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

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

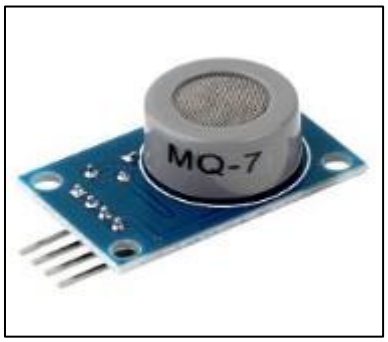


Figure 3.1.1: MQ-7 Sensor

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(SnO2) that has lower conductivity in clean air.

Working Principle

The sensor detection is through its sensitive layer of SnO2. In clean air, donor electrons in SnO2 will react with oxygen, which is adsorbed and form a layer of oxide on the surface of SnO2, 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

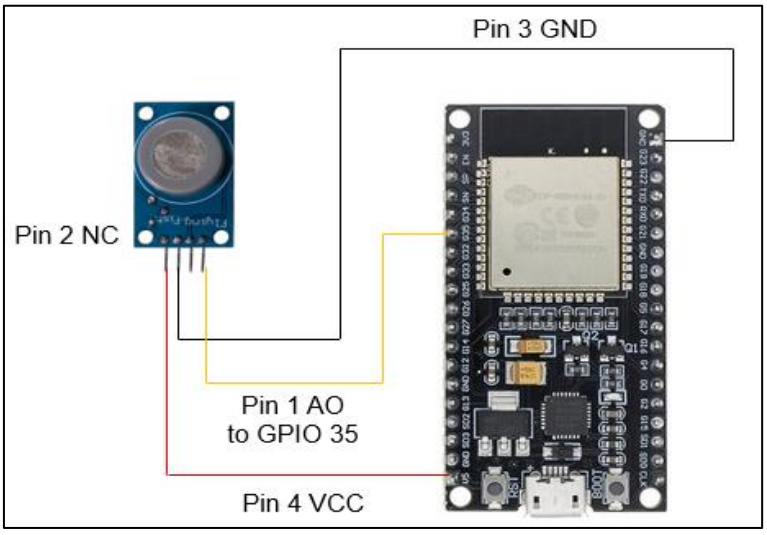


Figure 3.1.2: Connections between MQ-7 and ESP32

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.

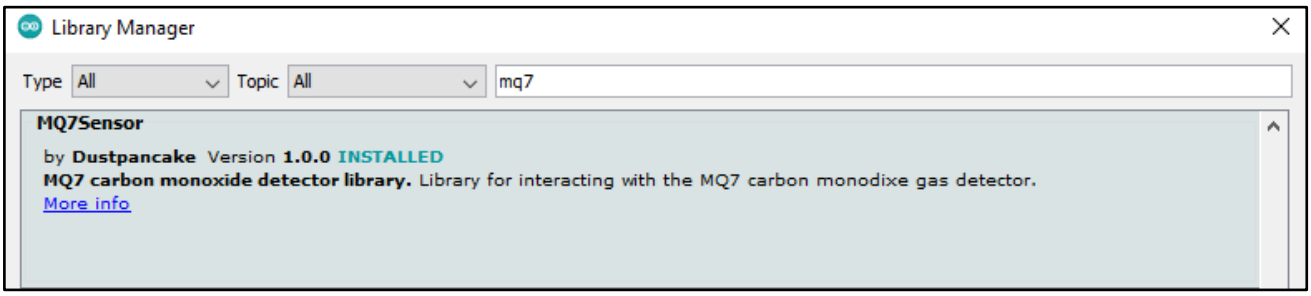


Figure 3.1.4: Library Installation

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.

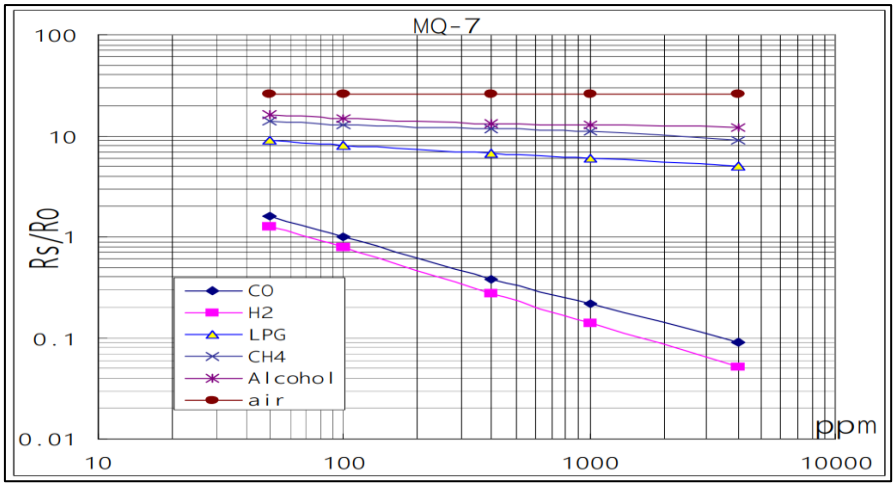


Figure 3.1.5: Datasheet Graph

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
Rs	Surface resistance Of sensitive body	2-20k	In 100ppm Carbon Monoxide
a (300/100ppm)	Concentration slope rate	Less than 0.5	Rs (300ppm)/Rs(100ppm)
Standard working condition	Temperature -20℃±2℃ relative humidity 65%±5% RL:10KΩ ±5%		
	Vc:5V±0.1V VH:5V±0.1V VH:1.4V±0.1V		
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

Figure 3.1.8: Example of Data Received

3.2 MQ-3 Alcohol Sensor

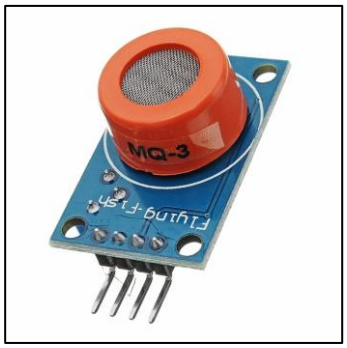


Figure 3.2.1: MQ-3 Sensor

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 SnO2 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

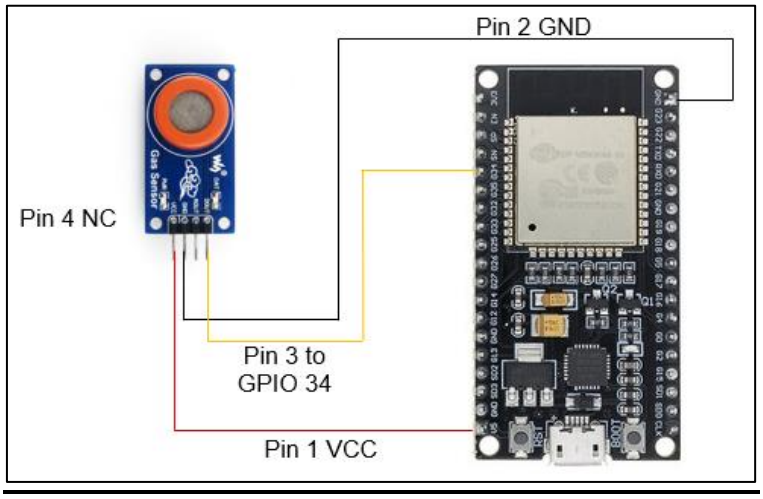


Figure 3.2.2: Connections between MQ-3 and ESP32

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.

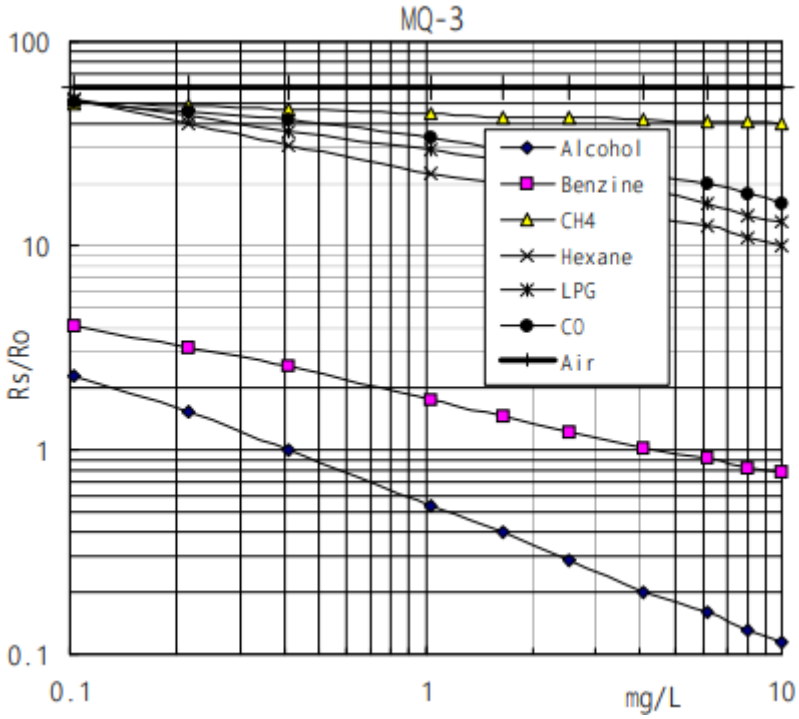


Figure 3.2.4: Datasheet Graph

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
Rs	Sensing Resistance	1MΩ - 8 MΩ (0.4mg/L alcohol)	Detecting concentration scope: 0.05mg/L—10mg/L Alcohol
α (0.4/1 mg/L)	Concentration slope rate	≤0.6	
Standard detecting condition	Temp: 20℃±2℃ Humidity: 65%±5%	Vc:5V±0.1 Vh: 5V±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.

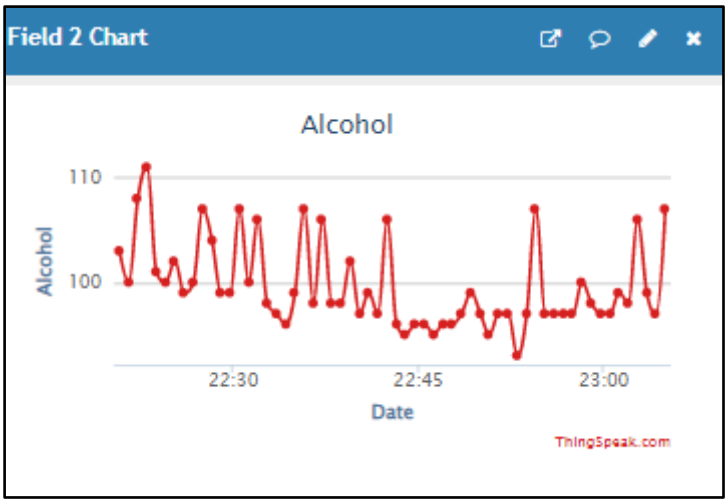


Figure 3.2.7: Example of Data Received at ThingSpeak

3.3 MQ-135 Carbon Diode Sensor

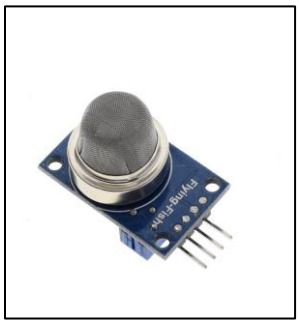


Figure 3.3.1: MQ-135 Sensor

Introduction

CO2 levels in the air are a strong indication of pollution levels. In areas that are not ventilated properly, CO2 levels can rise up to 1000ppm which is unhealthy.

Working Principle

The MQ-135 sensor is a metal oxide gas sensor that utilizes SnO2. 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 CO2.

Connection

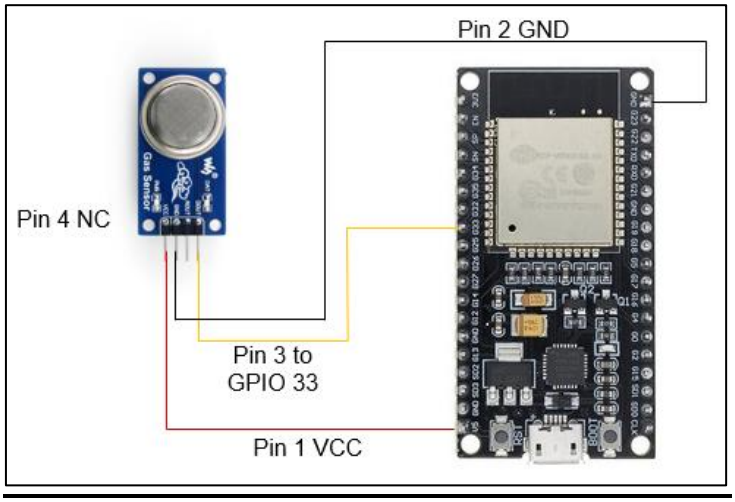


Figure 3.3.2: Connections between MQ-135 and ESP32

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 CO2 in the lab was approximated to be 450ppm when the sensor was calibrated. The Ro values were calculated by finding the current and the voltage passing through the sensor. The Rs values are the ones that kept changing.

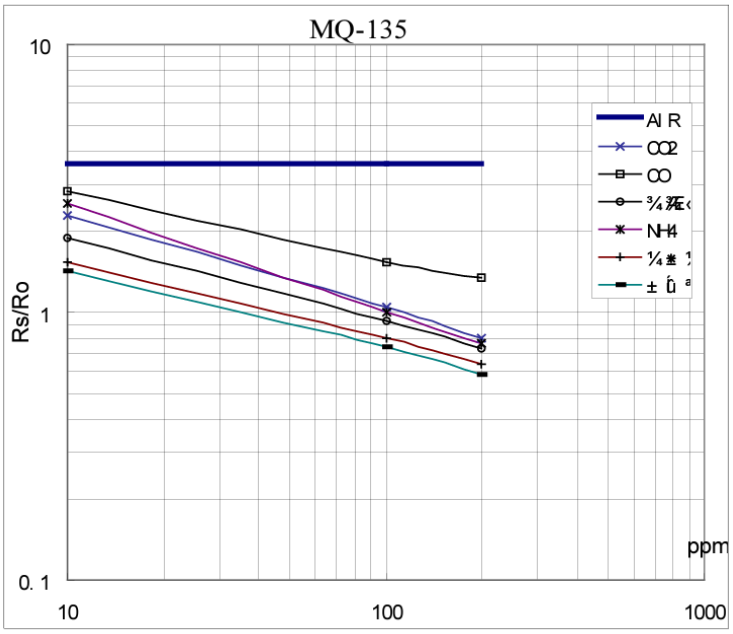


Figure 3.3.5: Datasheet Graph

Graph for ratio of Rs/Ro and ppm concentration

Once the concentration was found Rs was calculated with the formula:

$$Rs = (Vcc/VRL-1)x\ RL$$

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 Rs at 400ppm of CO2, we could use the spec sheet's graph to extrapolate the Rs/Ro values obtained from the sensor to find the respective concentration of CO2 in the air. The interesting thing to note here is that as CO2 concentration increases the output voltage of the MQ135 sensor decreases hence it behaves slightly differently from the other MQ sensors.

Code

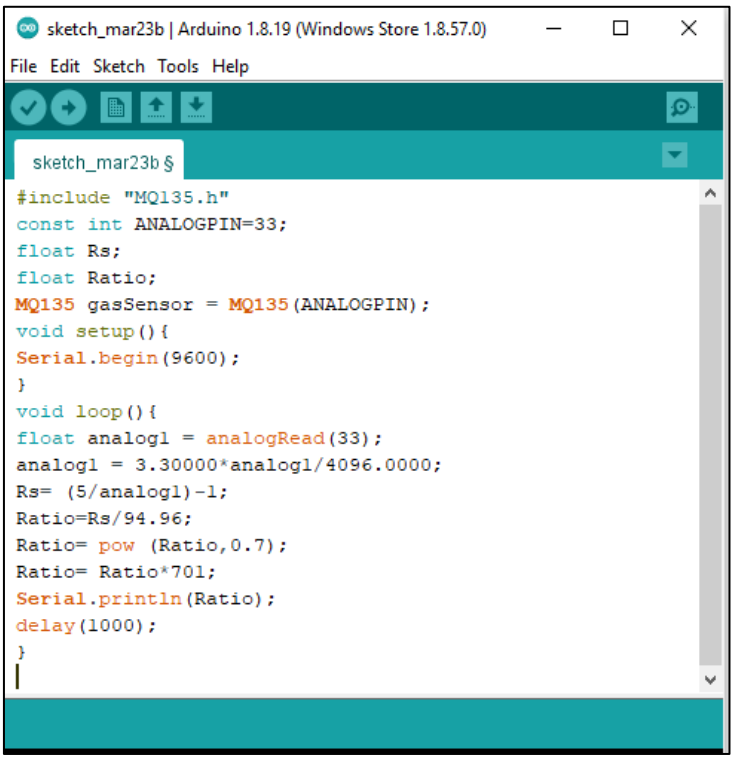


Figure 3.3.6 Code used to calibrate the input sensor value to detect concentration of CO2

3.4. PMS5003 PM2.5, PM10 Sensor

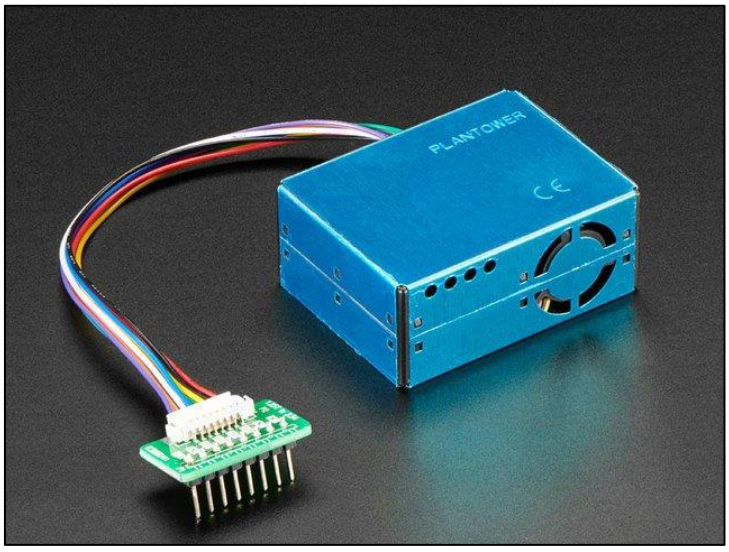


Figure 3.4.1: PMS5003 Sensor

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.

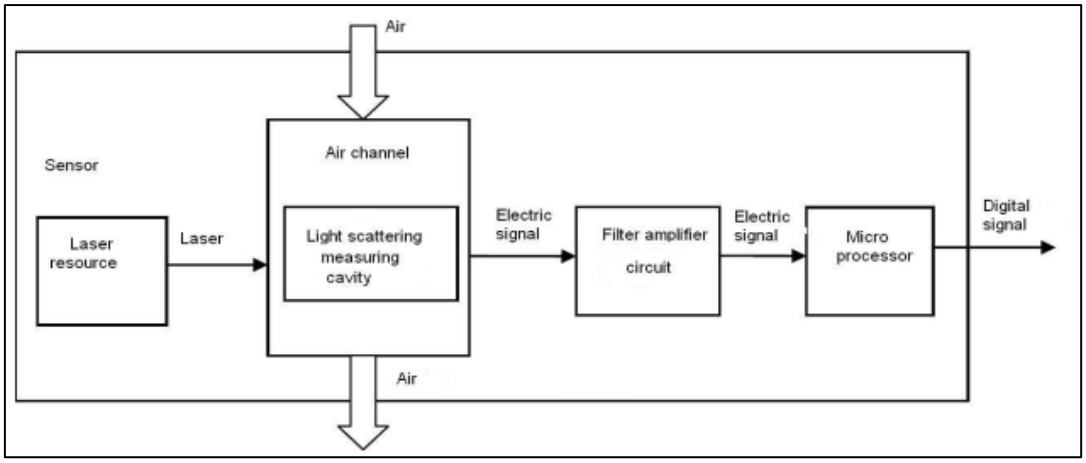
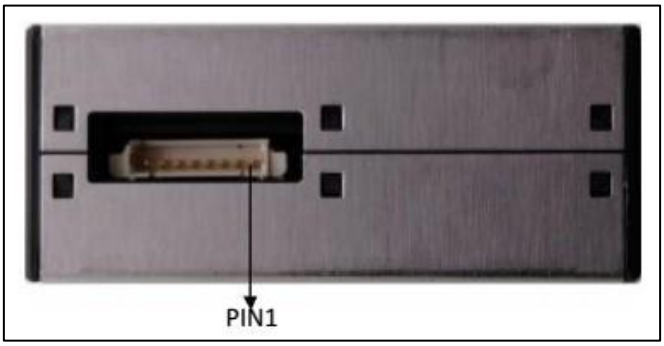


Figure 3.4.2: Workflow

Connecting PMS5003 to ESP32



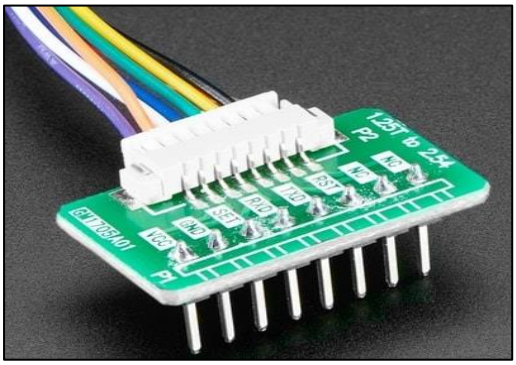


Figure 3.4.3 and Figure 3.4.4: PMS5003 Pins

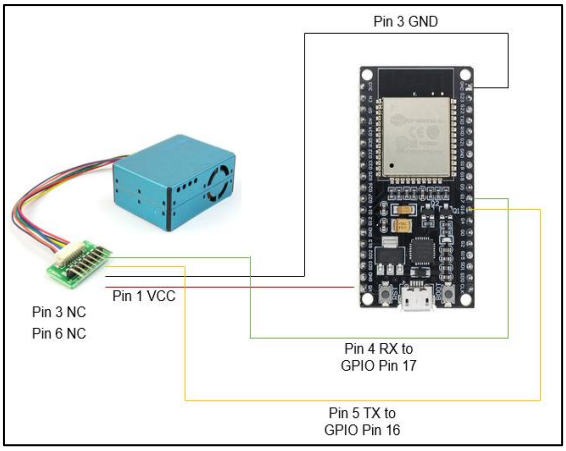


Figure 3.4.5: Connections between PMS5003 and ESP32

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 µg/m^3 and Data 3 refers to PM10 with a concentration unit of µg/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(&pmsSerial)) {
       (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 $PM_{2.5}$ ($\mu g/m^3$)	24-hr PM_{10} ($\mu g/m^3$)	24-hr SO_2 ($\mu g/m^3$)	8-hr CO (mg/m^3)	8-hr O_3 ($\mu g/m^3$)	1-hr NO_2 ($\mu g/m^3$)^
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

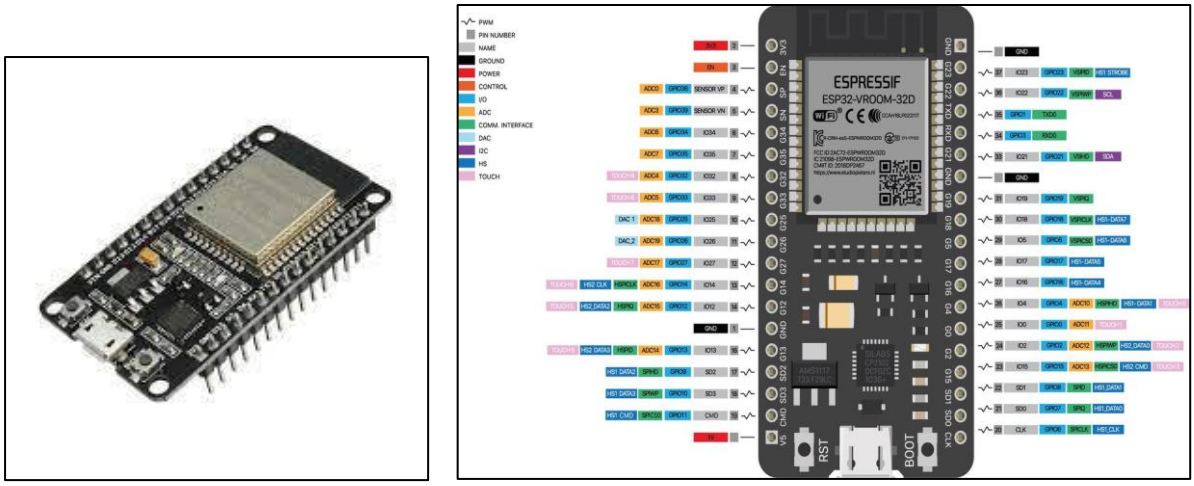


Figure 4.1: ESP32 Pin configuration

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:**
  - **I**nternal 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.

		Uno	**ESP32**
General	Dimension	2.7” x 2.1”	2” x 1.1”
	Pricing	$20-23	$10-12
Connectivity	I/O Pins	14	36
	PWM Pins	6	16
	Analog Pins	6	Up to 18*
	Analog Output Pins (DAC)		2
Computing	Processor	ATMega328P	Xtensa Dual Core 32-bit LX6 microprocessor
	Flash Memory	32kB	4MB
	SRAM	2kB	520kB
	EEPROM	1kB	-
	Clock SPeed	16MHz	Upto 240 MHz
	Voltage Level	5V	3.3V
	USB Connectivity	Standard A/B USB	Micro-USB
Communication	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)
Additional	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.

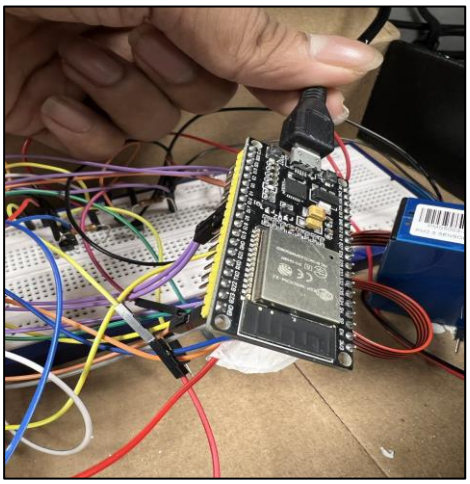


Figure 4.2: ESP32 Prototype Connection

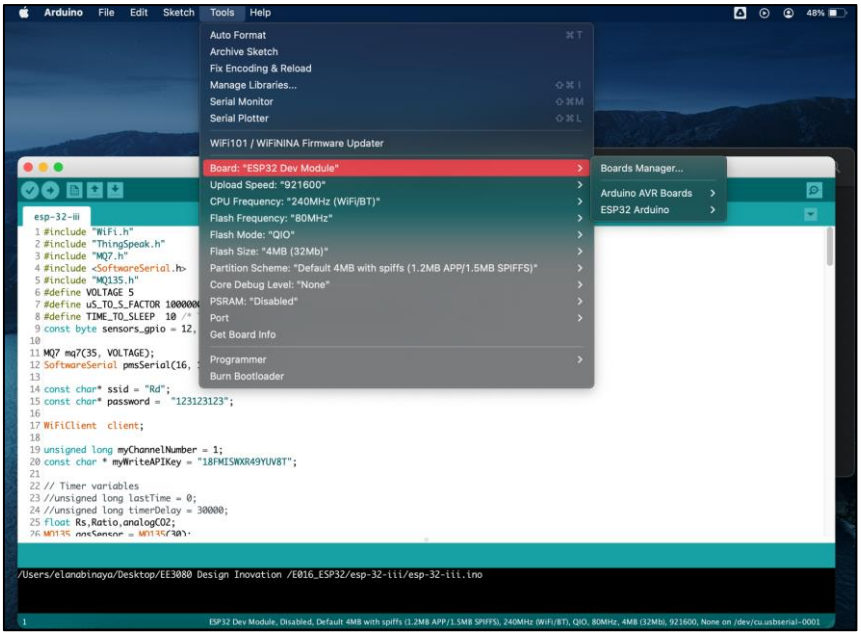


Figure 4.3: Arduino IDE: Board - - > ESP32 Dev Module

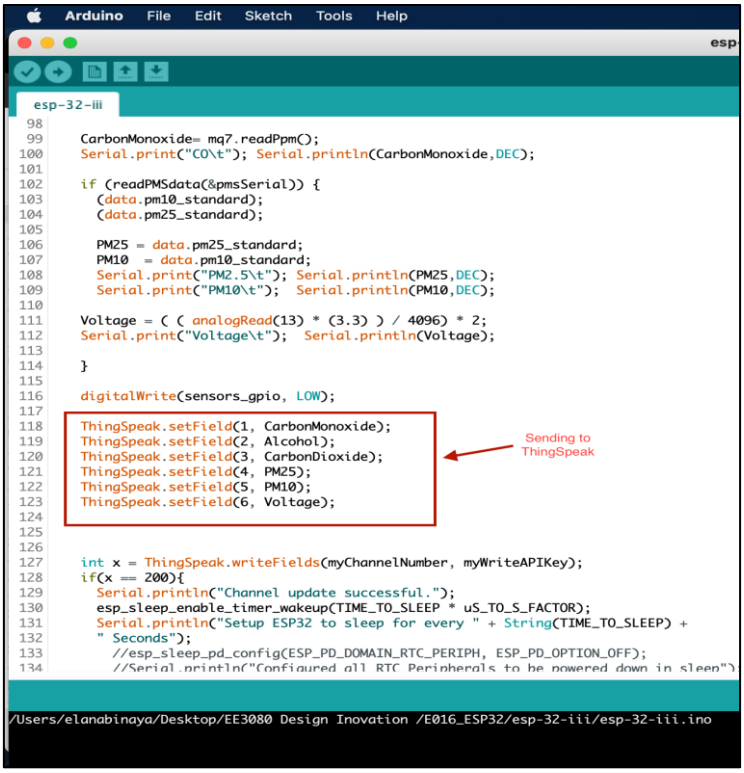


Figure 4.4: Arduino IDE: Set of sensor readings to be sent to ThingSpeak

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

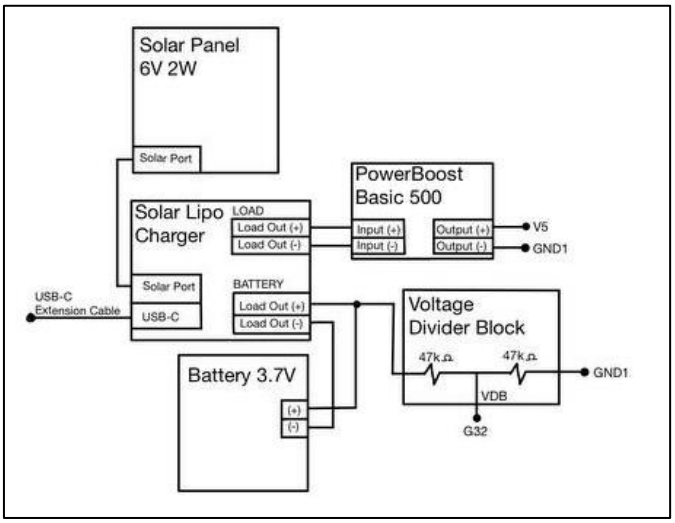


Figure 5.1: Power Management System

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 47kΩ 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

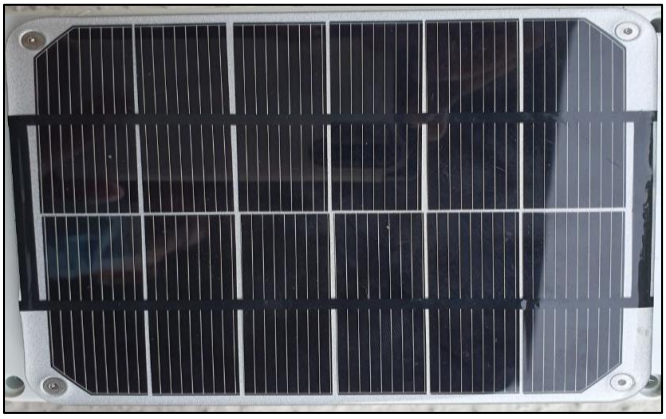
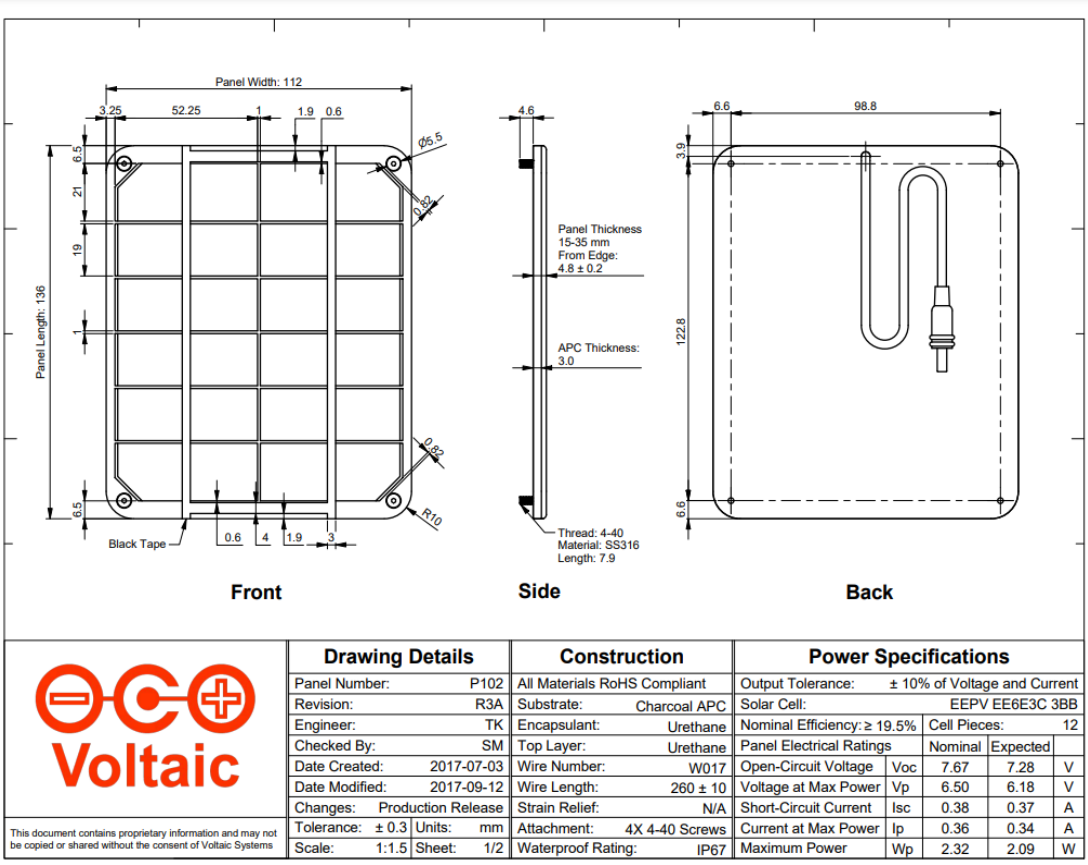


Figure 5.1.1: Solar Panel

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.



Voltaic

This document contains proprietary information and may not be copied or shared without the consent of Voltaic Systems

Drawing Details		Construction		Power Specifications				
Panel Number:	P102	All Materials RoHS Compliant		Output Tolerance:	± 10% of Voltage and Current			
Revision:	R3A	Substrate:	Charcoal APC	Solar Cell:	EEPV EE6E3C 3BB			
Engineer:	TK	Encapsulant:	Urethane	Nominal Efficiency: ≥ 19.5%		Cell Pieces:	12	
Checked By:	SM	Top Layer:	Urethane	Panel Electrical Ratings		Nominal	Expected	
Date Created:	2017-07-03	Wire Number:	W017	Open-Circuit Voltage	Voc	7.67	7.28	V
Date Modified:	2017-09-12	Wire Length:	260 ± 10	Voltage at Max Power	Vp	6.50	6.18	V
Changes:	Production Release	Strain Relief:	N/A	Short-Circuit Current	Isc	0.38	0.37	A
Tolerance: ± 0.3	Units: mm	Attachment:	4X 4-40 Screws	Current at Max Power	Ip	0.36	0.34	A
Scale: 1:1.5	Sheet: 1/2	Waterproof Rating:	IP67	Maximum Power	Wp	2.32	2.09	W

Figure 5.1.1.2: Solar Panel Specifications

5.1.2 Solar Charger (BQ24074) by Adafruit

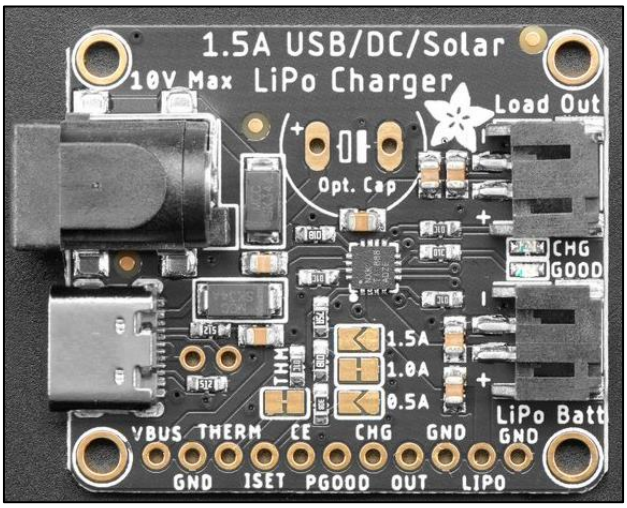


Figure 5.1.2: Solar Charger (BQ24074)

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.

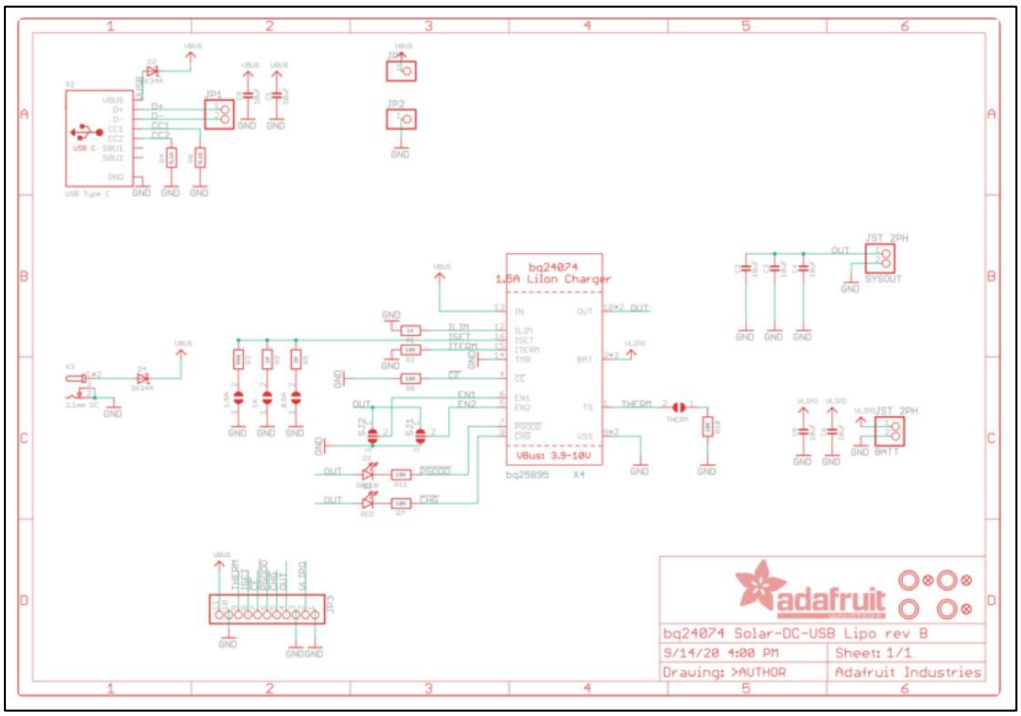


Figure 5.1.2: Solar Charger Schematic Print

5.1.3 PowerBoost 500 Basic DC / DC Converter by Adafruit

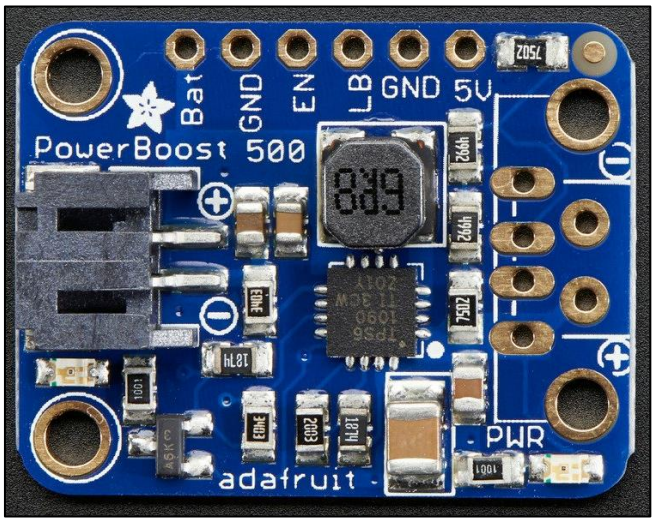


Figure 5.1.3: PowerBoost 500 Basic 5V USB @500mA

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.

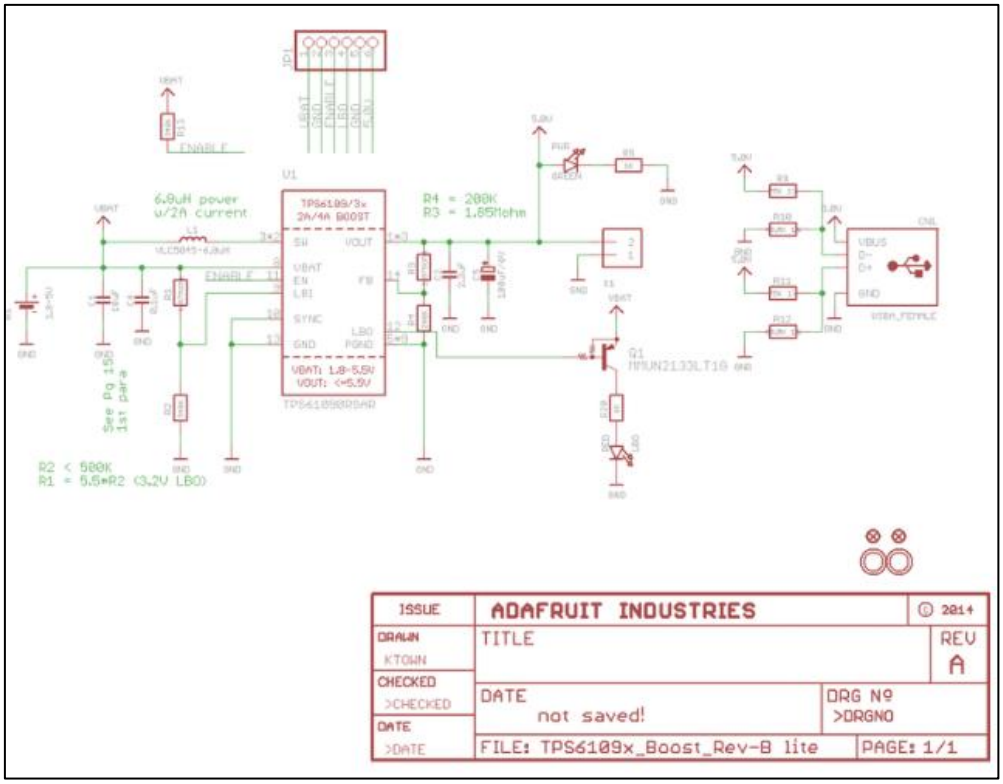


Figure 5.1.3.2: PowerBoost 500 Basic Schematic Print

5.1.4 Lithium-Ion Polymer Battery (LP785060)

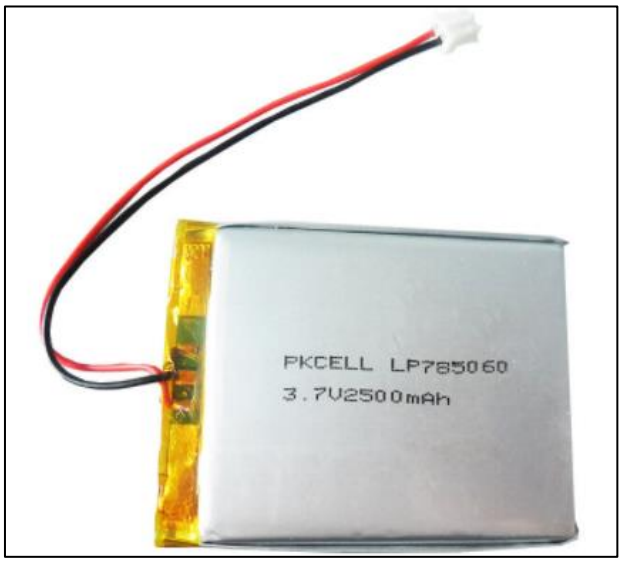


Figure 5.1.4.1: Lithium Ion Polymer Battery

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.

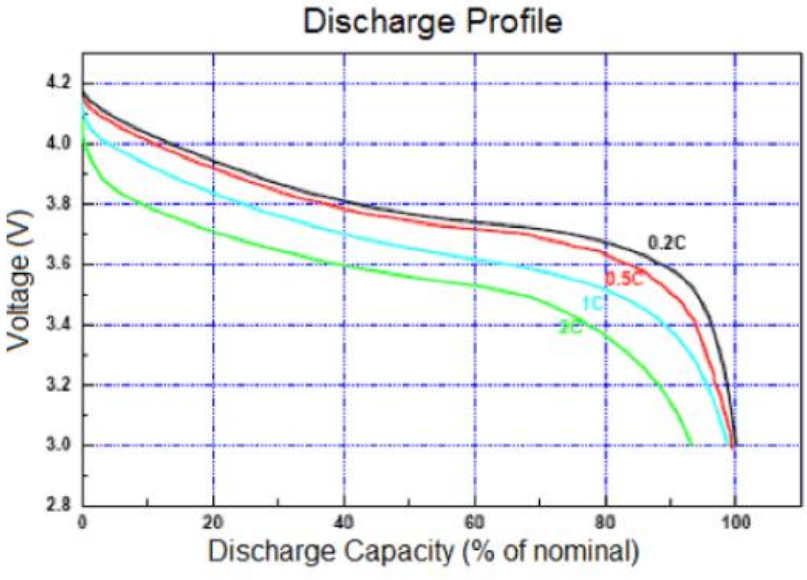


Figure 5.1.4.2: Lithium Ion Polymer Battery Discharge Profile

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_5A discharge，25℃
Nominal Voltage		3.75V	Average Voltage at 0.2C_5A discharge
Standard Charge Current		0.2 C_5A	Working temperature：0~40℃
Max Charge Current		1C_5A	Working temperature：0~40℃
Charge cut-off Voltage		4.2V	CC/CV
Standard Discharge Current		0.5C_5A	Working temperature：25℃
Discharge cut-off Voltage		2.75V	
Cell Voltage		3.7-3.9V	When leave factory
Impedance		≤35mΩ	AC 1KHz after 50% charge,25℃
Weight		Approx:46g	
Storage temperature	≤1month	-10~45℃	Best 20±5℃ for long-time storage
	≤3month	0~30℃	
	≤6month	20±5℃	
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 For TX RX more info in the next table
	Wi-Fi/BT Tx packet			
	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
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	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			
Hibernation	RTC timer only			5 µA
Power off	CHIP_PU is set to low level, the chip is powered off.			1 µA

Only RTC timer set to function

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 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.

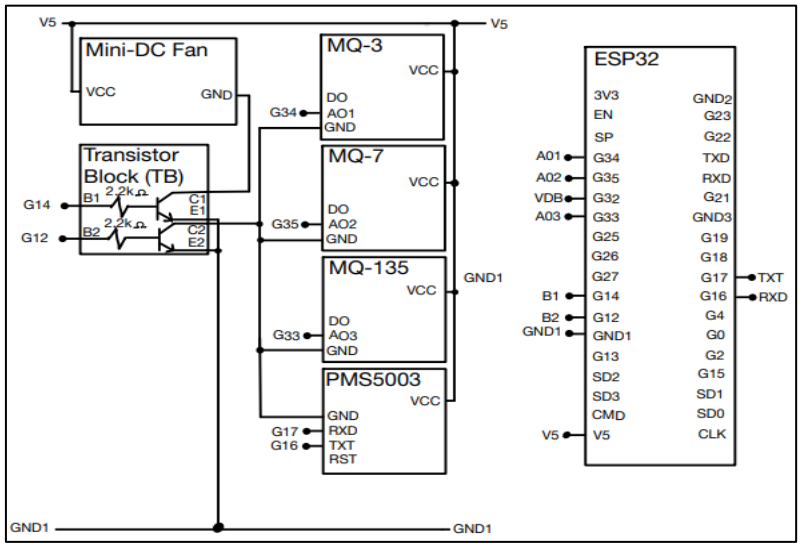
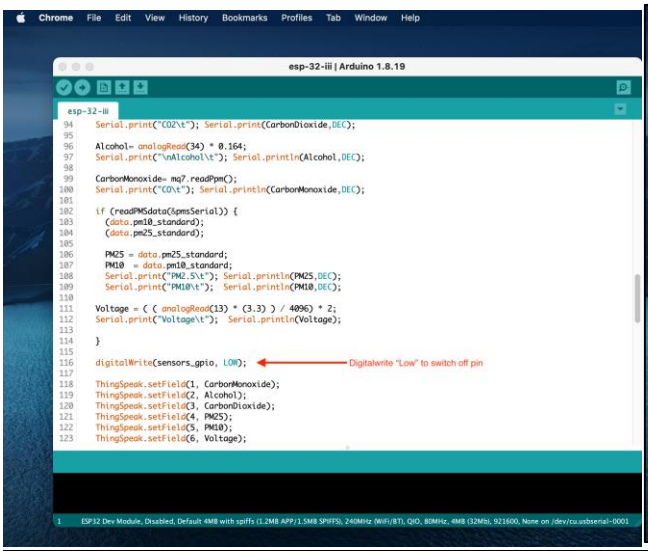


Figure 5.2.2:Schematic of the transistor connection to sensors and Esp32



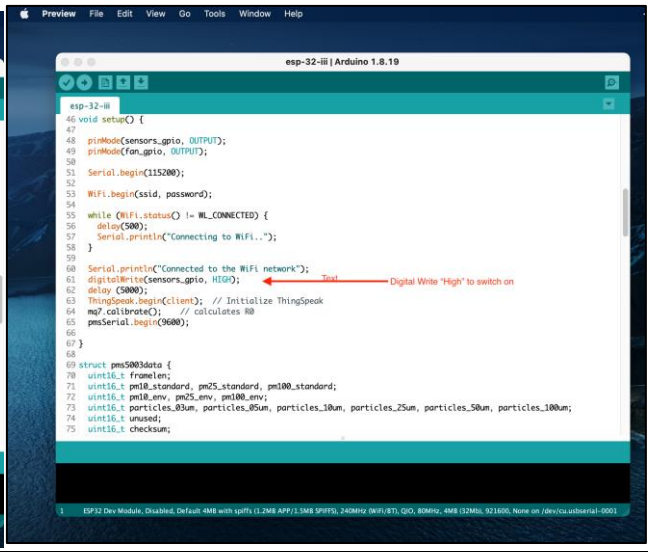


Figure 5.2.3 Coding to switch on pin12

Figure 5.2.3 Coding to switch off pin12

5.3 Battery Voltage Discharge Test

ESP32 ADC pins can only read voltages between 0V to 3.3V while the battery supplies 3.7V at full charge. Two 47kΩ resistors are connected to the battery and the ground, GPIO32 with analog read capability from the ESP32 is connected in between the two resistors in a voltage divider configuration. This allows the ESP32 to read the voltage from the Battery by multiplying the voltage read by the pin by 2. With latency and the 10 seconds delay between sending and going to sleep mode, each data point will be separated by a 40-second gap. The test starts with an almost full charge (70%) and ends at (30%).

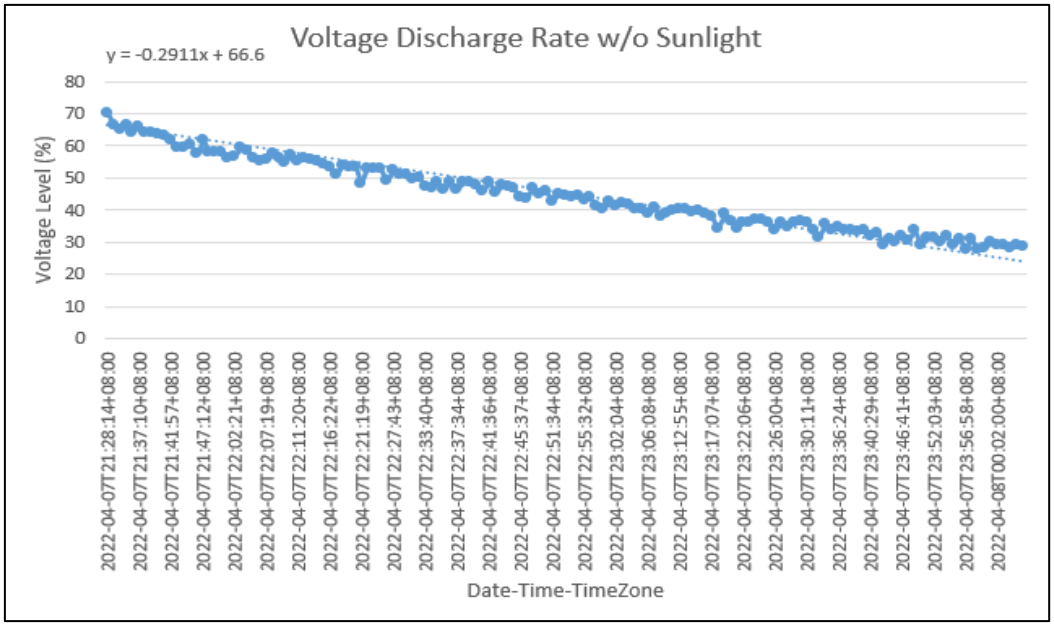


Figure 5.3.1: Voltage Discharge Rate with no Sunlight

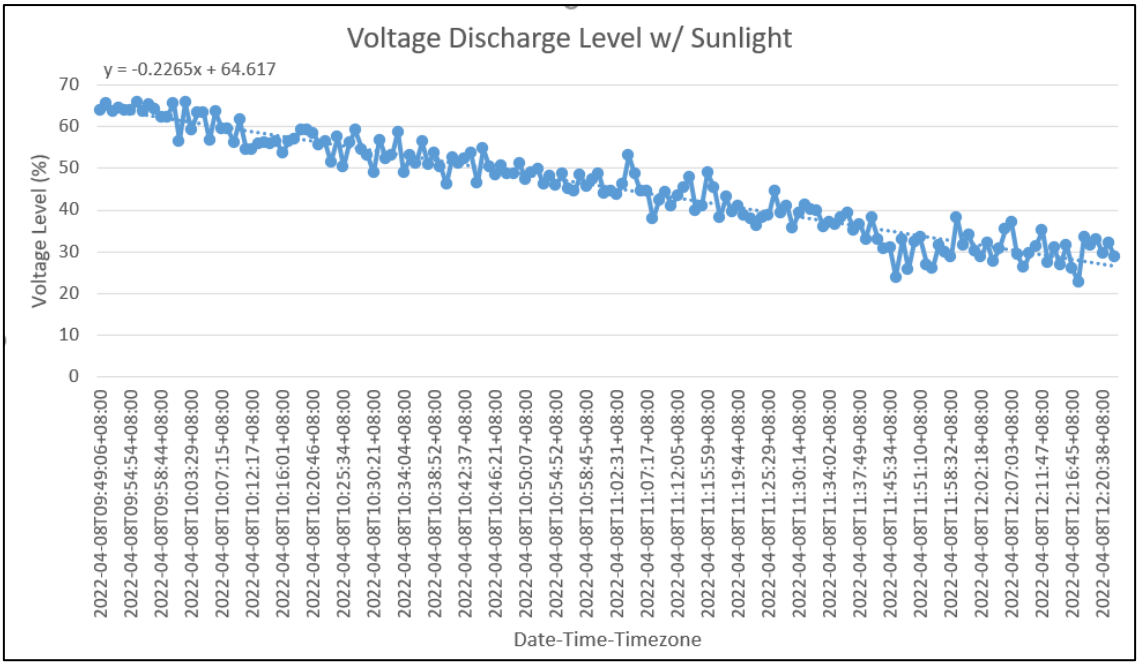


Figure 5.3.2: Voltage Discharge Rate with Sunlight

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.

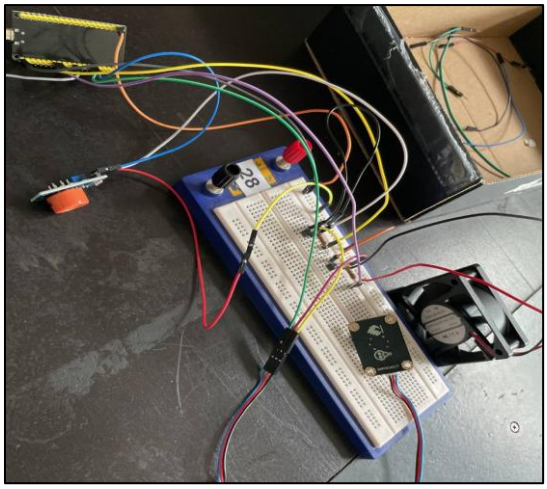


Figure 6.2.1: An early testing prototype with two sensors and a fan connected to the ESP32

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.

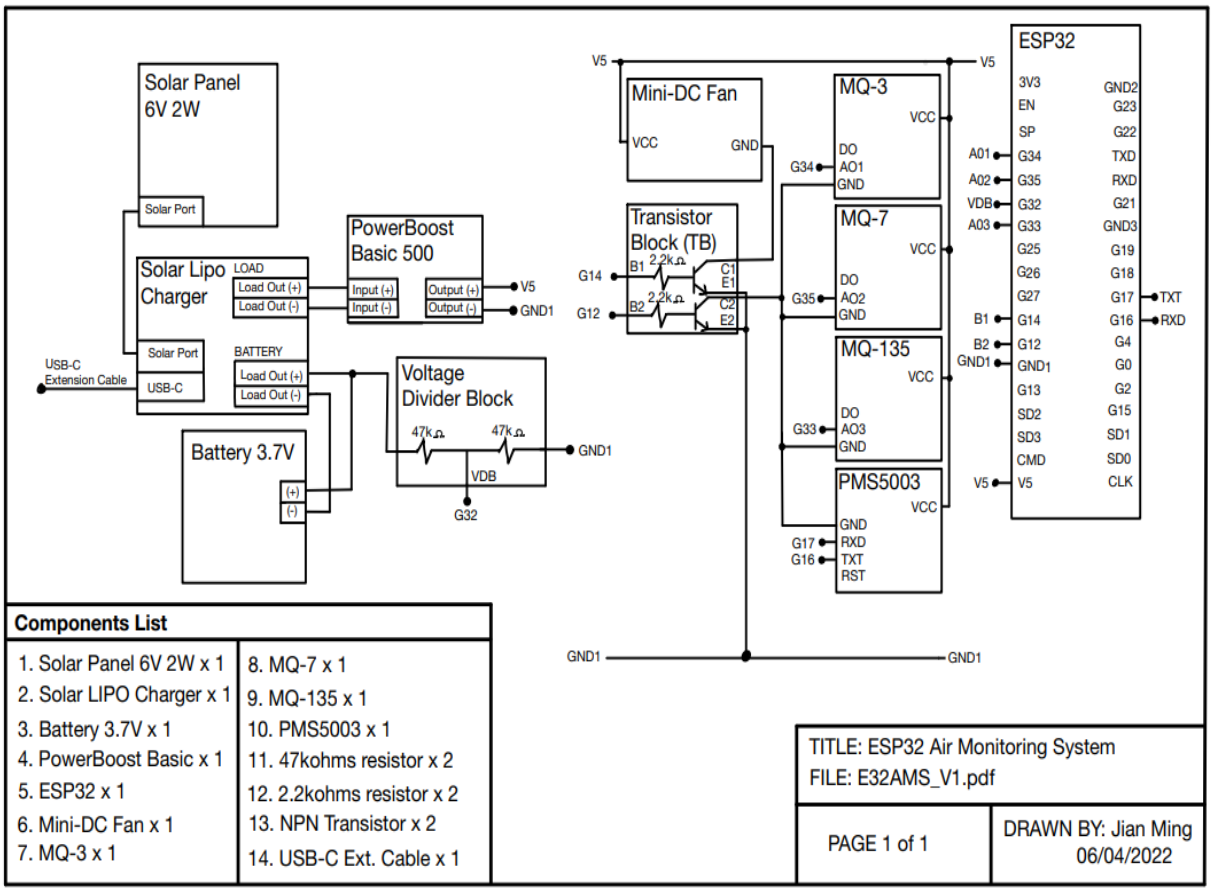


Figure 6.4.1: Schematic of the finalised design

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.

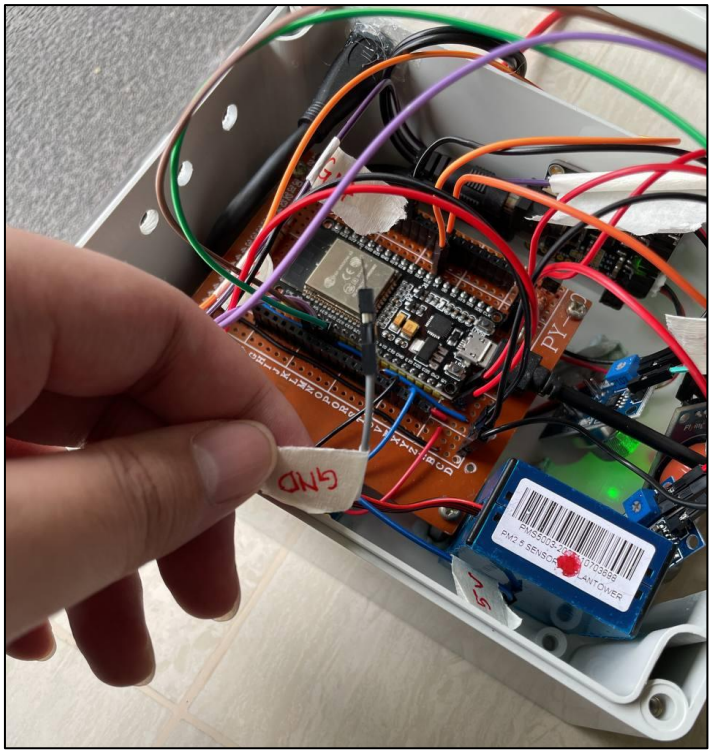


Figure:6.6.1 :Final prototype after all subsystems are integrated

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.

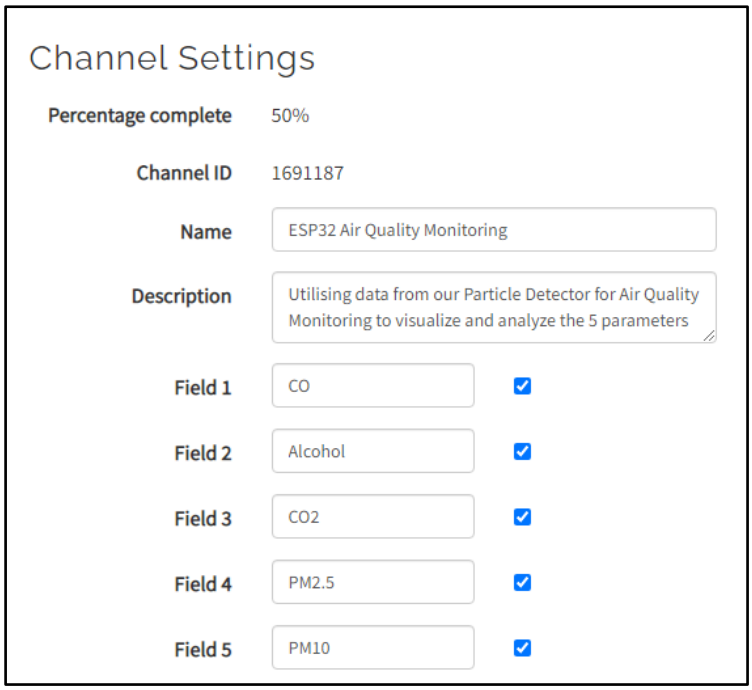


Figure 7.0.2: Channel settings of thingspeak

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.

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.

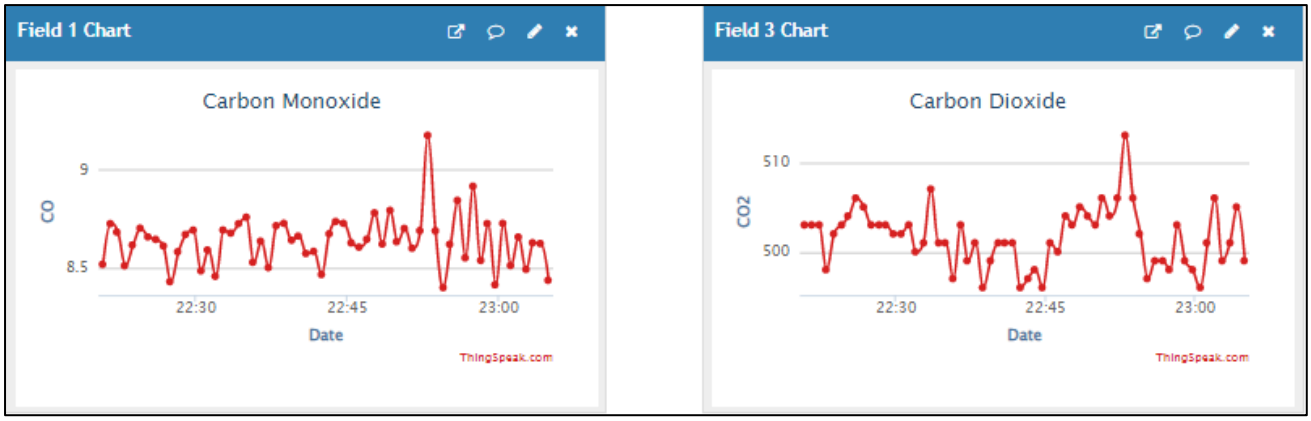


Figure 7.0.4: Data visualisation

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.

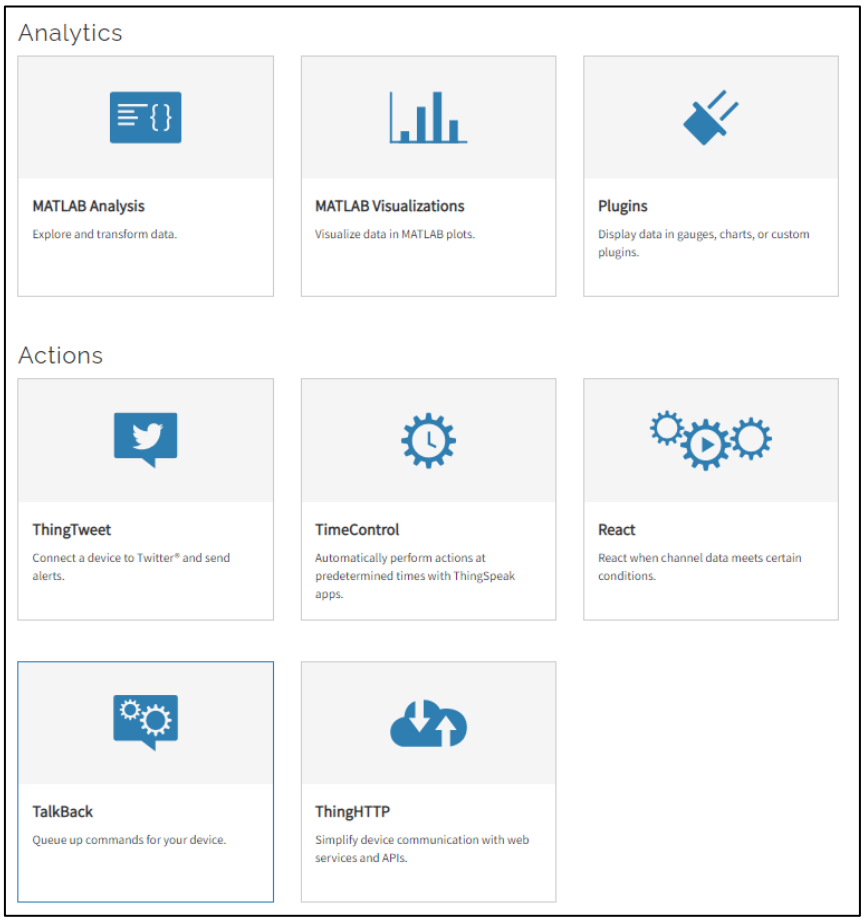


Figure 7.1: Applications of Thingspeak

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.

ThingSpeak™ Channels Apps Devices Support

Name

Google Gauge 3

HTML

```
<html>
  <head>

  <!--

  NOTE: This plugin will not be visible on public views of a channel.
        If you intend to make your channel public, consider using the
        MATLAB Visualization App to create your visualizations.

  -->

  <title>Google Gauge - ThingSpeak</title>

  %%PLUGIN_CSS%%
  %%PLUGIN_JAVASCRIPT%%

  </head>
```

CSS

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

JavaScript

```
<script type='text/javascript' src='https://ajax.googleapis.com/ajax/libs/jquer
<script type='text/javascript' src='https://www.google.com/jsapi'></script>
<script type='text/javascript'>

  // set your channel id here
  //var channel_id = 1665650;
  var channel_id = 1691187;
```

Figure 7.1.1: Google Gauge code

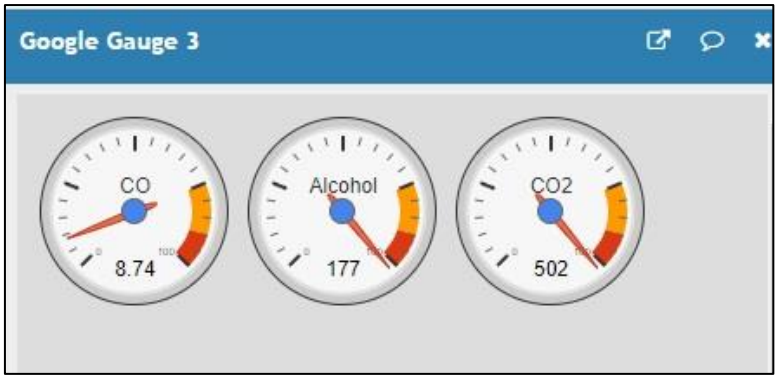


Figure 7.1.2: Google Gauges

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.

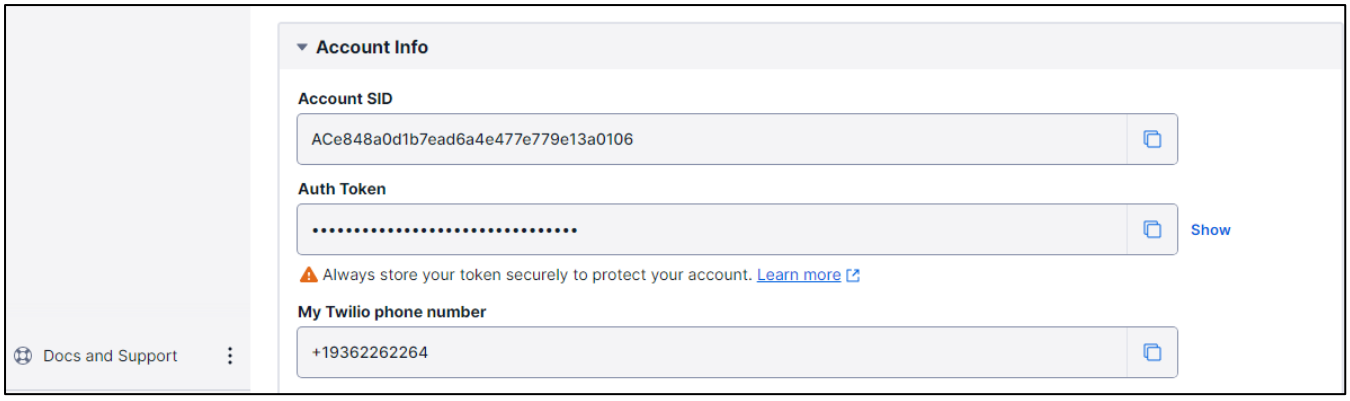


Figure 7.2.1: Twilio account info

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.

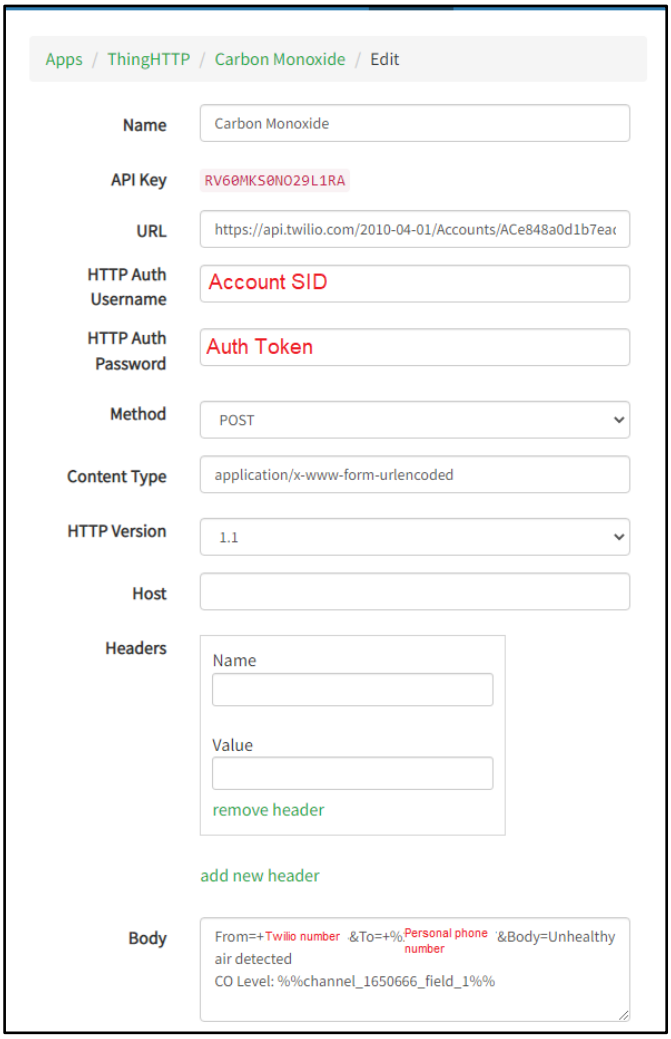


Figure 7.2.2: ThingHTTP settings

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 Alcoho	1000 ppm
Carbon Dioxide	5000 ppm
PM2.5	151 $\mu g/m^3$
PM10	351 $\mu g/m^3$

Figure 7.2.3: Threshold table

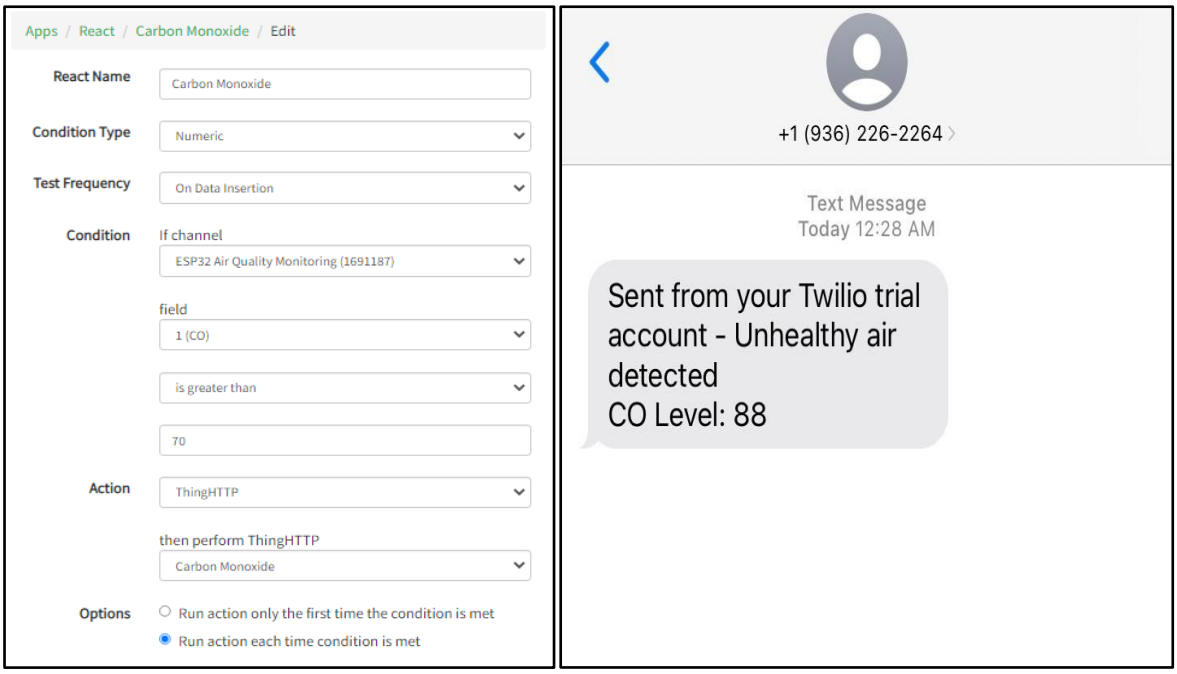


Figure 7.2.4: React settings and SMS received

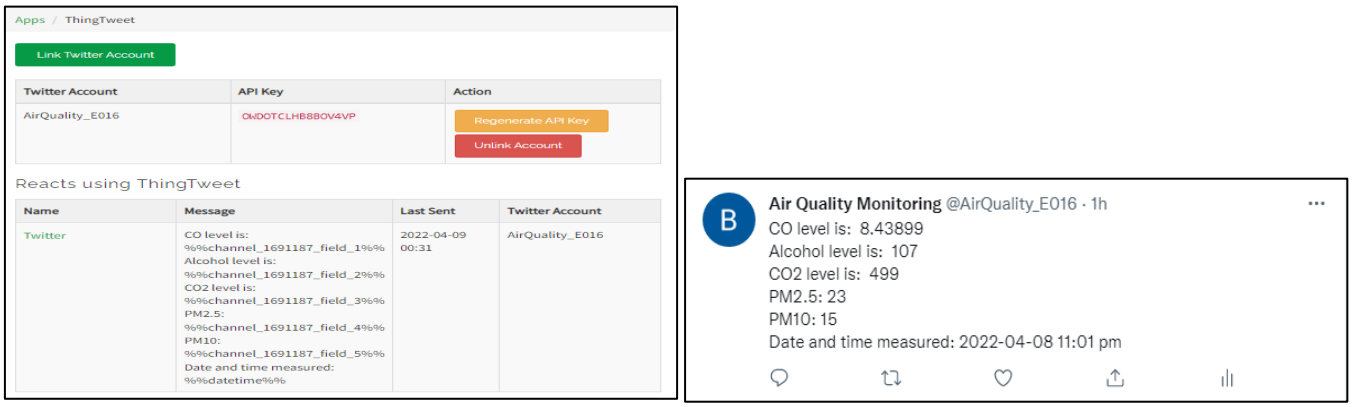


Figure 7.2.5: Thingtweet settings and the Tweet

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.

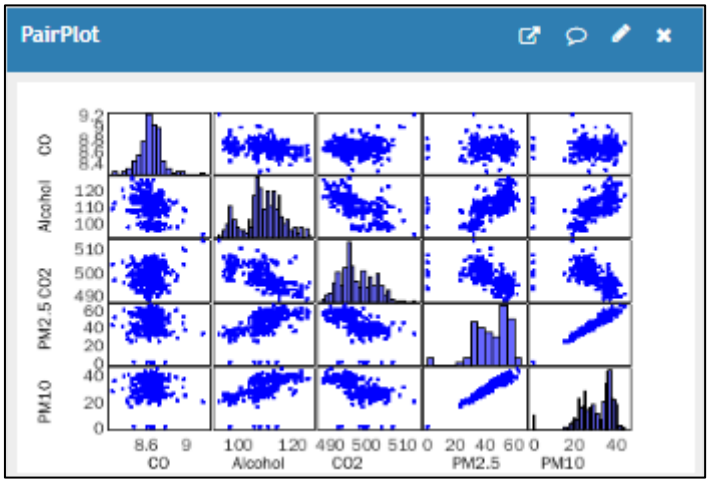


Figure 7.3.1: Pairplot

Name

PairPlot

MATLAB Code

```
% Channel ID to read data from
readChannelID = 1691187;
COFieldID = 1;
AlcoholFieldID = 2;
CO2FieldID = 3;
PM25FieldID = 4;
PM10FieldID = 5;

% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = '';
data = thingSpeakRead(readChannelID,'Fields',[COFieldID AlcoholFieldID CO2FieldID PM25FieldID
                                              'NumPoints',300, ...
                                              'ReadKey',readAPIKey);
% Read all Data
COData = data(:,1);
AlcoholData = data (:,2);
CO2Data = data (:,3);
PM25Data = data(:,4);
PM10Data = data(:,5);
x =[COData,AlcoholData,CO2Data,PM25Data,PM10Data];
figure
gplotmatrix(x);
varNames = {'CO'; 'Alcohol'; 'CO2'; 'PM2.5'; 'PM10'};
text([.08 .24 .43 .66 .83], repmat(-.1,1,5), varNames, 'FontSize',8);
text(repmat(-.12,1,5), [.86 .62 .41 .25 .02], varNames, 'FontSize',8, 'Rotation',90);
```

Figure 7.3.2: Pairplot code

Cleaning of Data

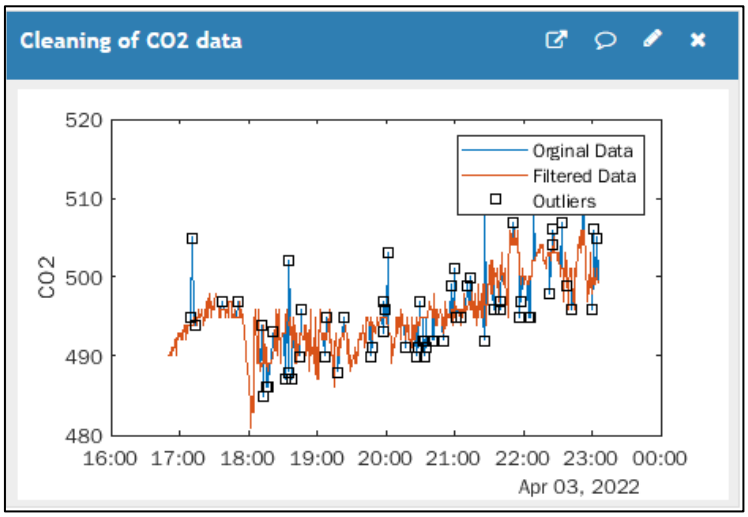


Figure 7.3.3: Visualisation of CO2 data cleaning

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.

Apps / MATLAB Visualizations / Cleaning of CO2 data / Edit

Name

Cleaning of CO2 data

MATLAB Code

```
% Enter your MATLAB code below

data = thingSpeakRead(1691187,'NumPoints',800,'Fields',3,'outputFormat','table');
windowSize = 6;
numMedians = 2;
[filteredData,outliers]=hampel(data.CO2,windowSize,numMedians);
plot(data.Timestamps,data.CO2);
hold
plot(data.Timestamps,filteredData);
plot(data.Timestamps(outliers),data.CO2(outliers),'Marker','Square','Linestyle','None','Color'
ylabel('Alcohol');
legend('Orginal Data', 'Filtered Data','Outliers');
```

Figure 7.3.4: CO2 data cleaning code

Forecasting

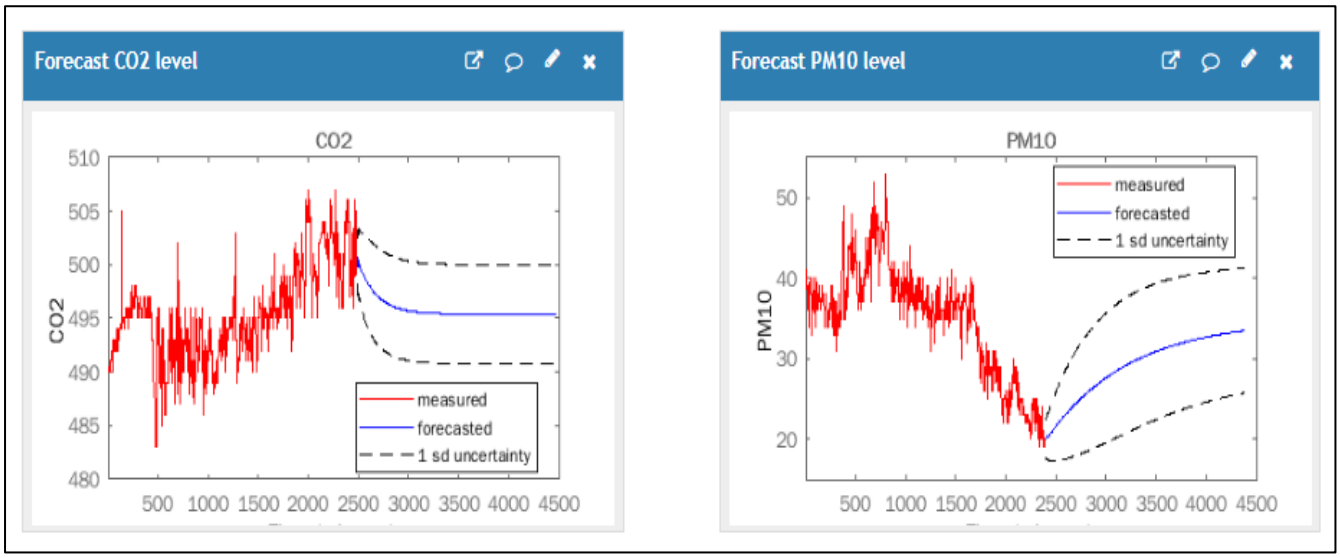


Figure 7.3.5: Visualisation of forecasting

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

```
startDate = datetime('March 31, 2022 10:00:00 AM');
endDate = datetime('April 01, 2022 10:00:00 AM');
dateRange = startDate:endDate;
data = thingSpeakRead(1691187,'DateRange',dateRange,'Fields',3);
sampleTime = 5;
IDdata = iddata(data,[],sampleTime,'OutputName',{'CO2'},'TimeUnit','minutes')
trend = getTrend(IDdata,0);
IDdata = detrend(IDdata,0);
modelOrder = 8;
sys = ar(IDdata,modelOrder);
numSamples = 400;
[yf,x0,sysf,yf_sd,x,x_sd] = forecast(sys,IDdata,numSamples);
IDdata = retrend(IDdata,trend);
yf = retrend(yf,trend);
figure;
UpperBound = iddata(yf.OutputData+1*yf_sd,[],yf.Ts,'Tstart',yf.Tstart,'TimeUnit','minutes');
LowerBound = iddata(yf.OutputData-1*yf_sd,[],yf.Ts,'Tstart',yf.Tstart,'TimeUnit','minutes');
plot(IDdata(:,:,[]),'r',yf(:,:,[]),'b');
hold on
plot(UpperBound,'k--',LowerBound,'k--');
legend({'measured','forecasted','1 sd uncertainty'},'Location','best');
xlabel('Time');
ylabel('CO2');


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.

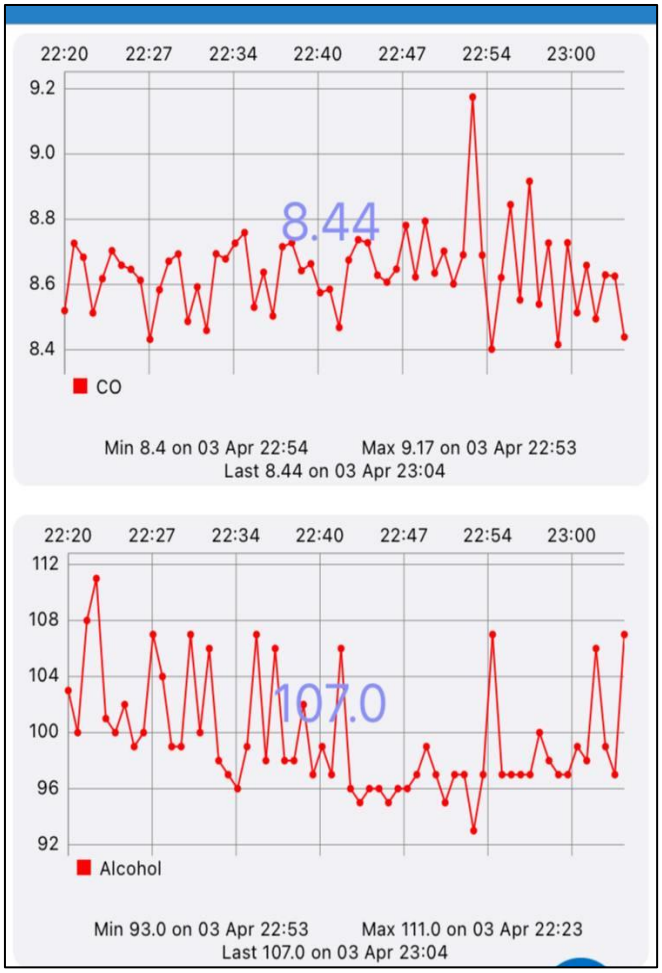


Figure 7.4.1: Thingview platform

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				✓									
Execution													
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 IOT and Power Management Making a prototype enclosure												✓	
Testing												✓	
Demonstration of prototype													

	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:
Power			
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
Sensors			
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
Assembly			
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

Project Deliverables	Learning Outcomes
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)

[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

[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

[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

[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

[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

[14] Solar Charger (BR-2489958) Schematic
adafruit_bq24074_universal_usb_dc_solar_charger_br-2489958.pdf (mouser.sg)

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

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.print(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 (readPMSdata(&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\t"); Serial.println(voltage_value);
120
121
122     digitalWrite(sensors_gpio, LOW);
123
124     ThingSpeak.setField(1, CarbonMonoxide);
125     ThingSpeak.setField(2, Alcohol);
126     ThingSpeak.setField(3, CarbonDioxide);
127     ThingSpeak.setField(4, PM25);
128     ThingSpeak.setField(5, PM10);
129     ThingSpeak.setField(6, voltage_value);
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()) {
```

```
    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);

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

  /* debugging
  for (uint8_t i=2; i<32; i++) {
    Serial.print("0x"); Serial.print(buffer[i], HEX); Serial.print(", ");
  }
  Serial.println();
  */

  // The data comes in endian'd, this solves it so it works on all platforms
  uint16_t buffer_u16[15];
  for (uint8_t i=0; i<15; i++) {
    buffer_u16[i] = buffer[2 + i*2 + 1];
    buffer_u16[i] += (buffer[2 + i*2] << 8);
  }

  // put it into a nice struct :)
  memcpy((void *)&data, (void *)buffer_u16, 30);

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