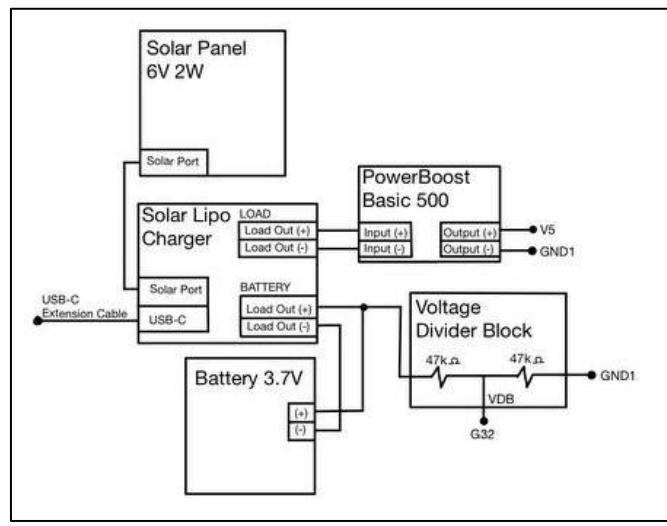


Figure 5.1: Power Management System

The simplified connection diagram in figure 5.1 shows the connection of individual components of the power management system. The BQ24074 which is the solar charger acts as the control for the charging of the battery and powering the load with the solar panel as well as the battery. It also has a USB-C connector to facilitate charging from an external source in case of poor solar panel performance due to bad weather. The PowerBoost DC/DC boost converter boosts the charger output voltage from 3V to 5V to supply power to the microcontroller and the sensors. The voltage divider with two  $47\text{k}\Omega$  resistors is for the ESP32 to track the battery voltage, this section will be elaborated after the power saving segment.

### 5.1.1 Solar Panel 6V 2W by Voltaic System

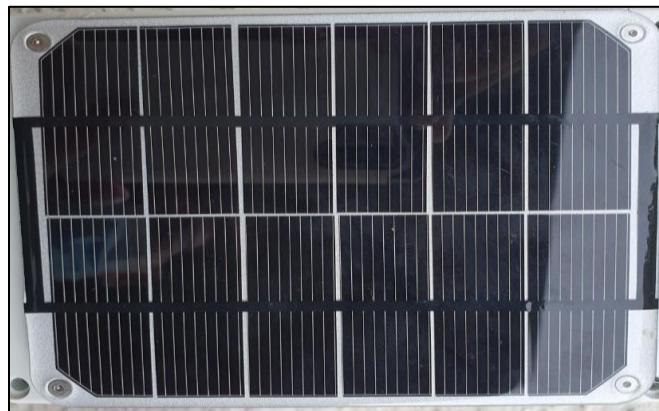


Figure 5.1.1: Solar Panel

This 6V 2W Solar panel was chosen as the ideal output of 6V at 340mA fits the power requirement of our project. It is also compact and has a 19% efficiency in electrical generation from solar energy. It is also outdoor proof with its IP67 waterproof rating and is UV and corrosion-resistant. These specifications aligned with the need for long-term durability for usage outdoors and that is the reason it was chosen.

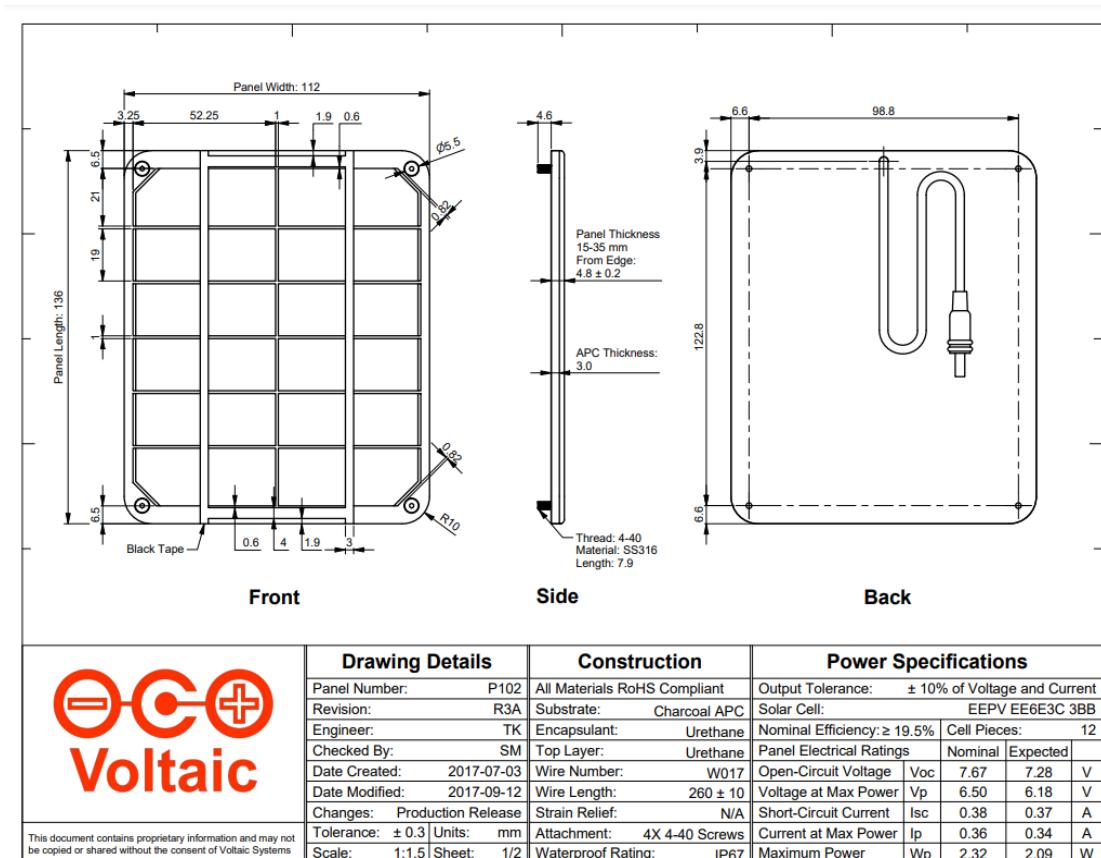
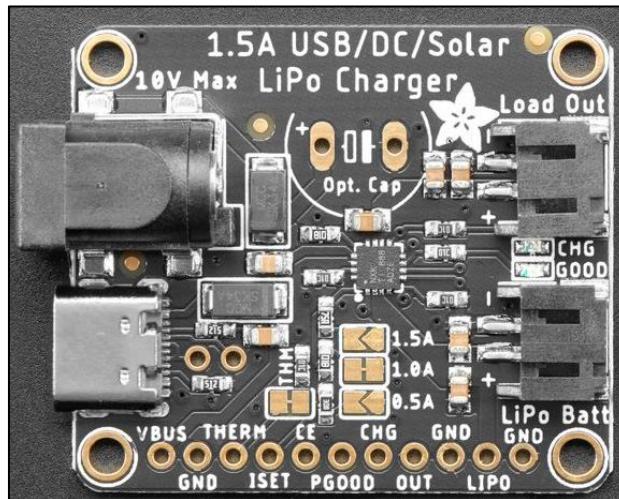


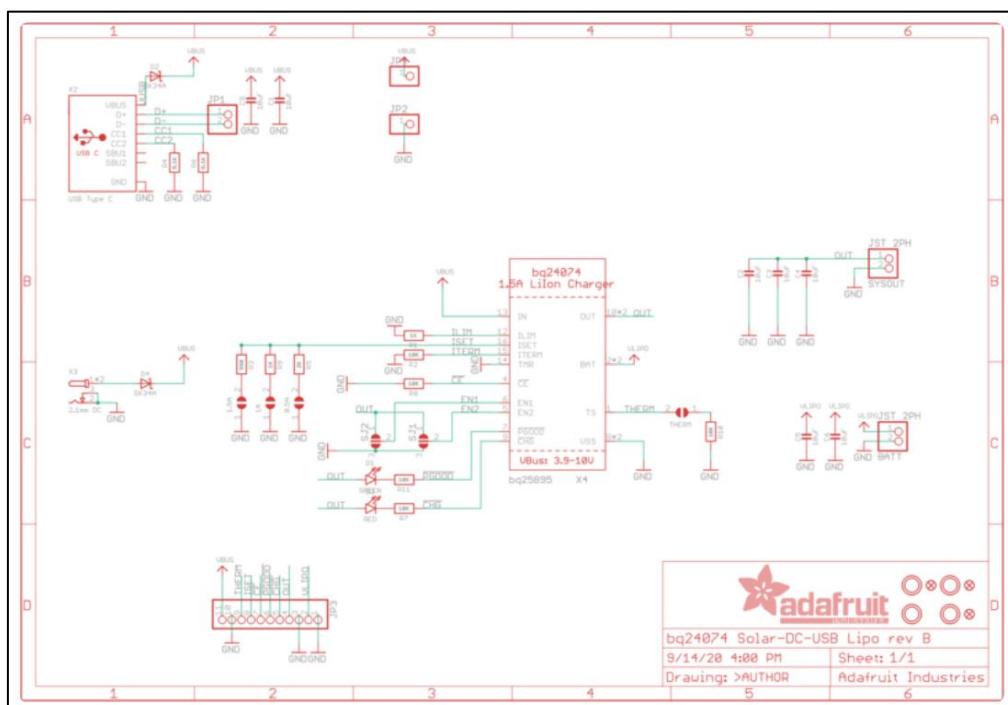
Figure 5.1.1.2: Solar Panel Specifications

### 5.1.2 Solar Charger (BQ24074) by Adafruit



*Figure 5.1.2: Solar Charger (BQ24074)*

The B24074 Solar Charger handles the charging of the battery and powering the load independently. It draws the maximum current from the solar panel while also limiting the current draw when input voltage reduces below 4.5V. The solar charger also has a smart load-sharing function that prioritizes input power over battery power to prolong battery charge cycles. This charger was chosen as it provides similar performance to a Maximum Power Point Tracking charge controller at a relatively low cost.



*Figure 5.1.2: Solar Charger Schematic Print*

### 5.1.3 PowerBoost 500 Basic DC / DC Converter by Adafruit

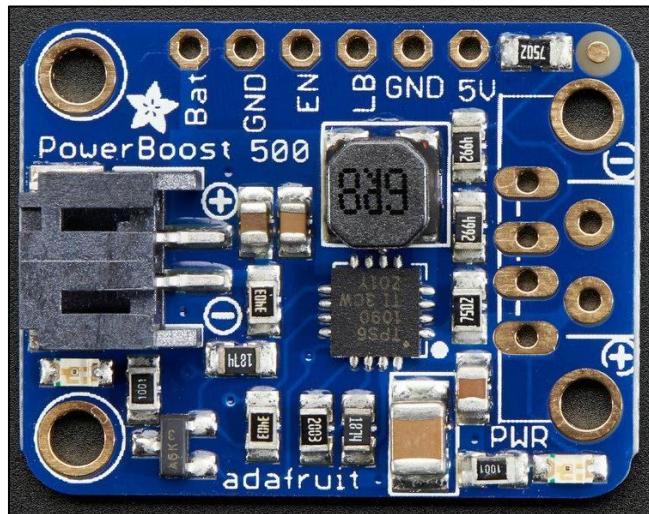


Figure 5.1.3: PowerBoost 500 Basic 5V USB @500mA

The voltage requirement for the ESP32 and the air sensors is 5V. To supply it, the PowerBoost 500 Basic is used to step up the 3V - 4.4V to 5V. The TPS61090 DC/DC boost converter chip used in PowerBoost 500 has a 90% operating efficiency during the conversion.

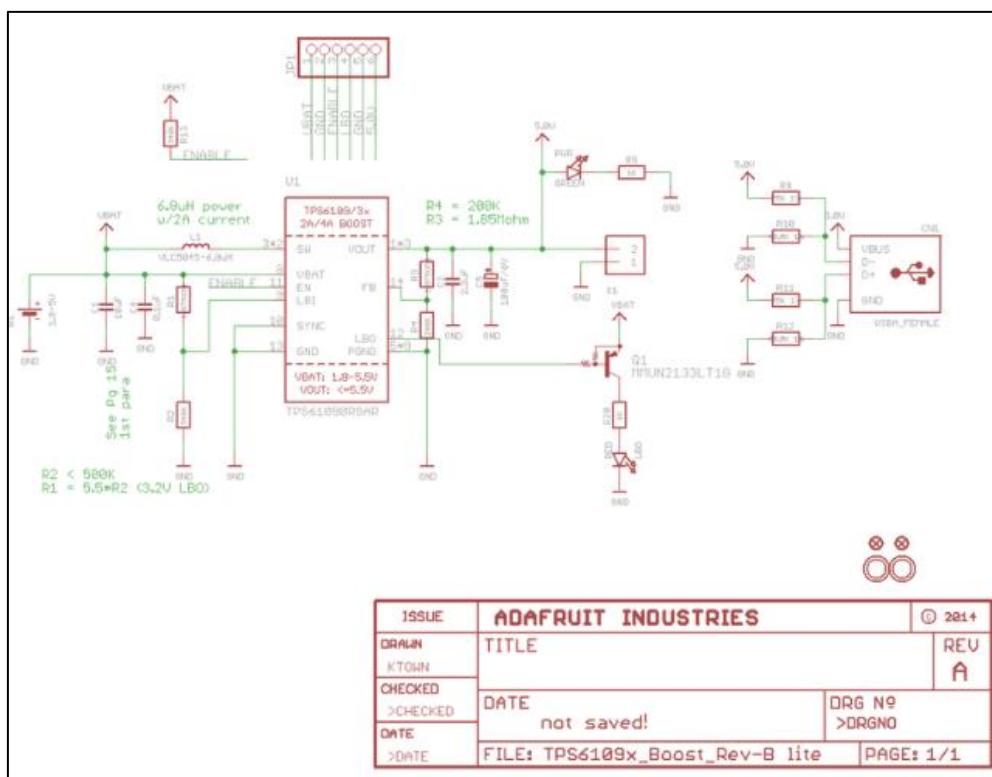


Figure 5.1.3.2: PowerBoost 500 Basic Schematic Print

### 5.1.4 Lithium-Ion Polymer Battery (LP785060)



Figure 5.1.4.1: Lithium Ion Polymer Battery

Lithium Ion Polymer Batteries are rechargeable batteries that have a high energy density and a low self-discharge rate. The LP785060 Model can supply 3.7V and has a capacity of 2500mAh, sufficient for our low-powered air quality monitoring device. This battery is chosen compared to the other batteries due to the number of electronic projects that use this type of battery. It was also recommended by Adafruit that this battery is used for the solar charger to ensure compatibility.

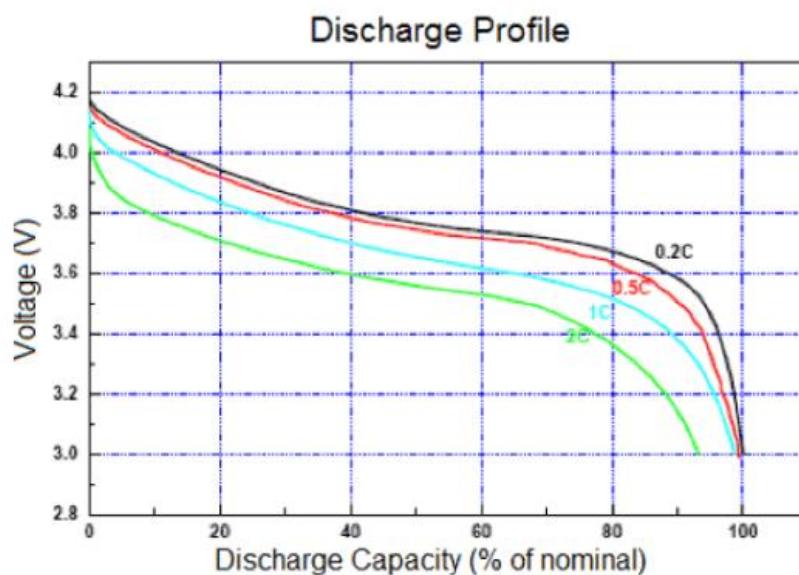


Figure 5.1.4.2: Lithium Ion Polymer Battery Discharge Profile

Referring to figure 5.9, a fully charged lithium-ion polymer battery has a voltage of less than 4.2V. and it will be fully discharged at 3V. Doing further research on the topic, an 8.1% drop in optimal voltage is a good estimate to tell if the battery is low. So 4V at full charge and 3.4V for a low charge.

### 3、 Specification

Item	Specifications	Remark
Nominal Capacity	2500mAh	0.2C <sub>s</sub> A discharge, 25°C
Nominal Voltage	3.75V	Average Voltage at 0.2C <sub>s</sub> A discharge
Standard Charge Current	0.2 C <sub>s</sub> A	Working temperature: 0~40°C
Max Charge Current	1C <sub>s</sub> A	Working temperature: 0~40°C
Charge cut-off Voltage	4.2V	CC/CV
Standard Discharge Current	0.5C <sub>s</sub> A	Working temperature: 25°C
Discharge cut-off Voltage	2.75V	
Cell Voltage	3.7-3.9V	When leave factory
Impedance	≤35mΩ	AC 1KHz after 50% charge, 25°C
Weight	Approx.46g	
Storage temperature	≤1month ≤3month ≤6month	-10~45°C 0~30°C 20±5°C
Storage humidity		65±20% RH

Figure 5.1.4.3: LP785060 Specifications

## 5.2 Power Saving Module

With our initial idea of collecting continuous sensor readings there can be situations where the set of sensor readings will be constant for a period of time and the power available will be wasted for such inefficient operations. In order to improve the efficiency and the lifetime of a fully charged LiPo battery during operation, we can preserve the energy by implementing the available power management feature in the ESP32 board where the microcontroller will be programmed into one of the sleep modes (**Hibernation mode with a timer to wakeup**) utilizing only 5 microampere of current to operate for a certain period of time after collecting a successful batch of sensor readings.

### In Hibernation Mode

- Features ON: RTC timer
- Features OFF: WiFi, BlueTooth, ESP32 Core & Memory and Peripherals

## Available Power Management Modes in ESP32

Power mode	Description			Power consumption
Active (RF working)	Wi-Fi Tx packet			78 mA ~ 90 mA without communication
	Wi-Fi/BT Tx packet			For TX RX more info in the next table
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered on.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
	Normal speed: 80 MHz		Single-core chip(s)	27 mA ~ 34 mA
			Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	–			0.8 mA
Deep-sleep	The ULP co-processor is powered on.			
	ULP sensor-monitored pattern			150 µA 100 µA @1% duty 10 µA
RTC timer + RTC memory			Only RTC timer set to function	
Hibernation	RTC timer only			5 µA
Power off	CHIP_PU is set to low level, the chip is powered off.			1 µA

Fig 5.2.1: Esp32 Power Modes

Additionally, we have installed a 2N6427 transistor block into our circuit to control the current supplied to the sensors and fan by utilizing a Digitalwrite function to enable a particular pin to act as a programmed switch setting it to “High” and “Low” which will turn on and off concurrently with the Esp32 when it’s in sleep mode and wakes up respectively to ensure maximum power consumption can be obtained in the process.

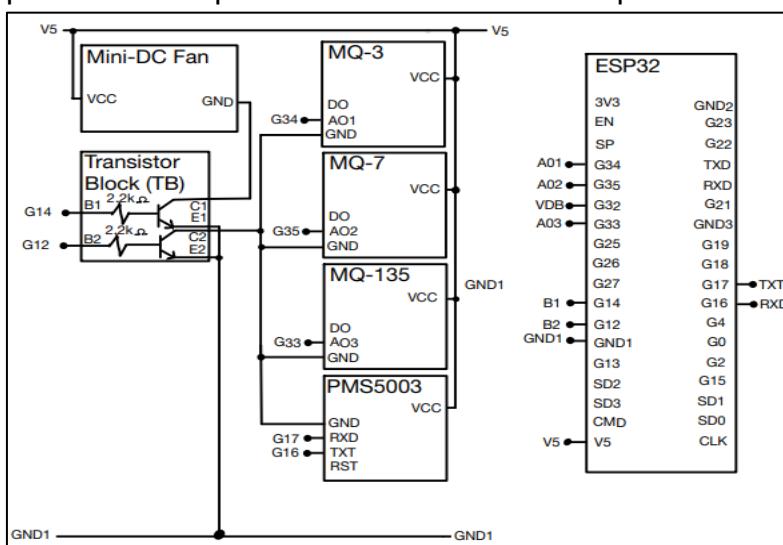


Figure 5.2.2:Schematic of the transistor connection to sensors and Esp32

```

esp-32-iii | Arduino 1.8.19

46 void setup() {
47     // put your setup code here, to run once:
48     pinMode(Censors.gpio, OUTPUT);
49     pinMode(Fan.gpio, OUTPUT);
50     Serial.begin(115200);
51     WiFi.begin(ssid, password);
52     while (WiFi.status() != WL_CONNECTED) {
53         delay(500);
54         Serial.println("Connecting to WiFi...");
55     }
56     Serial.println("Connected to the WiFi network");
57     digitalWrite(Censors.gpio, HIGH); ← Text
58     delay(5000);
59     ThingSpeak.beginClient(); // Initialize ThingSpeak
60     pm25.calibrate(); // calculates R0
61     pmSerial.begin(9600);
62 }
63
64 struct pm500Data {
65     uint16_t framelen;
66     uint16_t pm10.standard, pm25.standard, pm100.standard;
67     uint16_t pm10.env, pm25.env, pm100.env;
68     uint16_t particles_05um, particles_10um, particles_25um, particles_50um, particles_100um;
69     uint16_t checksum;
70 };
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
625
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
15
```

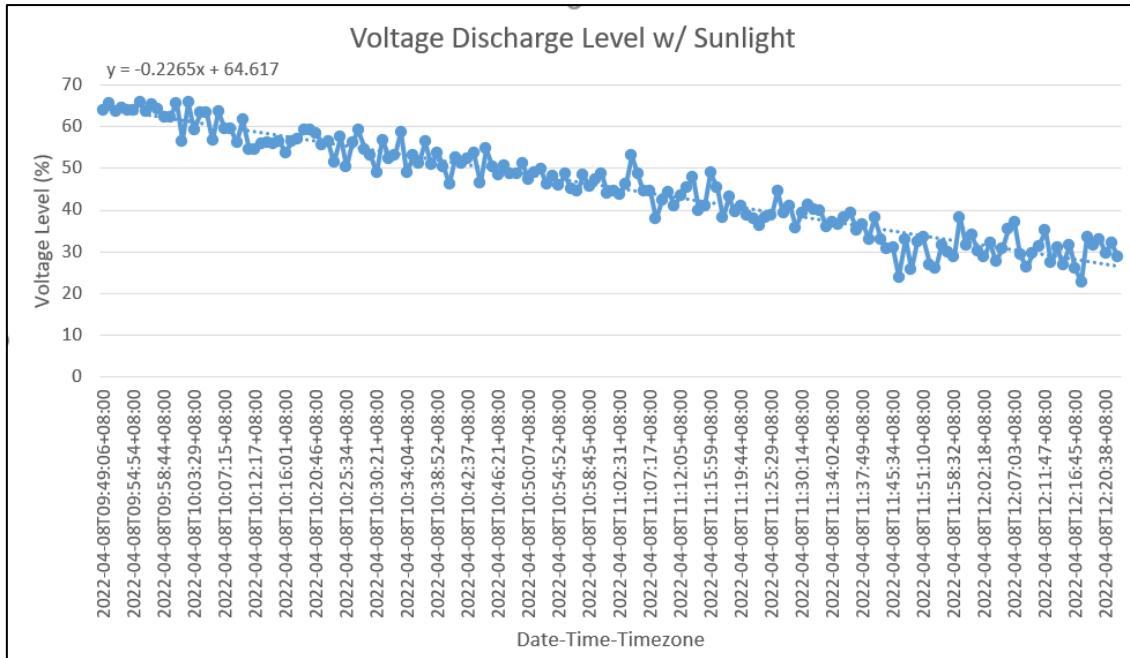


Figure 5.3.2: Voltage Discharge Rate with Sunlight

The first discharge test was done without any sunlight. Referring to figure 5.11 above, a trendline was generated and the gradient of the line is -0.2911, this means that the discharge rate is approximately 0.4366% per minute. The second discharge test was done under clear skies with a good amount of sunlight. Referring to figure 6.12 above, the trendline has a gradient of -0.2265, this means that the discharge rate is roughly 0.3397% per minute.

According to the test results, the solar panel has reduced the discharge rate by 0.1% per minute. This result is not ideal if the device is meant to be self-sufficient. This is mitigated by increasing the delay between collecting data to allow this device to work longer before battery depletion. Counter measures like a low battery notification to the user would allow the device to last longer.

## **CHAPTER 6 System and Hardware integration**

### **6.1 Subsystems in the project**

The final system consisted of several different subsystems. The subsystems in the air quality monitoring systems were:

#### **6.1.1 Sensors**

The first and most crucial subsystem in the air quality monitoring system is the group of sensors. As mentioned before, 4 different sensors were used for this purpose: MQ135, MQ3, MQ7, PMS5003.

#### **6.1.2 Power sub-system**

The power subsystem comprises two main components: The power harvesting system and the power storage and management system. The use of a solar panel was incorporated to extend the battery life of the air quality monitoring system outdoors.

#### **6.1.3 Air intake system**

A DC fan was fitted on the enclosure of the air quality monitoring system to uniformly sample the air preset outside. This was done so that the sensors sitting inside the device get enough air to be sampled for accurate readings, and non uniform air flow does not compromise accuracy of the sensors.

#### **6.1.4 ESP32 microcontroller**

The ESP32 microcontroller is used for two main purposes. The first is to convert the analog or I2C bus protocol signals into numerical values that can be understood by everyone. And later is to act as a IoT node by pushing in the live stream of processed data into the cloud which in our case is ThinkSpeak.

### **6.2 Testing of working of individual components before integration**

Before all the components were put together and integrated, each component was checked thoroughly to reduce troubleshooting time. Components were extensively tested on the arduino board first followed by the ESP32 controller to fix any problems. A breadboard was used to make the connections and built rapid prototypes. This methodology allowed the team to detect a lot of problems early on that could have caused an issue, For example, it was found that the MQ7 sensor that was originally purchased for the project was incompatible with the ESP32 as it gave analog values higher than the ESP's 3.3 V threshold. Hence a new MQ7 sensor was bought which was compatible with the ESP32. Several other issues were detected early, similarly.

As components passed the checks and worked correctly, we added more and more components to the ESP32 board to increase complexity.

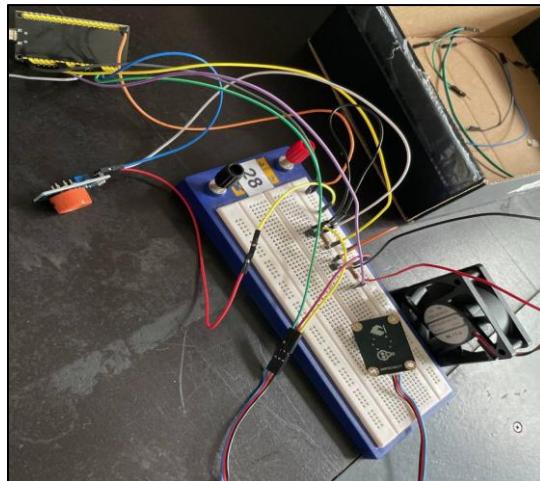


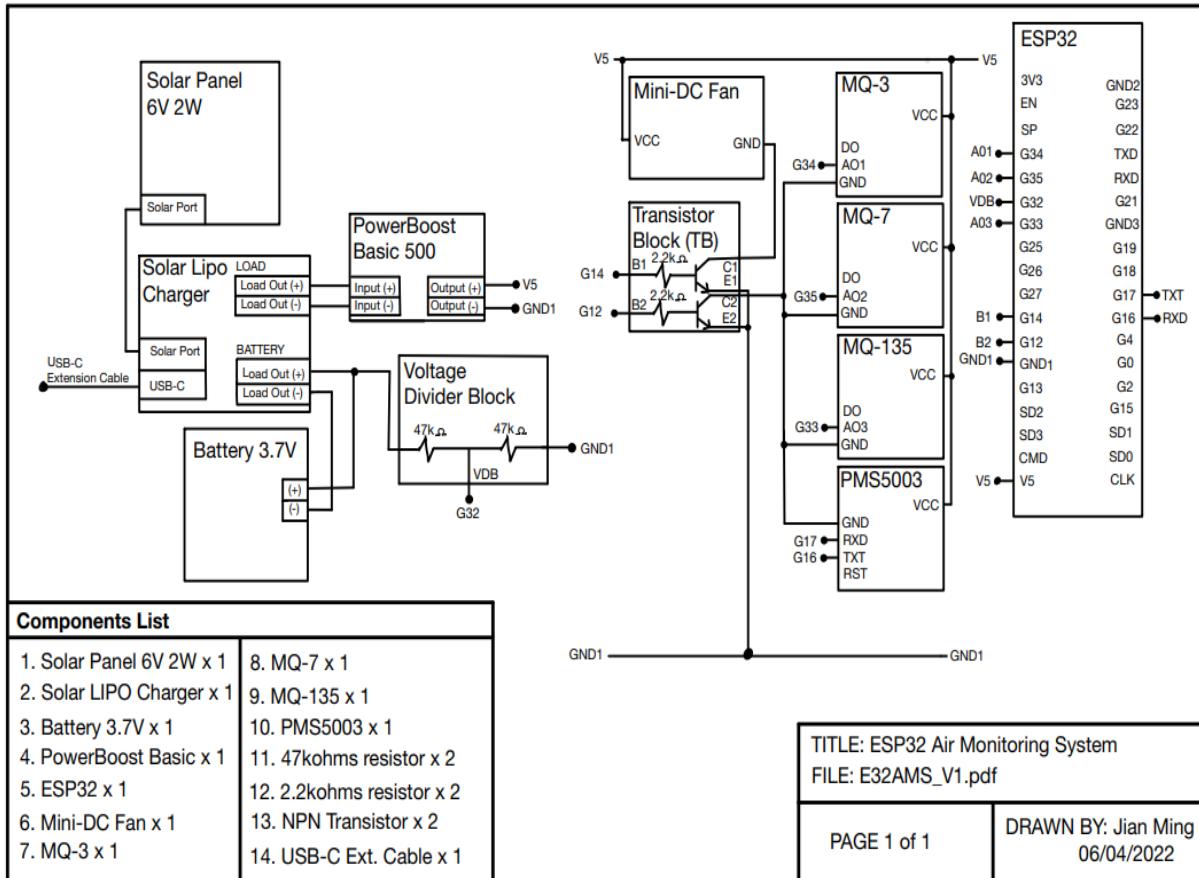
Figure 6.2.1: An early testing prototype with two sensors and a fan connected to the ESP32

### 6.3 Addition of the power system

Once all the components worked well with the power of the ESP, the next step was to make the prototype work with power from the battery and the power harvesting system. While the sensor team integrated the sensors together, the power team had the complete power setup ready. Hence a prototype consisting of a few sensors, a fan, and the battery connected to ESP32 was built. At this stage, we also added transistors to control the power flow from the ESP32 to all its peripherals, i.e. the sensors and DC fan. The transistors behaved as switches and helped regulate power usage.

### 6.4 Schematic design

Several prototypes in which the designs differed slightly, i.e. the ports used on the ESP32 differed and there were a few other changes made as well. For example, in the end the transistor switch for the Fan and the sensors was separated, as the fan needed to be on for a shorter time as compared to the sensors. A schematic was drawn to lay out the necessary connections that had to be made.



*Figure 6.4.1: Schematic of the finalised design*

## 6.5 Final test on breadboard

After the design for the schematic was finalized, the connections were realized on the breadboard for the last time to ensure there are no errors in the connections or components. The breadboard prototype was tested, and all the sensor values were checked again. During one of the tests, it was found that when the transistors were added to the sensor, the output voltage of the transistor changes. This in turn called for calibration of the sensors. Hence depending on the new voltage changes in clean air, the sensors had to be recalibrated.

## 6.6 Hardware integration into the enclosure box

The last step of the hardware connection was to realize all the connections inside the enclosure box. Firstly, the ESP32 board was soldered onto a prototyping board. It was decided that the sensor connection will not be soldered and instead made using pin connectors. This was done to ensure easy debugging in case of failure. The transistor switch and the other connections were soldered onto the board and finalized.

Once the required soldering was done, parts were stuck to the enclosure using blue tack and a hot glue gun depending on which part was being stuck. This was done to hold all the components in place. The final enclosure was then tested for 24hours to see how well it was working.

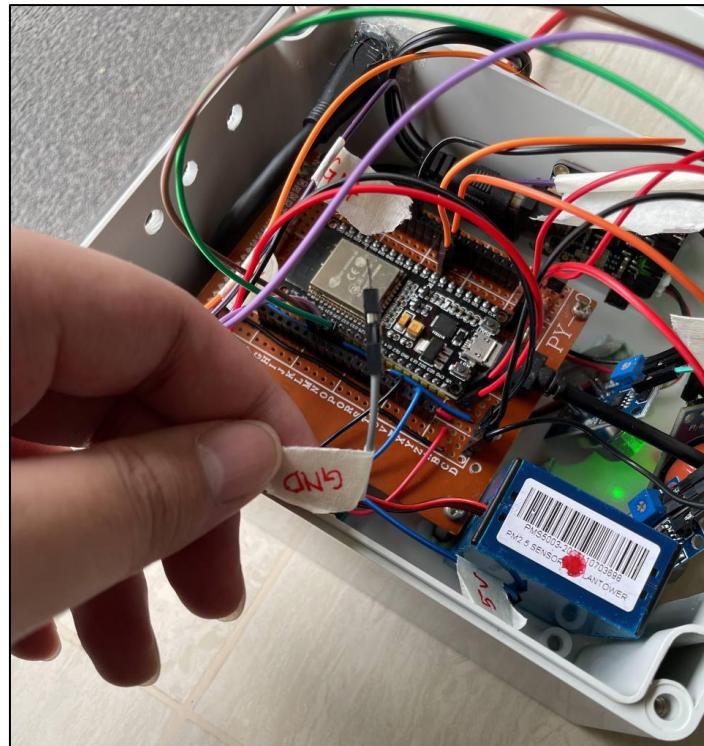


Figure 6.6.1 :Final prototype after all subsystems are integrated

## **CHAPTER 7 Thingspeak IoT platform**

ThingSpeak is a cloud-based Internet of Things (IoT) platform that enables users to gather, analyze and visualize live data. It also offers analytics tools that can be utilized to evaluate data to discover trends, relationships, and patterns across multiple data sets and display it via charts, gauges and plots.



Figure 7.0.1: ThingSpeak logo

In order to start sending sensor data to the ThingSpeak server, a channel must first be created. For this project, the team would like to analyse the values of Carbon Monoxide, Alcohol, Carbon Dioxide, PM2.5 and PM10 from ESP32. As such, it will require 5 different fields.

**Channel Settings**

Percentage complete 50%

Channel ID	1691187
Name	ESP32 Air Quality Monitoring
Description	Utilising data from our Particle Detector for Air Quality Monitoring to visualize and analyze the 5 parameters
Field 1	CO <input checked="" type="checkbox"/>
Field 2	Alcohol <input checked="" type="checkbox"/>
Field 3	CO2 <input checked="" type="checkbox"/>
Field 4	PM2.5 <input checked="" type="checkbox"/>
Field 5	PM10 <input checked="" type="checkbox"/>

Figure 7.0.2: Channel settings of thingspeak

Next, insert the ‘Write API key’ and channel number in the Arduino code. Both details are essential to update the channel with live data from the sensors.

**Write API Key**

Key

**Read API Keys**

Key

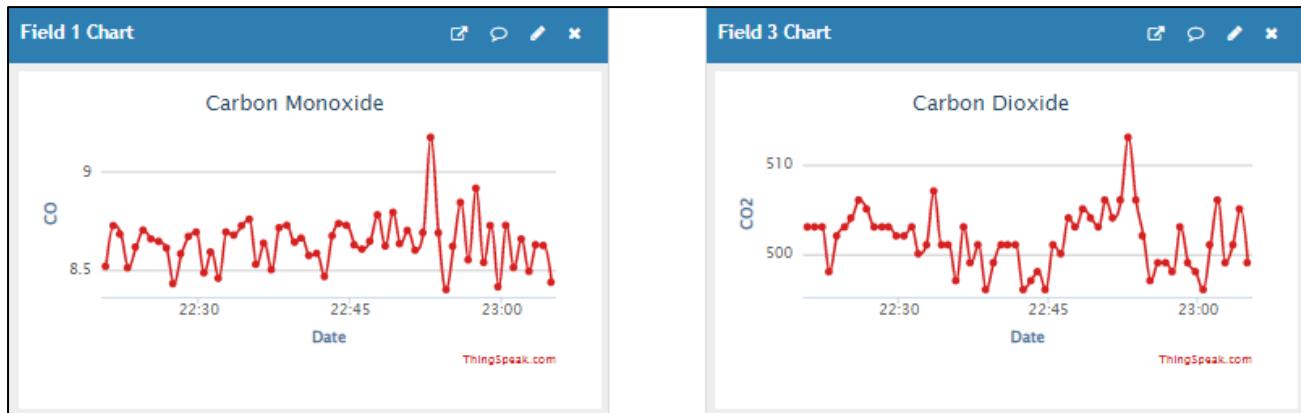
Note

```
WiFiClient client;

unsigned long myChannelNumber = 2;
const char * myWriteAPIKey = "QMJKFKVXHZT21Z89";
```

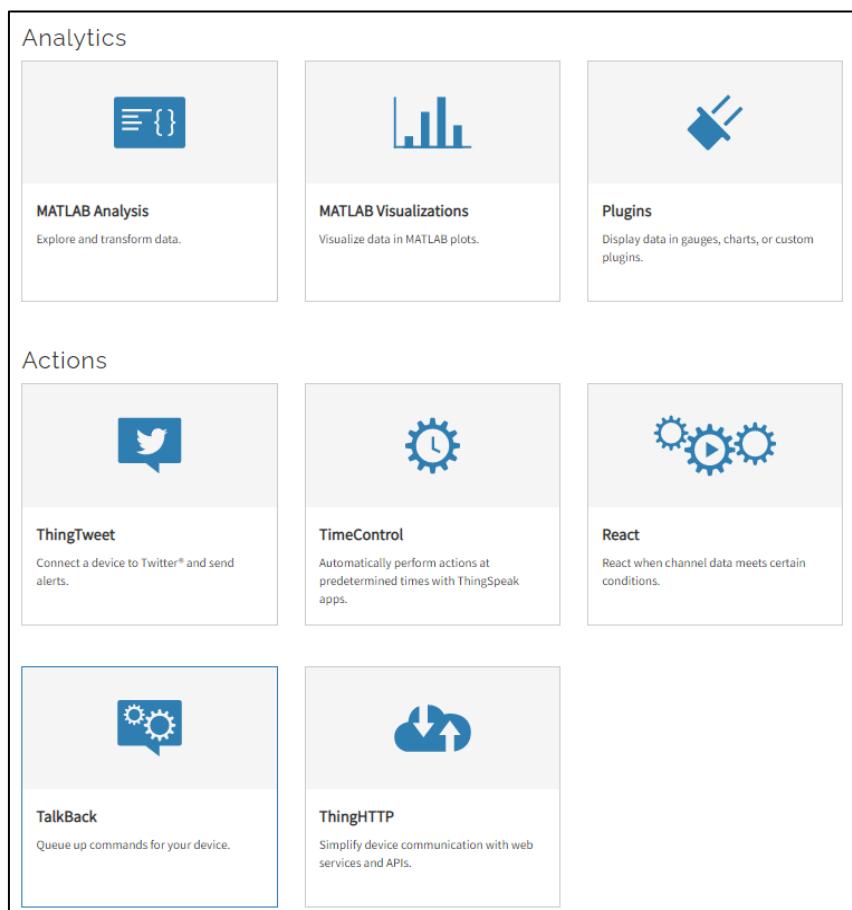
Figure 7.0.3: Usage of API Key and Channel number

After uploading the full Arduino code to ESP32 which includes WiFi connectivity, sensor codes and Thingspeak details, the real-time data of all parameters can now be visualized on Thingspeak.

*Figure 7.0.4: Data visualisation*

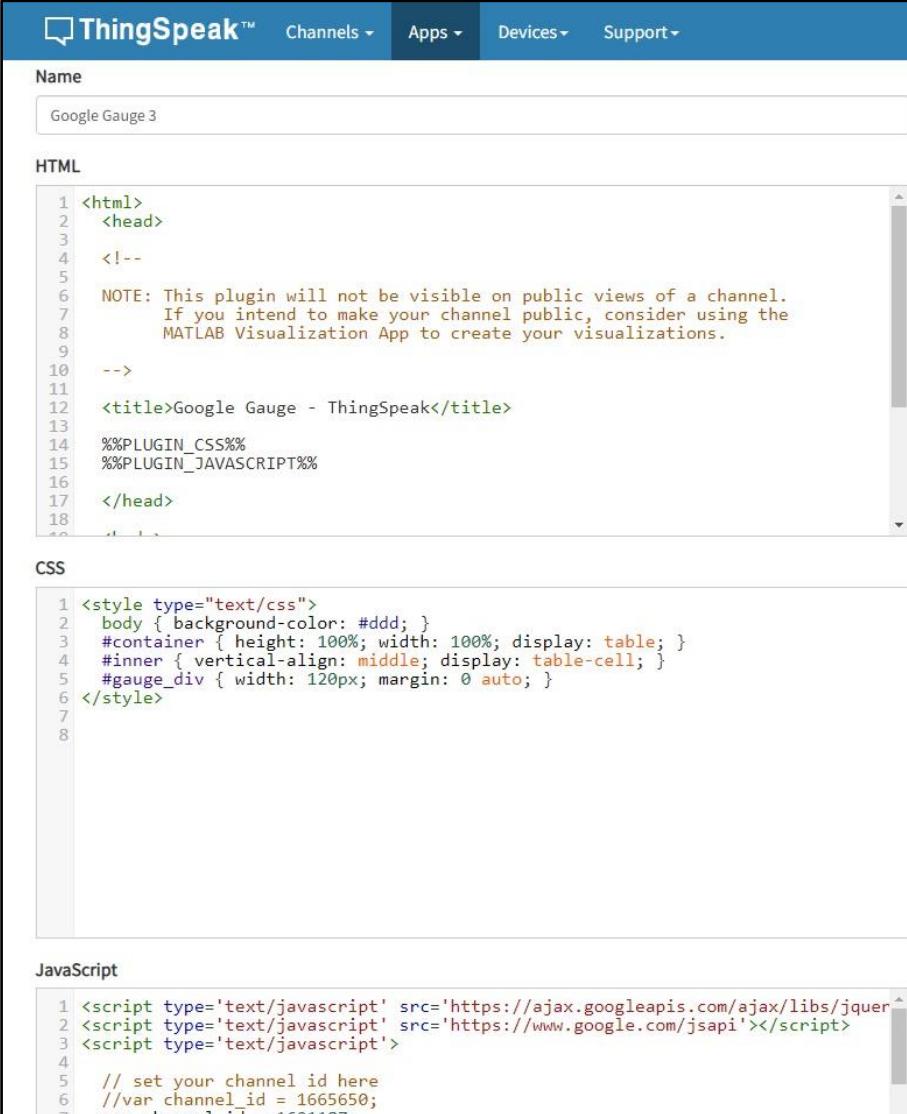
## 7.1 Applications of Thingspeak

Thingspeak provides a wide variety of applications that can be used to visualise data, trigger actions and communicate with third-party services. We will be introducing the applications that have been used throughout the project.

*Figure 7.1: Applications of Thingspeak*

## Plugins

Besides using the visualisation tools provided on ThingSpeak, the platform also gives users the flexibility of utilising the Plugins feature to customise gauges, charts or displays using HTML, CSS and JavaScript. To further enhance the data visualization, google gauges will be created to display the current values.



The screenshot shows the ThingSpeak web interface with the 'Plugins' section open. A new plugin is being created with the name 'Google Gauge 3'. The configuration is divided into three main sections: HTML, CSS, and JavaScript.

**HTML:**

```

1 <html>
2   <head>
3     <!--
4       NOTE: This plugin will not be visible on public views of a channel.
5         If you intend to make your channel public, consider using the
6         MATLAB Visualization App to create your visualizations.
7     -->
8   <title>Google Gauge - ThingSpeak</title>
9
10  %%PLUGIN_CSS%%
11  %%PLUGIN_JAVASCRIPT%%
12  </head>
13
14
15
16
17
18

```

**CSS:**

```

1 <style type="text/css">
2   body { background-color: #ddd; }
3   #container { height: 100%; width: 100%; display: table; }
4   #inner { vertical-align: middle; display: table-cell; }
5   #gauge_div { width: 120px; margin: 0 auto; }
6 </style>
7
8

```

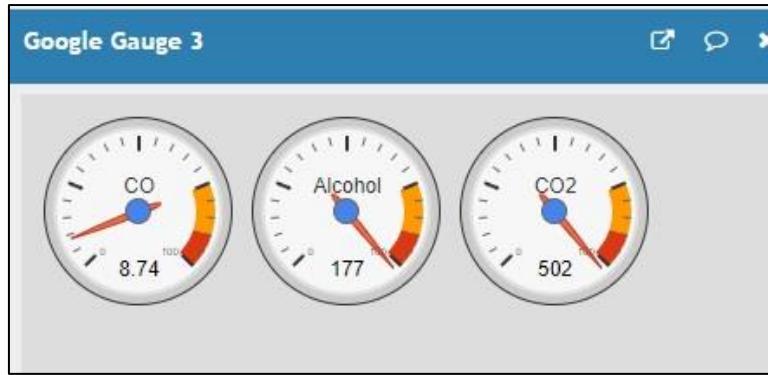
**JavaScript:**

```

1 <script type='text/javascript' src='https://ajax.googleapis.com/ajax/libs/jquery/
2 <script type='text/javascript' src='https://www.google.com/jsapi'></script>
3 <script type='text/javascript'>
4
5   // set your channel id here
6   //var channel_id = 1665650;
7   var channel_id = 1691187;

```

*Figure 7.1.1: Google Gauge code*



*Figure 7.1.2: Google Gauges*

## ThingHTTP

ThingHTTP enables communication among devices, websites, and web services without having to implement the protocol on the device level. Actions specified in ThingHTTP can be triggered by the ThingSpeak Application – React. This application is essential for the project's integration of alerts.

## 7.2 Alerts

Alerts are necessary to disseminate air quality information especially when there are a lot of pollutants present. It can be effectively done through SMS and social media platforms. As such, the team will be providing these alert options.

## Twilio

Twilio is a communications platform that allows users to integrate voice and SMS functionality into their apps. It accomplishes this by making APIs available to authenticate to their services. To begin, create an account and purchase a phone number with the given starting credits.

The screenshot shows the 'Account Info' section of the Twilio dashboard. It includes fields for 'Account SID' (ACe848a0d1b7ead6a4e477e779e13a0106), 'Auth Token' (redacted), and 'My Twilio phone number' (+19362262264). A note at the bottom says 'Always store your token securely to protect your account. [Learn more](#)'.

*Figure 7.2.1: Twilio account info*

Next, insert the Account SID and Auth Token in the ThingHTTP application. After keying in the required details on ThingHTTP, users can personalize their message under 'Body' to send out important alerts.

The screenshot shows the 'Edit' screen for the 'Carbon Monoxide' app in ThingHTTP. The configuration includes:

- Name:** Carbon Monoxide
- API Key:** RV60MKS0NO29L1RA
- URL:** https://api.twilio.com/2010-04-01/Accounts/ACe848a0d1b7ead6a4e477e779e13a0106
- HTTP Auth Username:** Account SID
- HTTP Auth Password:** Auth Token
- Method:** POST
- Content Type:** application/x-www-form-urlencoded
- HTTP Version:** 1.1
- Host:** (empty)
- Headers:**
  - Name: (empty)
  - Value: (empty)
  - remove header
- Body:** From=+Twilio number &To=+%Personal phone number&Body=Unhealthy air detected  
CO Level: %%channel\_1650666\_field\_1%%

*Figure 7.2.2: ThingHTTP settings*

## ThingTweet

Twitter is an extremely popular social media application that boasts up to 217 million daily users worldwide as of Q4 2021. ThingTweet application can be used to directly link a Twitter account to a ThingSpeak account to post automatically and reach out to many users. For example, we can tweet out the consolidated parameters on Twitter, in an interval of 10 minutes to consistently keep individuals updated.

## React

React application is used together with ThingHTTP and ThingTweet to perform actions when the channel data meets a certain condition. The conditions are set based on the threshold value that is predetermined in the table below, to trigger the SMS alert function via Twilio and send out tweets to inform users that the air quality of a place is unhealthy at the moment so that they can avoid it.

Type of Particles	Threshold Level
Carbon Monoxide	70 ppm
Alcohol(Ethyl Alcohol)	1000 ppm
Carbon Dioxide	5000 ppm
PM2.5	151 $\mu\text{g}/\text{m}^3$
PM10	351 $\mu\text{g}/\text{m}^3$

Figure 7.2.3: Threshold table

The left panel shows the configuration of a Twilio 'React' rule. The 'React Name' is 'Carbon Monoxide'. The 'Condition Type' is 'Numeric'. The 'Test Frequency' is 'On Data Insertion'. The 'Condition' is set to 'If channel' with 'ESP32 Air Quality Monitoring (1691187)' selected. The 'field' is '1 (CO)', the operator is 'is greater than', and the value is '70'. The 'Action' is 'ThingHTTP' with 'Carbon Monoxide' selected. Under 'Options', the radio button for 'Run action each time condition is met' is selected. The right panel shows an incoming text message from '+1 (936) 226-2264' with the subject 'Text Message' and timestamp 'Today 12:28 AM'. The message content is: 'Sent from your Twilio trial account - Unhealthy air detected CO Level: 88'.

*Figure 7.2.4: React settings and SMS received*

The left panel shows the configuration of a ThingTweet app. It has a 'Twitter Account' table with one entry: 'AirQuality\_E016' and 'API Key' 'ONIDOTCLHB8BOV4VP'. Buttons for 'Regenerate API Key' and 'Unlink Account' are present. Below is a table titled 'Reacts using ThingTweet' showing a single row for 'Twitter' with the message: 'CO level is: 8.43899 %channel\_1691187\_field\_1% Alcohol level is: %channel\_1691187\_field\_2% CO2 level is: %channel\_1691187\_field\_3% PM2.5: %channel\_1691187\_field\_4% PM10: %channel\_1691187\_field\_5% Date and time measured: %datetime%' and 'Last Sent' '2022-04-09 00:31'. The 'Twitter Account' is 'AirQuality\_E016'. The right panel shows a Twitter post by 'Air Quality Monitoring @AirQuality\_E016 · 1h' with the same message and additional details: 'CO level is: 8.43899', 'Alcohol level is: 107', 'CO2 level is: 499', 'PM2.5: 23', 'PM10: 15', and 'Date and time measured: 2022-04-08 11:01 pm'. The tweet has standard social media interaction icons below it.

*Figure 7.2.5: Thingtweet settings and the Tweet*

### 7.3 Data Analysis and AI

Data analysis is the process of evaluating data using analytical tools to extract useful information to identify trends or patterns while artificial intelligence can be used for the prediction of future values based on past data.

## Pairplot

PairPlot consists of scatterplot and histogram on the diagonal plots. It is used to visualize multivariate data to understand the relationships among the 5 parameters. From the plot below, it can be seen that there is a strong relationship between PM2.5 and PM10. This form of visualization is achieved by using the data from our channel in the gplotmatrix function.

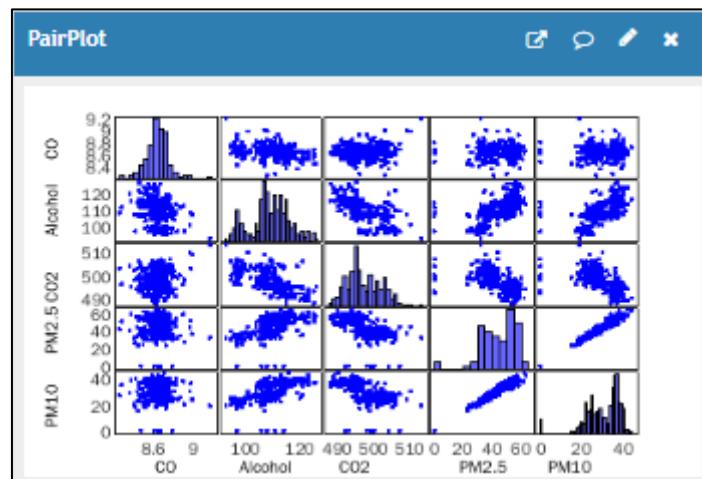


Figure 7.3.1: Pairplot

Name	PairPlot
MATLAB Code	<pre> 1 % Channel ID to read data from 2 readChannelID = 1691187; 3 COfieldID = 1; 4 AlcoholfieldID = 2; 5 CO2FieldID = 3; 6 PM25FieldID = 4; 7 PM10FieldID = 5; 8 9 % Channel Read API Key 10 % If your channel is private, then enter the read API 11 % Key between the '' below: 12 readAPIKey = ''; 13 data = thingspeakRead(readChannelID,'Fields',[COfieldID AlcoholfieldID CO2FieldID PM25FieldID 14   'NumPoints',300, 15   'ReadKey',readAPIKey); 16 % Read all Data 17 COData = data(:,1); 18 AlcoholData = data (:,2); 19 CO2Data = data (:,3); 20 PM25Data = data(:,4); 21 PM10Data = data(:,5); 22 X =[COData,AlcoholData,CO2Data,PM25Data,PM10Data]; 23 figure 24 gplotmatrix(X); 25 varNames = {'CO'; 'Alcohol'; 'CO2'; 'PM2.5'; 'PM10'}; 26 text([.08 .24 .43 .66 .83], repmat(-.1,1,5), varNames, 'FontSize',8); 27 text(repmat(-.12,1,5), [.86 .62 .41 .25 .02], varNames, 'FontSize',8, 'Rotation',90); </pre>

Figure 7.3.2: Pairplot code

## Cleaning of Data

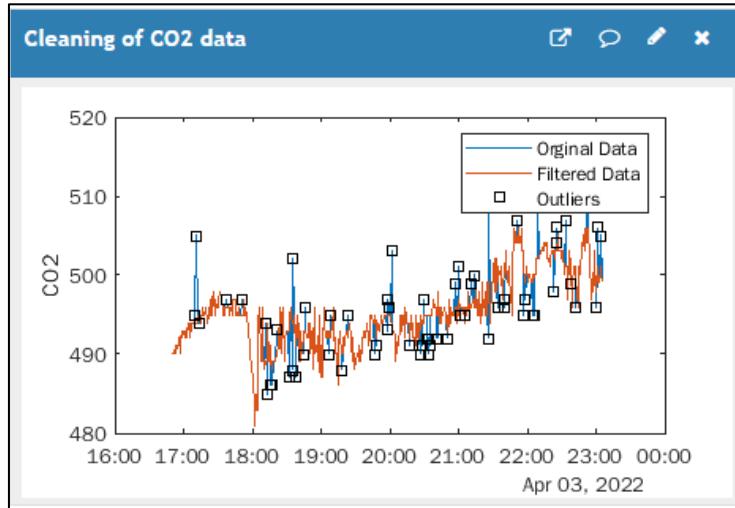


Figure 7.3.3: Visualisation of CO2 data cleaning

The code below shows how outliers are detected and removed from our channel data using a Hampel filter. This filter is closely related to a median filter, and it helps to remove outliers from a signal without overly smoothing the data.

Start by reading the CO2 data from our channel by using the thingSpeakRead function. Next, remove the outliers in the raw data by applying the Hampel function. By setting a window size of 6 and a median of 2, it allows for sufficient data to decide whether each point is an outlier. The number of standard deviations has been set to be 2 which means that if a data point differs from the local median by more than this number, it will be replaced with the local median value. By plotting the original data and the filtered data, the outliers that are detected and removed can be visualized.

The screenshot shows a MATLAB application window titled "Cleaning of CO2 data". The "Name" field contains "Cleaning of CO2 data". The "MATLAB Code" section displays the following script:

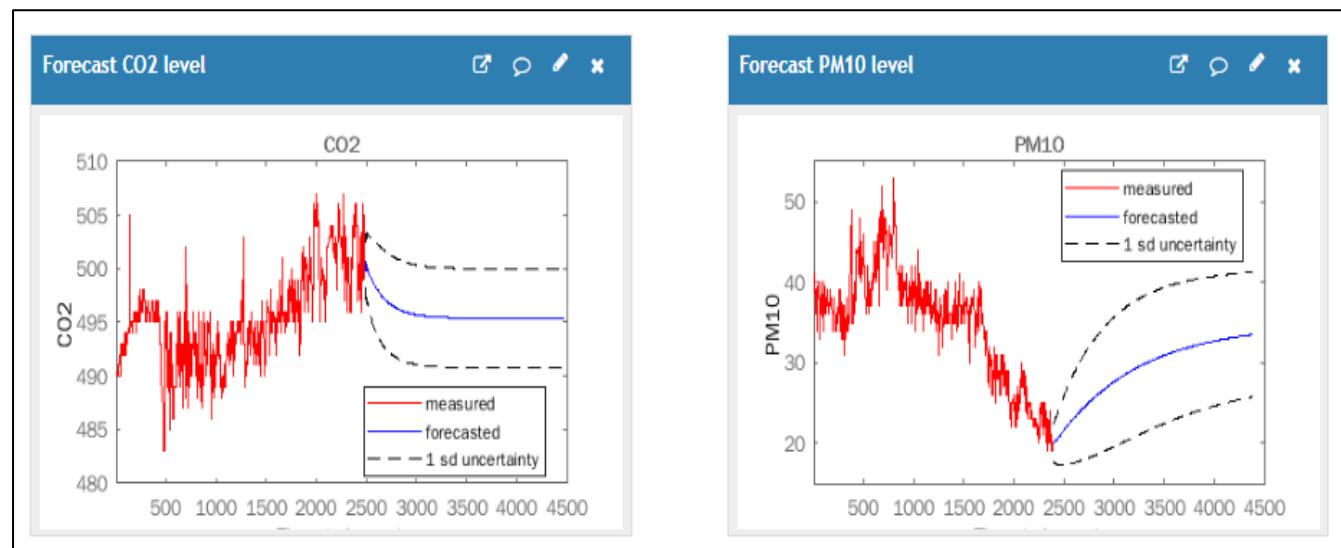
```

1 % Enter your MATLAB code below
2
3 data = thingSpeakRead(1691187,'NumPoints',800,'Fields',3,'outputFormat','table');
4 windowSize = 6;
5 numMedians = 2;
6 [filteredData,outliers]=hampel(data.CO2,windowSize,numMedians);
7 plot(data.Timestamps,data.CO2);
8 hold
9 plot(data.Timestamps,filteredData);
10 plot(data.Timestamps(outliers),data.CO2(outliers),'Marker','Square','LineStyle','None','Color','red');
11 ylabel('Alcohol');
12 legend('Orginal Data', 'Filtered Data','Outliers');

```

*Figure 7.3.4: CO2 data cleaning code*

## Forecasting

*Figure 7.3.5: Visualisation of forecasting*

Forecasting air quality helps people to plan ahead, decreasing the effects on health and the costs associated. The code below shows how forecasting is done for CO2 level by using the data from our channel.

Start by retrieving the CO2 data within a date range and fit a model to the data. Since the parameter varies with time, an autoregressive model was chosen. Next, use the forecast function to predict future values of CO2 level for the next day. Afterwards, the plot function is used to visualize the measured and forecasted value of up to 1 standard deviation of uncertainty.

```

Forecast CO2 level

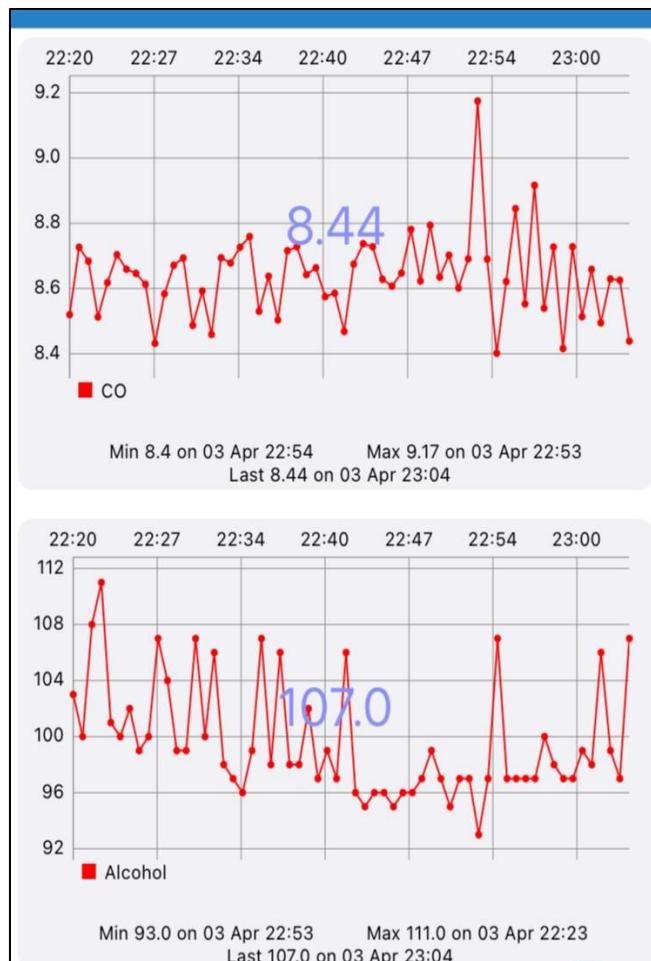
MATLAB Code

1 startDate = datetime('March 31, 2022 10:00:00 AM');
2 endDate = datetime('April 01, 2022 10:00:00 AM');
3 dateRange = startDate:endDate;
4 data = thingSpeakRead(1691187,'DateRange',dateRange,'Fields',3);
5 sampleTime = 5;
6 IDdata = iddata(data,[],sampleTime,'OutputName',{'CO2'},'TimeUnit','minutes')
7 trend = getTrend(IDdata,0);
8 IDdata = detrend(IDdata,0);
9 modelOrder = 8;
10 sys = ar(IDdata,modelOrder);
11 numSamples = 400;
12 [yf,x0,sysf,yf_sd,x,yx_sd] = forecast(sys,IDdata,numSamples);
13 IDdata = retrend(IDdata,trend);
14 yf = retrend(yf,trend);
15 figure;
16 UpperBound = iddata(yf.OutputData+1*yf_sd,[],yf.Ts,'Tstart',yf.Tstart,'TimeUnit','minutes');
17 LowerBound = iddata(yf.OutputData-1*yf_sd,[],yf.Ts,'Tstart',yf.Tstart,'TimeUnit','minutes');
18 plot(IDdata(:, :, []), 'r', yf(:, :, []), 'b');
19 hold on
20 plot(UpperBound,'k--',LowerBound,'k--');
21 legend({'measured','forecasted','1 sd uncertainty'},'Location','best');
22 xlabel('Time');
23 ylabel('CO2');
24
25
26 title('Measured and Forecasted CO2 level');
```

Figure 7.3.6: CO2 Forecasting code

## 7.4 ThingView

Another benefit of using Thingspeak is that it has a mobile application known as Thingview, which gives users another alternative method to visualize the data from our channel in a simple way. By keying in our channel ID in the app, users can immediately view our channel data.



*Figure 7.4.1: Thingview platform*

## CHAPTER 8 Schedule

Planning/ Week	1	2	3	4	5	6	7	8	9	10	11	12	13
Research on Sensors				✓									
Risk Assessment			✓										
Project Charter				✓									
Purchasing of parts				✓									
<b>Execution</b>													
Calibration of sensors + testing MQ-7 MQ-135									✓				
Arduino programming Process data from sensor									✓				
IOT Dashboard Data Transmission & visualization									✓				
Power management Energy harvesting Energy Saving										✓			
Integration Sensor and Arduino Programming Arduino and IOT <del>IOT and Power Management</del> Making a prototype enclosure												✓	
<b>Testing</b>												✓	
Demonstration of prototype													Active Go to

Everyone
Sensor group
IOT group
Power group
Denotes actual completion milestone

Figure 8.1: Project Gantt Chart with Actual Completion Milestone

Most of the project milestones were met on time except for the execution phase due to some unforeseen circumstances. Experimental/ academic use sensors give output voltage in the range from 0-5V with respect to the level of intensity of the measured parameter. However, due to our low-power microcontroller ESP32 only recognizes input value range 0-3.3V which causes some erratic data values once the input voltage level exceeds the upper limit. Hence, the team spent extra time troubleshooting the problem and propose a few solutions to tackle it including the implementation of a voltage divider and re-tuning of the sensors. Besides that, some deliveries of new components have a longer lead time due to logistic issues which delayed the progress of the project. In the end, the team is still able to overcome the issues and new challenges imposed which we were able to complete without further delay.

## CHAPTER 9 Cost

Item description:	Quantity:	Price:	Total Cost:
<b>Power</b>			
Solar Panel	2	\$42.92	\$85.94
USB C extension cable type C male to female	1	\$9.90	\$9.90
1.1mm to 5.5 DC Jack Adapter Cable	2	\$2.22	\$4.44
<b>Sensors</b>			
MQ-7 Carbon Monoxide Sensor	2	\$17.50	\$35.00
MQ-3 Alcohol Sensor	2	\$3.00	\$6.00
MQ-135 Carbon Dioxide Sensor	2	\$3.00	\$6.00
PM2.5 and PM10 Air Quality Sensor - PMS5003	2	\$59.13	\$118.26
MIKROE-1630 Sensor	2	\$24.42	\$48.84
<b>Assembly</b>			
Super Glue	1	\$5.50	\$5.50
Enclosure	1	\$47.19	\$47.19
DC Fan	2	\$12.45	\$24.90

Our plans to incorporate sensors which we have deemed necessary to monitor the air quality include the carbon monoxide, alcohol, carbon dioxide, PM2.5 and PM10 sensors in an enclosed box design. Thus we set out and purchased the sensors and enclosure to proceed to build a prototype structure. Thereafter, we improved on this prototype by making it to be rechargeable by solar panels so that the sensors are able to obtain values and monitor the air outdoors without the need to be connected to a computer, giving the project the ability to be portable.

## CHAPTER 10 Project Outcomes

<b>Project Deliverables</b>	<b>Learning Outcomes</b>
Sensors calibrated up to the reference standard compatible with ESP32 microcontroller	Team has calibrated all the 4 sensors using the original manufacturer data sheet where calculation & appropriate extrapolation techniques were used to offset the sensor's output value. Besides, the team has also implemented voltage dividers for the output value from the sensors to tune down the range of voltage ratio values compatible with ESP32 voltage range (3.3V).
Power management from power harvesting to the power storage device.	A 6V 2W solar panel was used as a power harvesting device and regulated by a solar charger before charging up the LiPo battery. The team has analyzed the overall performance of the power module such as the time-to-charge, charging, and discharging characteristics of the battery.
Power saving mechanism & circuitry to increase the power utilisation.	As the theoretical data pulling/ updating rate was set to be every 15 minutes, constant powering up of the prototype would be unnecessary and a waste of power. Hence, algorithms such as hibernation mode with a timer to wake up allow ESP32 main modules such as WiFi & Bluetooth together with its peripherals to go into sleep mode for a preset period. Besides, transistors were also used as a switch to control the amount of time that will power the sensors up.
ESP32 microcontroller setting up & communication link to IoT cloud/ IoT platform, Thingspeak via Wifi.	Setting up of ESP32 WiFi function requires Arduino programming through Arduino IDE. SSID and password have to be declared before association with the intended WiFi network. Coding of the microcontroller cloud service and calling of the IoT platform API must be done in the script. Once the communication link was established, The respective parameter will be

	mapped to the various fields in the IoT platform to collect the calibrated data values.
IoT data visualization, analysis & initial prediction model.	When data is sent to the IoT platform, the data will be first processed by removing the outliers before it gets tabulated into the population. With the set of values, we could invoke some visualizing widgets such as graphs, indicators, meters, etc to look at the trend of the data values. Going in deeper, we could analyze the obtained data values and relate them to the testing environment. A Matlab forecasting model was implemented with the input of a sufficiently large data population to have a more accurate prediction of the data trend.
IoT alerts when any of the air parameters exceed a healthy range or approach a dangerous level	Another feature of this project is such that users do not need to monitor the data values constantly which alerts will be triggered automatically when any of the parameters fall in the dangerous level. This feature requires hand-shaking through web application & API between the IoT platform, ThingSpeak, and a third-party messaging service provider, Twilio.
System & hardware integration	Towards the last phase of this project, we have to integrate all the sub-modules together, mainly the sensors, microcontroller, and the power harvesting & storage module. This step requires mostly synchronization in order to have all data values arrive at the same time and the sleep mode adjustment along with the peripherals. besides, designing of hardware in such a way that the end prototype is compact, accessible, and secure.

## **CHAPTER 11 Reflection**

Throughout an extensive 13 weeks of DIP engagement, various engineering knowledge and techniques were used. From circuit analysis and analog circuits on sensor integration with voltage regulating mechanism, transitioning to the wireless communication & algorithm on Internet of Things (IoT), ending with data analytic and visualization. We could see how far the engineering & technology sector has evolved over decades with the aim to serve the community and allow a better future for the generations to come. It brings convenience, rapid data exchanges, AI-driven forecasting, and smart sensing.

However, projects were meant to be challenging to allow breakthroughs, and even simple scenarios require complex algorithms to enhance the efficiency and the overall system reliability. This is where problem analysis comes into place. In our project context, how to ensure constant network coverage to achieve self-sustainable features? What are the trade-offs on the power module while ensuring the network coverage? And to tackle these issues, we often need to leverage and find the balance point which optimizes all of the variables. Looking at a problem from different angles, and performing complete analysis on various approaches to determine if it's the best fit for the problem.

Our group consisted of 7 members. As we did not know each other prior to DIP, each of the member's attitudes and working styles constituted the overall team performance. For a team project to be successful, the amalgamation of individuals with different backgrounds, domain experience, and knowledge through proper coordination & effective communication plays a very crucial part to maximize the useful inputs to the project. We divided ourselves into 3 sub-groups (sensor, IoT, and power) to invoke deeper discussion within the sub-modules and enhance hardware implementation. Besides that, the team manages to have a high level of awareness where it cultivates a positive atmosphere for every member. With this, each member would have a better sense of belonging where everyone could fit in, be respected despite differences in various aspects.

Moving forward, we could foresee this project has a very good potential to stretch further and improve. First of all, the microcontroller used could be replaced with a stronger alternative where the power efficiency, scalability, compatibility, transmission bandwidth, etc were greatly enhanced but at a higher cost. Besides, artificial intelligence & machine learning could also be implemented in our project with the goal to study the correlation between each parameter and predict the instances where the air quality could be compromised or early prediction for excessive emission of hazardous air particles. Last but not least, the power harvesting & management of this project can be reviewed and improved such that the prototype could achieve self-sustainability in an indoor environment where solar power is limited.

In conclusion, we were grateful to have this opportunity to work on this DIP project and we believe that the skills that we'd learned through the project would be utilized in the near future.

## References

- [1] Article on air pollution. 2018 (WHO)  
[http://www.emro.who.int/media/news/9-out-of-10-people-worldwide-breathe-polluted-air.html#:~:text=New%20data%20from%20the%20World,household%20\(indoor\)%20air%20pollution](http://www.emro.who.int/media/news/9-out-of-10-people-worldwide-breathe-polluted-air.html#:~:text=New%20data%20from%20the%20World,household%20(indoor)%20air%20pollution)
- [2] Infographics on air pollution. 2018, ET EnergyWorld.  
<https://energy.economictimes.indiatimes.com/news/coal/9-out-of-10-people-worldwide-breathe-polluted-air-who/66416570>
- [3] Comparison on Arduino Uno vs Esp32  
[https://www.tutorialspoint.com/arduino\\_uno\\_vs\\_esp32](https://www.tutorialspoint.com/arduino_uno_vs_esp32)
- [4] ESP32  
<https://en.wikipedia.org/wiki/ESP32>
- [5] Esp32 power management  
<https://www.mischianti.org/2021/03/06/esp32-practical-power-saving-manage-wifi-and-cpu-1/>
- [6] PMS5003 datasheet  
[https://www.sgbotic.com/products/datasheets/sensors/plantower-pms5003-manual\\_v2-3.pdf](https://www.sgbotic.com/products/datasheets/sensors/plantower-pms5003-manual_v2-3.pdf)
- [7] PMS5003 information  
<https://how2electronics.com/interfacing-pms5003-air-quality-sensor-arduino/>
- [8] PSI and PPM information  
[https://www.haze.gov.sg/docs/default-source/faq/computation-of-the-pollutant-standards-index-\(psi\).pdf](https://www.haze.gov.sg/docs/default-source/faq/computation-of-the-pollutant-standards-index-(psi).pdf)
- [9] MQ-7 datasheet  
<https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf>
- [10] MQ-3 datasheet  
<https://www.sparkfun.com/datasheets/Sensors/MQ-3.pdf>
- [11] Computation of PSI  
[https://www.haze.gov.sg/docs/default-source/faq/computation-of-the-pollutant-standards-index-\(psi\).pdf](https://www.haze.gov.sg/docs/default-source/faq/computation-of-the-pollutant-standards-index-(psi).pdf)
- [12] Alcohol level  
<https://www.nj.gov/health/eoh/rtkweb/documents/fs/0844.pdf>
- [13] Voltaic Solar Panel 6V 2W Specification  
[Voltaic Systems P102 R3A.pdf](Voltaic%20Systems%20P102%20R3A.pdf)
- [14] Solar Charger (BR-2489958) Schematic  
[adafruit\\_bq24074\\_universal\\_usb\\_dc\\_solar\\_charger\\_br-2489958.pdf \(mouser.sg\)](adafruit_bq24074_universal_usb_dc_solar_charger_br-2489958.pdf)
- [15] Solar Charger (BR-2489958) Schematic  
<adafruit-powerboost.pdf>
- [16] Li-Ion Discharge Curve  
<batteries - LiPoly Battery - When to stop draining? - Electrical Engineering Stack Exchange>
- [17] LP785060 Battery Specification  
[785060-2500mAh\\_specification\\_sheet.pdf \(adafruit.com\)](785060-2500mAh_specification_sheet.pdf)
- [18] Twitter statistics  
<https://www.statista.com/statistics/970920/monetizable-daily-active-twitter-users-worldwide/>

## Appendix A - Project Members Contribution

	Name	Project contributions	Report Contribution
1	Ho Kok Pin	<b>Group Leader,</b> <b>IoT Team:</b> Internet of Things (Interfacing ESP32 with Thingspeak), Hardware design & integration.	<b>Project Objectives;</b> <b>Project Schedule;</b> <b>Project outcome;</b> <b>Reflection.</b>
2	Kanishk Srivastav	Sensor team: Calibration of sensors; Power team: Power calculations; Hardware integration;	<b>MQ135 sensor;</b> <b>System and hardware integration;</b> <b>Power calculations.</b>
3	Joe Cheng Kim Wei	Treasurer(Procurement and claiming of invoices) Sensor team: Calibration of sensors Hardware integration	<b>MQ-7 sensor</b> <b>MQ-3 sensor</b> <b>PMS5003 sensor</b> <b>Connection schematic of MQ-135 sensors</b> <b>Costs of the Project</b>
4	Chua Beng Choon	IoT Team: Configuring ESP32, Thingspeak ( Alerts, Data Analysis and AI )	<b>Thingspeak IoT platform</b> <b>Applications of Thingspeak</b> <b>Alerts</b> <b>Data Analysis and AI</b> <b>Thingview</b>
5	Dhuwaragish Ravichandrakumar	IoT Team: Configuring ESP32 microcontroller, Interfacing ESP32 with ThingSpeak, ThingSpeak (Dashboard), Troubleshooting code  <b>Power Team: Power Saving</b>	<b>ESP32</b> <b>Power Saving Module</b>
6	Xu Jie	Sensor team Calibrations of sensors Hardware integration	<b>Project Scope and summary</b> <b>MQ-3 sensor</b> <b>Costs of the Project</b>
7	Chua Jian Ming	<b>Power Team:</b> Design and Implementation of Power Harvesting and Management System. Troubleshooting Connection Issues related to the power system. Design schematic diagram of device.	<b>Power Harvesting and Management System;</b> <b>Testing of Power System;</b> <b>Overall Schematic Diagram;</b>

## Appendix B – Full Code

```
#include "WiFi.h"
#include "ThingSpeak.h"
#include "MQ7.h"
#include <SoftwareSerial.h>
#include "MQ135.h"
#define VOLTAGE 5
#define uS_TO_S_FACTOR 1000000ULL /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 10 /* Time ESP32 will go to sleep (in seconds) */
const byte sensors_gpio = 12, fan_gpio=14;

MQ7 mq7(35, VOLTAGE);
SoftwareSerial pmsSerial(16, 17);

const char* ssid = "Rd";
const char* password = "123123123";

WiFiClient client;

unsigned long myChannelNumber = 2;
const char * myWriteAPIKey = "QMJFKVXHYZT21z89";

// Timer variables
//unsigned long lastTime = 0;
//unsigned long timerDelay = 30000;
float Rs,Ratio,analogCO2;
MQ135 gasSensor = MQ135(30);
float CarbonMonoxide;
int Alcohol;
int CarbonDioxide;
int PM25 = 0;
int PM10 = 0;
const int Analog_channel_pin=32;
int ADC_VALUE = 0;
float voltage_value = 0;

void print_wakeup_reason(){
    esp_sleep_wakeup_cause_t wakeup_reason;

    wakeup_reason = esp_sleep_get_wakeup_cause();
```

```
switch(wakeup_reason)
{
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
}

void setup() {

    pinMode(sensors_gpio, OUTPUT);
    pinMode(fan_gpio, OUTPUT);

    Serial.begin(115200);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Connecting to WiFi..");
    }

    Serial.println("Connected to the WiFi network");
    digitalWrite(sensors_gpio, HIGH);
    delay (5000);
    ThingSpeak.begin(client); // Initialize ThingSpeak
    mq7.calibrate(); // calculates R0
    pmsSerial.begin(9600);

}

struct pms5003data {
    uint16_t framelen;
    uint16_t pm10_standard, pm25_standard, pm100_standard;
    uint16_t pm10_env, pm25_env, pm100_env;
    uint16_t particles_03um, particles_05um, particles_10um, particles_25um, particles_50um, particles_100um;
    uint16_t unused;
    uint16_t checksum;
};
```

```

82
83 void loop() {
84
85     digitalWrite(fan_gpio, HIGH);
86     delay (10000);
87
88     analogCO2 = analogRead(33);
89     //CarbonDioxide = 624650/analogCO2;
90     analogCO2 = 3.30000*analogCO2/4096.0000;
91     Rs= (5/analogCO2)-1;
92     Ratio=Rs/94.96;
93     Ratio= pow (Ratio,0.7);
94     Ratio= Ratio*701;
95     CarbonDioxide= Ratio*10;
96     Serial.print("CO2\t"); Serial.println(CarbonDioxide,DEC);
97
98     Alcohol= _analogRead[34];
99     Serial.print("\nAlcohol\t"); Serial.println(Alcohol,DEC);
100
101    CarbonMonoxide= mq7.readPpm();
102    Serial.print("CO\t"); Serial.println(CarbonMonoxide,DEC);
103
104    if (readPMSSdata(&pmsSerial)) {
105        (data.pm10_standard);
106        (data.pm25_standard);
107
108        PM25 = data.pm25_standard;
109        PM10 = data.pm10_standard;
110        Serial.print("PM2.5\t"); Serial.println(PM25,DEC);
111        Serial.print("PM10\t"); Serial.println(PM10,DEC);
112
113    }
114
115    ADC_VALUE = analogRead(Analog_channel_pin);
116    Serial.print("ADC VALUE = ");
117    Serial.println(ADC_VALUE);
118    voltage_value = 100*(( (ADC_VALUE * 3.3) / (4095) *2 ) - 3 ) /1.2);

```

```

119   serial.print("voltage"); serial.println(voltage_value);
120
121   digitalWrite(sensors_gpio, LOW);
122
123   ThingSpeak.setField(1, CarbonMonoxide);
124   ThingSpeak.setField(2, Alcohol);
125   ThingSpeak.setField(3, CarbonDioxide);
126   ThingSpeak.setField(4, PM25);
127   ThingSpeak.setField(5, PM10);
128   ThingSpeak.setField(6, voltage_value);
129
130
131
132
133   int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
134   if(x == 200){
135     Serial.println("Channel update successful.");
136     esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
137     Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
138     " Seconds");
139     //esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
140     //Serial.println("Configured all RTC Peripherals to be powered down in sleep");
141     Serial.println("Going to sleep now");
142     Serial.flush();
143     esp_deep_sleep_start();
144     Serial.println("This will never be printed");
145   }
146   else{
147     Serial.println("Problem updating channel. HTTP error code " + String(x));
148   }
149   //digitalWrite(sensors_gpio, LOW);
150   //delay (10000);
151   //lastTime = millis();
152 }
153
154 boolean readPMSdata(Stream *s) {
155   if (! s->available())

```

```

156     return false;
157 }
158
159 // Read a byte at a time until we get to the special '0x42' start-byte
160 if (s->peek() != 0x42) {
161     s->read();
162     return false;
163 }
164
165 // Now read all 32 bytes
166 if (s->available() < 32) {
167     return false;
168 }
169
170 uint8_t buffer[32];
171 uint16_t sum = 0;
172 s->readBytes(buffer, 32);
173
174 // get checksum ready
175 for (uint8_t i=0; i<30; i++) {
176     sum += buffer[i];
177 }
178
179 /* debugging
180 for (uint8_t i=2; i<32; i++) {
181     Serial.print("0x"); Serial.print(buffer[i], HEX); Serial.print(", ");
182 }
183 Serial.println();
184 */
185
186 // The data comes in endian'd, this solves it so it works on all platforms
187 uint16_t buffer_u16[15];
188 for (uint8_t i=0; i<15; i++) {
189     buffer_u16[i] = buffer[2 + i*2 + 1];
190     buffer_u16[i] += (buffer[2 + i*2] << 8);
191 }
192
193 // put it into a nice struct
194 memcpy((void *)&data, (void *)buffer_u16, 30);
195
196 if (sum != data.checksum) {
197     Serial.println("Checksum failure");
198     return false;
199 }
200 // success!
201 return true;
202 }

```