Project No: A2038-231

Final Year Project (FYP)

Smart Dust Monitoring System with IoT Cloud

Submitted by: Muhammad Ameerul Bin Azman

Supervisor: Assoc Prof Chan Pak Kwong

School of Electrical & Electronic Engineering

A final year project report presented to Nanyang Technological University
in partial fulfilment of the requirements of the degree of
Bachelor of Engineering

2024

Table of Contents

Abstract

This project is about the development of a smart dust monitoring system that is also equipped with IoT Cloud. As dust is crucial to the health of the people, it is important to monitor the dust level and to get alerts if the dust level exceeds the healthy range. With the inclusion of IoT Cloud, the ability to remotely monitor the dust level, sending out alerts and having historical data further enhance the capabilities of the system.

For this project, a system was developed using an ESP32, dust sensor and ThingSpeak as the IoT Cloud platform. Every reading taken would be transmitted to the ThingSpeak channel where you can monitor the dust level and view historical data. If dust level exceeds the threshold, an alert will be sent out. For power efficiency, a dynamic sampling rate is implemented to control the system's timer for its deep sleep mode where most of its functions will be shut down while saving some states on its RAM to reduce its energy consumption.

Through this report, I will be sharing my hands-on experience with designing and developing a micro-controller-based system with IoT Cloud.

Acronyms

IoT	Internet of Things
LED	Light emitting diode
IDE	Integrated Development Environments
API	Application Programming Interface
URL	Uniform Resource Locator
CSV	Comma-Separated Values

List of Figures

List of Tables

Chapter 1 Introduction

This chapter is on the background information, objectives, and scope of the project.

1.1 Background Information

Air quality and environmental concerns are increasing in today’s world. The presence of dust particles is a worry for areas that are especially close to industrial factories. According to Voiland [1], the airborne particles which are thirty times smaller than the width of a human hair can easily pass into the lungs and bloodstream, where they can increase a person’s risk of dying from heart disease, stroke, lung cancer, chronic obstructive pulmonary disease, and lower respiratory infections [1]. Older adults with chronic heart or lung disease, children and asthmatics are the groups most likely to experience adverse health effects with the exposure as stated from the California’s Air Resources Board [2]. Wu et al. [3] also mentioned that it has been regarded that microelectromechanical systems (MEMS) are a key technology that will contribute greatly for the success of IoT as it could be used to implement a system on a chip with higher performance and lower cost. The use of microcontroller has increased throughout the years.

Thus, it is essential to study and develop a “Smart Dust Monitor System with IoT Cloud” to monitor the dust level and to get the alert as well as for remotely monitoring through the IoT cloud.

1.2 Objectives

This project aims to design and develop a microcontroller-based dust monitoring system that can measure the dust level in real-time. Integrating the system with an Internet of Things (IoT) cloud system allows us to remotely monitor the dust level, get instant alerts when dust level is high and have cloud-based data storage and analysis. Güven et al. [4] stated that even in Universities, courses that deals with automation, robotics, communication, and control are mentioned with these microcontrollers. The high performance, low cost and the scalability of the microcontroller-based system is what made it a very popular choice to work with.

1.3 Scope

The system will periodically scan the air for its dust particles using a sensor that uses an infrared emitting diode and a phototransistor to allow it to detect the reflected light of dust in the air. Data that are collected will be sent to the IoT cloud server which will then be processed for storage, visualization and for triggering alerts if dust level exceeds the safe level. Users will be able to see actual dust level in real-time and get alerts remotely with this system.

For the hardware, an ESP32 will be used as the microcontroller that has an in-built Wi-Fi capability and a SHARP GP2Y1014AU0F Dust Sensor for the sensor. As for the software, Arduino IDE is used for the coding and ThinkSpeak for the IoT Cloud server.

Chapter 2 Literature Review

2.1 How dust affects our health

Ambient air pollution in both cities and rural area has been estimated to have caused 4.2 million premature deaths worldwide per year back in 2019 [5]. At home, even after a thorough cleaning and dusting have been carried out, dust will still be present in the air. While household dust is mostly made up of human skin, microscopic creatures and dead bugs, it does not cause significant health risks for most [6]. The dust that we have to wary of are of the fine particulate matter (PM2.5) which have been mentioned in Chapter 1. This fine particulate matter can cause serious health problems especially to people who have respiratory issues. According to the National Environmental Agency (NEA), for a 1-hour PM2.5 levels should be in the 0-55 µg/m^3 for it to be in the normal range [7]. Anything above this range would be considered as high and long exposure to high level of PM2.5 may have adverse effects on your health.

2.2 Micro-controller

Micro-controllers are integrated circuits (IC) that combines the function of microprocessor, memory and input/output (I/O) peripherals onto a single chip [8]. In the current market, there are various micro-controller available that caters to different needs. There are multiple ways that a micro-controller is classified. According to Agarwal [9], they can be characterized according to the number of bits, embedded or external memory, instruction set (CISC and RISC), memory architecture and types. Its versatility is what makes it a popular choice for hobbyists and even amongst the professionals.

Chapter 3 Components and Clouds

This chapter will be on the components, software and IoT Cloud service used for this project.

3.1 Hardware

3.1.1 Microcontroller

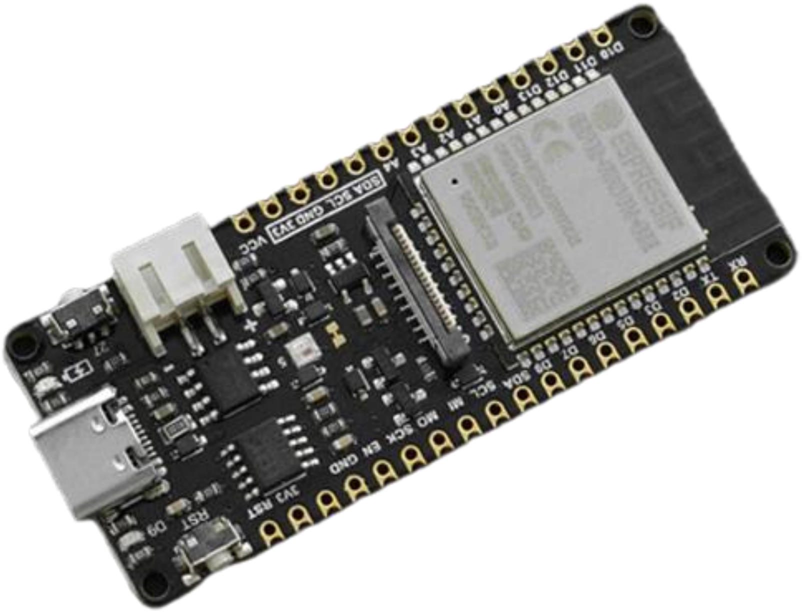


Figure 1: FireBeetle-ESP32

The microcontroller chosen is the FireBeetle-ESP32 by DFRobot FireBeetle series. This specific ESP32 was chosen as it is a low-power consumption micro-controller that is intentionally designed for Internet of Things (IoT) projects. It features a dual-core ESP-WROOM-32 module designed by Espressif System. It supports MCU, Wi-Fi and Bluetooth dual-mode communication and can be powered by a USB cable or a 3.7V external lithium battery.

In addition, the FireBeetle-ESP32 also features peripheral equipment like ADC, I2C, I2S, SPI, UART etc. Programming the micro-controller can also be done on the Arduino IDE which only requires the ESP32 libraries to be installed.

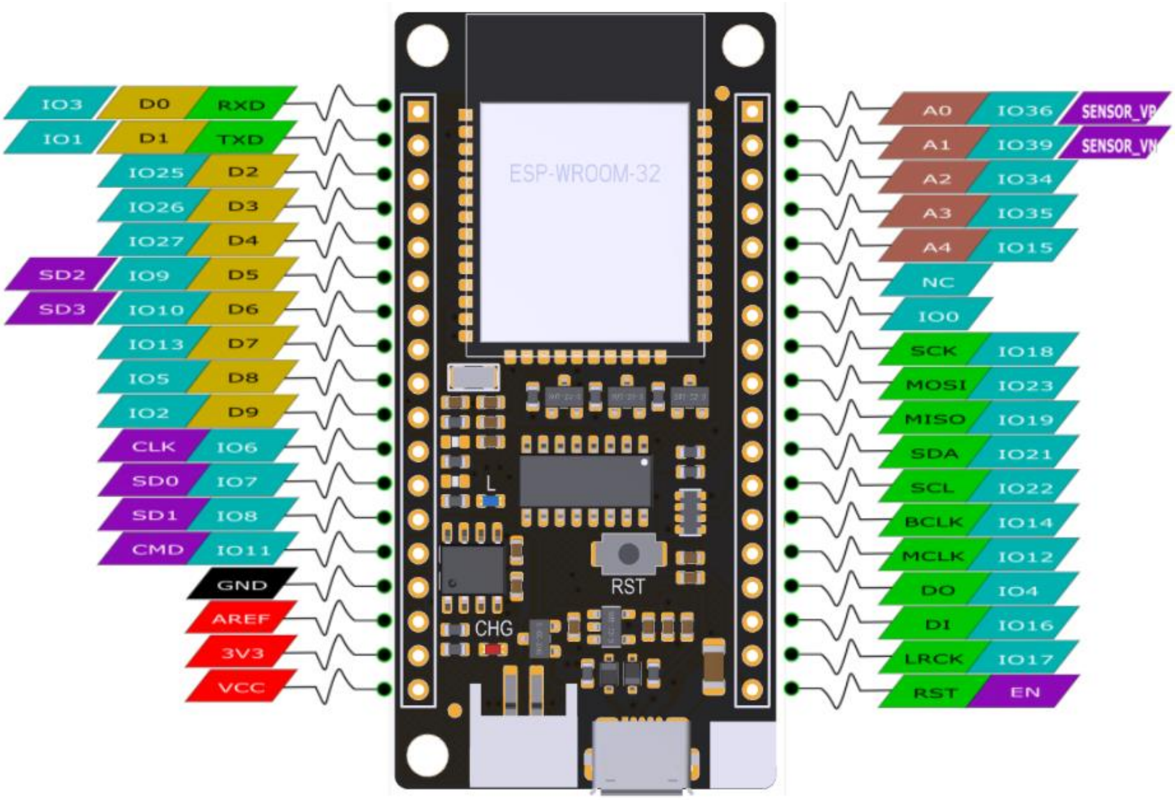


Figure 2: FireBeetle-ESP32 Board Overview

3.1.2 Dust Sensor

Figure 3: Sharp GP2Y1010AU0F

The dust sensor used in the project is the Sharp GP2Y1010AU0F. It uses infrared LED that emits a beam of light into the air. As the emitted light beam encounters particles present in the air, such as dust, smoke, and other airborne particles, it will cause some of the light to be scattered in various directions. The sensor also has a photodetector that would measure the intensity of the light scattered by the particles and that is how we will get the readings from the sensor. The photodetector converts the light intensity into electricity signal which is typically analog in nature. Processing of ADC would be done by the micro-controller to finally get the reading in

ug/m³.

3.2 Software/IoT Cloud Platform

3.2.1 Arduino IDE

Figure 4: Arduino IDE

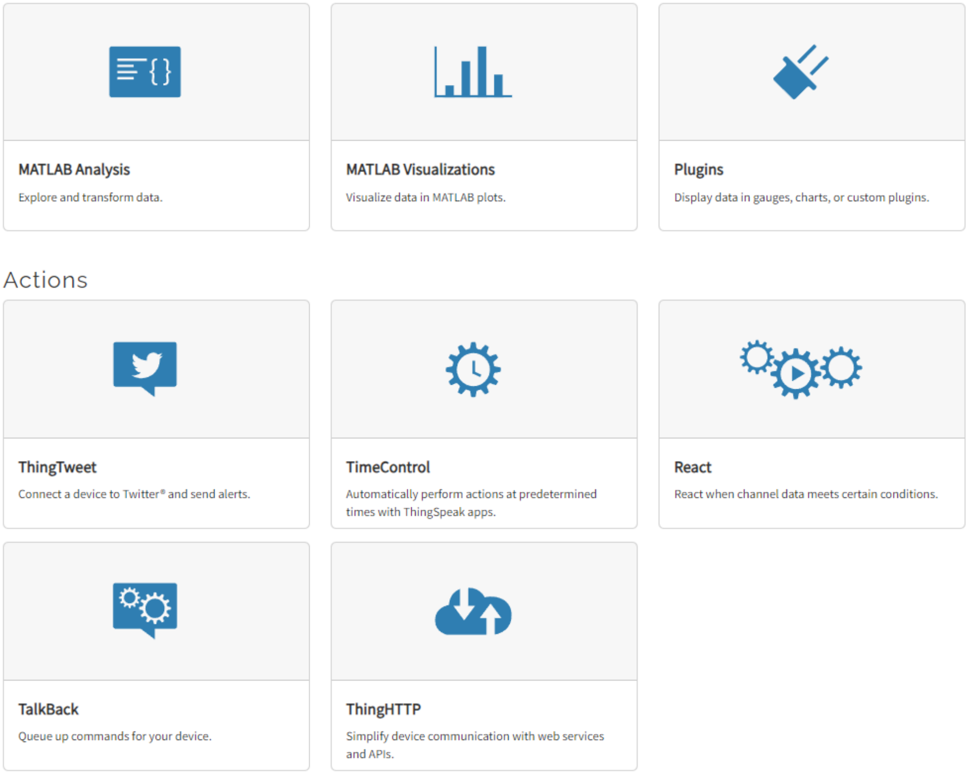
To program the micro-controller, Arduino Integrated Development Environments (IDE) will be used. Arduino IDE is widely known and used platform for programming Arduino boards, but it also supports programming the ESP32. It is also a beginner friendly environment with simplified programming based on C/C++.

Arduino has a large and active community of developers, makers, and hobbyists. There are also various documentation, tutorials and libraries for different sensors, modules and other peripherals which also include the ESP32.

3.2.2 ThingSpeak

Figure 5: ThingSpeak

ThingSpeak is an IoT platform developed by MathWorks, who is also the company behind MATLAB. They provide an integrated cloud service for data collection, analyzing and visualizing data from IoT devices such as micro-controllers which is used in this project. ThingSpeak offers a wide range of features and tools that can help simplify the process of building of IoT systems.



MATLAB Analysis
Explore and transform data.

MATLAB Visualizations
Visualize data in MATLAB plots.

Plugins
Display data in gauges, charts, or custom plugins.

Actions

ThingTweet
Connect a device to Twitter® and send alerts.

TimeControl
Automatically perform actions at predetermined times with ThingSpeak apps.

React
React when channel data meets certain conditions.

TalkBack
Queue up commands for your device.

ThingHTTP
Simplify device communication with web services and APIs.

Figure 6: ThingSpeak Features

ThingSpeak also enables us to monitor the dust density level in real time, sending alerts and having historical data which can be used to monitor trends as the data log accumulates over time.

3.2.3 Telegram Bot

Figure 7: Telegram

Telegram is a cloud-based instant messaging platform and voice-over IP service developed by Telegram Messenger LLP. It is the fourth most popular messaging application globally after WhatsApp, WeChat and Facebook Messenger [10]. According to Turner [10], it has an estimated 1562 billion total users and 800 million monthly active users in 2024.

Telegram also allows the creation of bots that enables users to interact and access different services. Telegram bots automate interactions with the users which allows you to provide immediate responses and various services without any human intervention. Its versatility, scalability and outreach are also the traits that makes it stand out.

3.2.3 Twilio

Figure 8: Twilio

Twilio is a cloud communications platform that allows integration of several communication methods such as voice calls, text messages (SMS) and video calls into an application or system using their APIs. This will be used in our system to send out alerts with the help of ThingSpeak's React and MATLAB analysis.

Chapter 4 Methodology

This chapter will be on the hardware assembly process and the development of the software through Arduino IDE and linkage to ThingSpeak.

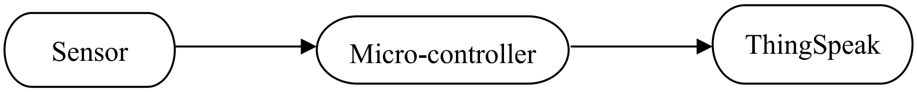


Figure 9: Overview of the system

4.1 Hardware Assembly

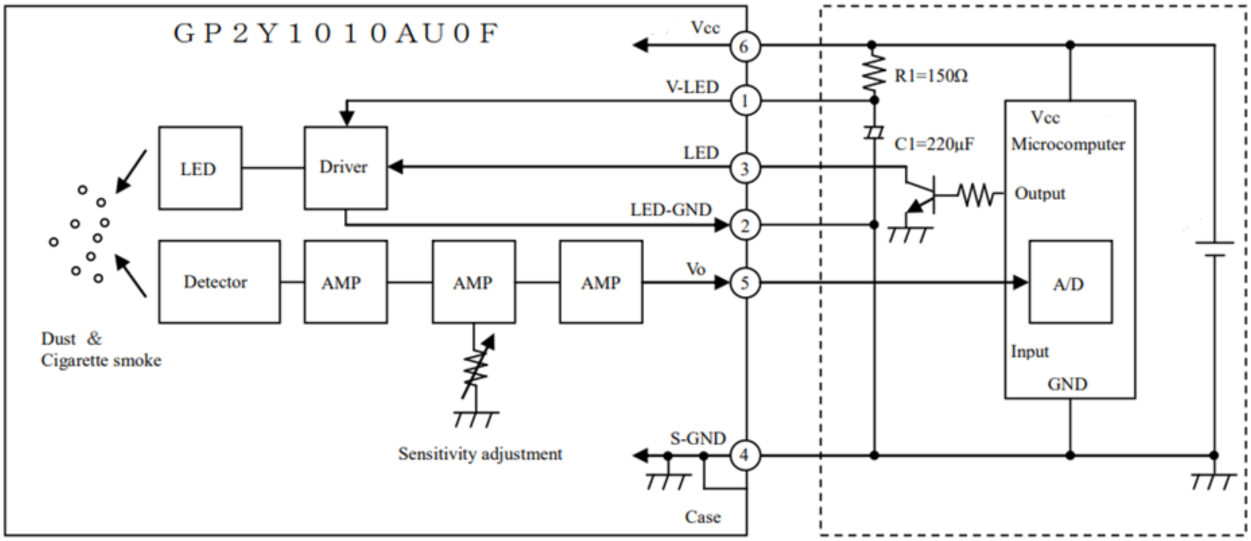


Figure 10: Schematic Diagram of sensor and micro-controller

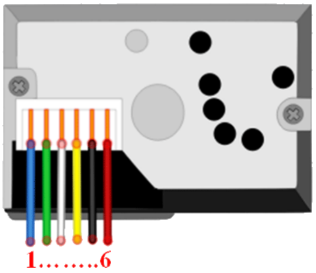


Figure 11: Pin numbering of sensor

Sharp GY2Y1010AU0F (Pin No.)	ESP32
Vcc (6)	Positive supply pin. Connect to supply of ESP32

V-LED (1)	Connect to supply through a 150ohm resistor
LED (3)	Connect to any digital any digital pin of ESP32. Used to toggle LED On/Off
LED-GND (2)	Connect to GND
Vo (5)	Connect to any Analog pin of ESP32. Sensor analog output
S-GND (4)	Connect to GND

Table 1: Pin connections

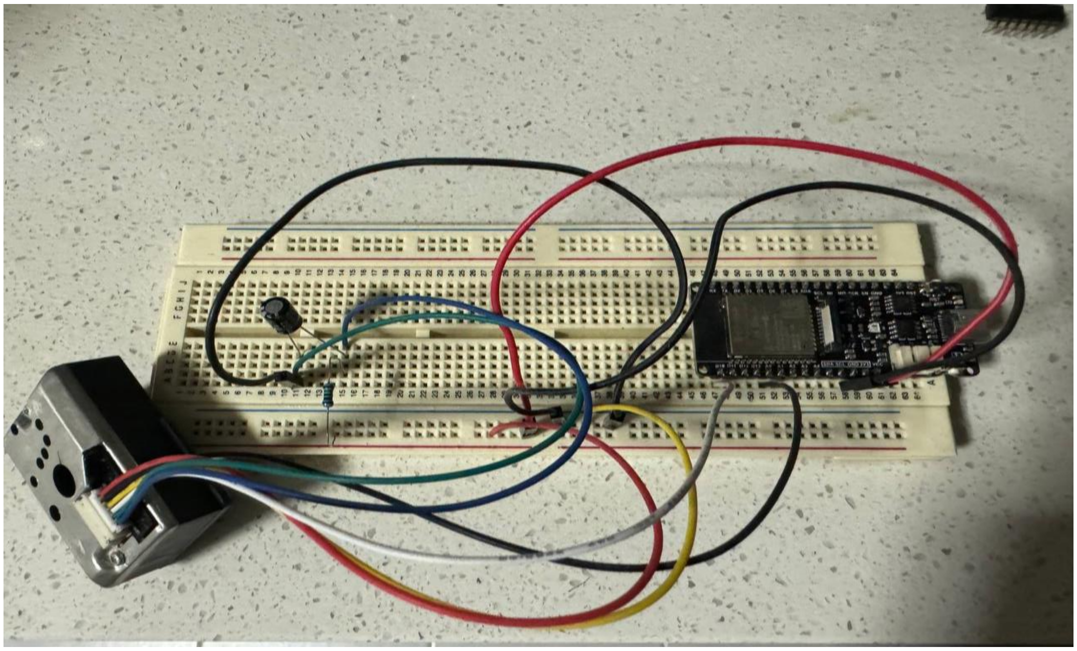


Figure 12: Connected circuit

The figure above depicts the circuit layout following the completion of necessary connections.

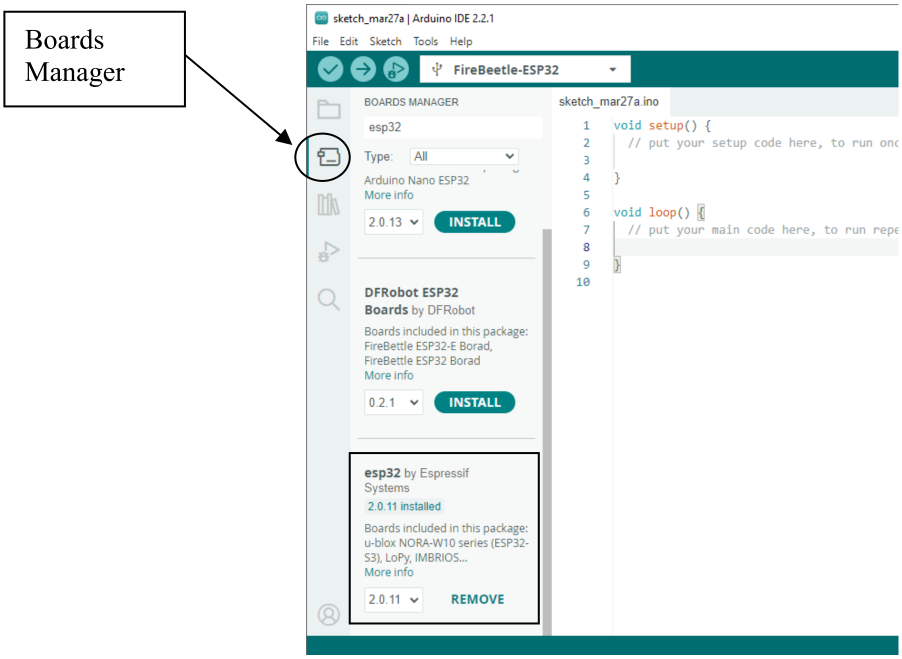
Hardware used:

- Breadboard
- ESP32
- Dust Sensor
- 150Ω resistor x1
- 220µF capacitor x1
- Jumper Wires

To check if the circuit is working as intended, we must first write the code to check if the basic function which is to scan the air for dust is working well. Next would be the software development.

4.2 Software Development

4.2.1 Basic Test



esp32

Type: All

Arduino Nano ESP32
More info
2.0.13 INSTALL

DFRobot ESP32 Boards by DFRobot
Boards included in this package:
FireBettle ESP32-E Borad,
FireBettle ESP32 Borad
More info
0.2.1 INSTALL

esp32 by Espressif Systems
2.0.11 installed
Boards included in this package:
u-blox NORA-W10 series (ESP32-S3), LoPy, IMBRIOS...
More info
2.0.11 REMOVE

```
void setup() {
  // put your setup code here, to run onc

}

void loop() {
  // put your main code here, to run repe

}
```

Figure 13: View of Board Manager

Since we are using the ESP32, we first need to install the ESP32 Boards from the "Boards Manager". Usually if you are using an Arduino micro-controller, the IDE can auto detect the board you are using and set the target to that specific board.

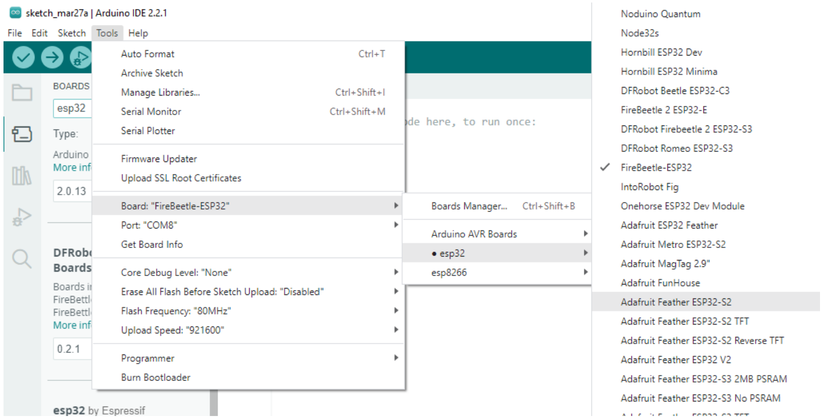
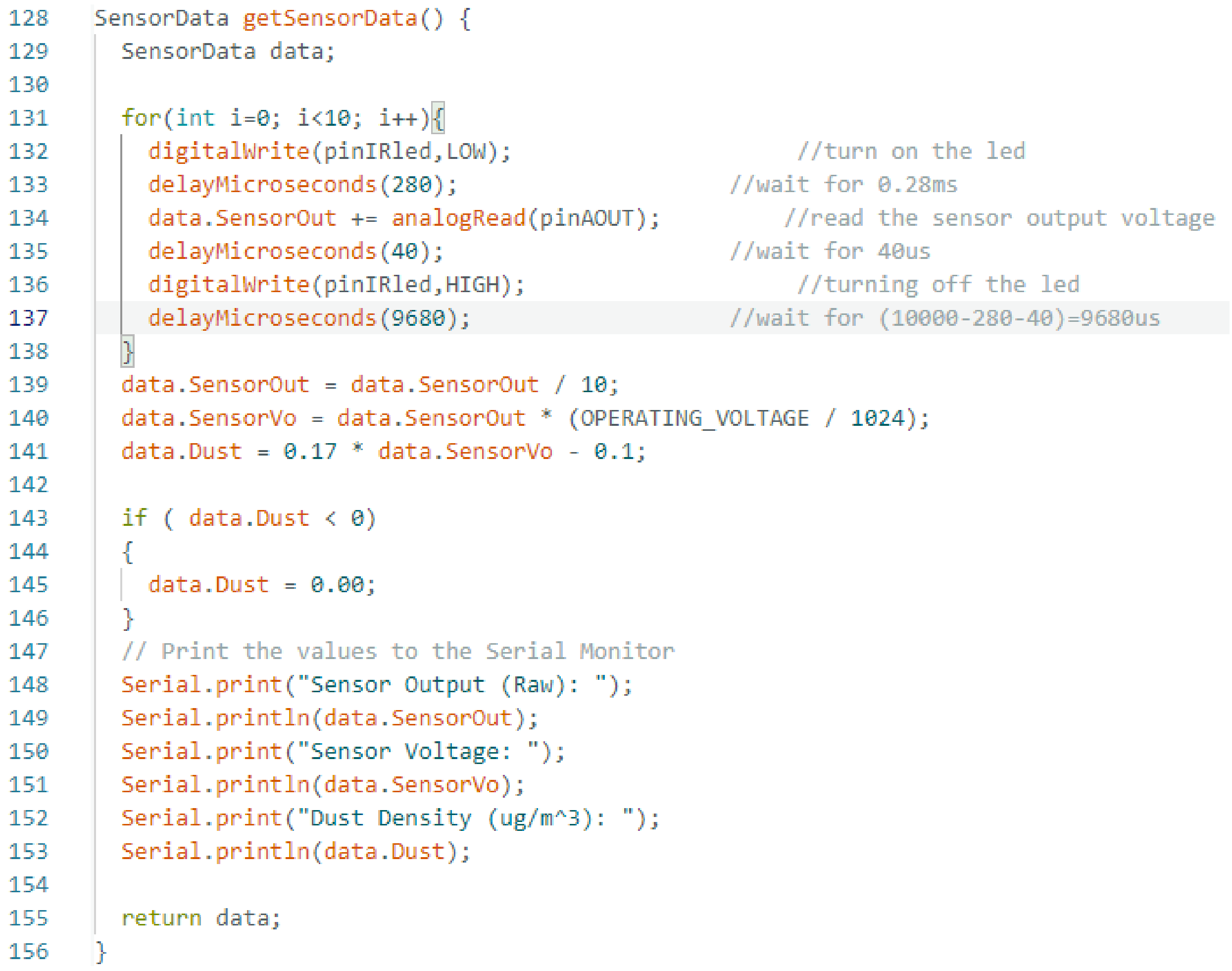


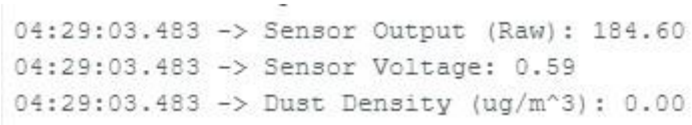
Figure 14: How to select board

After installation, select the ESP32 board you are using. You can select from: **Tools** > **Board:** > **esp32** > **”yourboard”.**



```
SensorData getSensorData() {
  SensorData data;

  for(int i=0; i<10; i++){
    digitalWrite(pinIRled,LOW);                        //turn on the led
    delayMicroseconds(280);                       //wait for 0.28ms
    data.SensorOut += analogRead(pinAOUT);            //read the sensor output voltage
    delayMicroseconds(40);                        //wait for 40us
    digitalWrite(pinIRled,HIGH);                       //turning off the led
    delayMicroseconds(9680);                      //wait for (10000-280-40)=9680us
  }
  data.SensorOut = data.SensorOut / 10;
  data.SensorVo = data.SensorOut * (OPERATING_VOLTAGE / 1024);
  data.Dust = 0.17 * data.SensorVo - 0.1;

  if ( data.Dust < 0)
  {
    data.Dust = 0.00;
  }
  // Print the values to the Serial Monitor
  Serial.print("Sensor Output (Raw): ");
  Serial.println(data.SensorOut);
  Serial.print("Sensor Voltage: ");
  Serial.println(data.SensorVo);
  Serial.print("Dust Density (ug/m^3): ");
  Serial.println(data.Dust);

  return data;
}
```

Figure 15: Sensor scanning code



```
04:29:03.483 -> Sensor Output (Raw): 184.60
04:29:03.483 -> Sensor Voltage: 0.59
04:29:03.483 -> Dust Density (ug/m^3): 0.00
```

Figure 16: Serial Monitor output

Figure 15 and 16 show a snippet of the code for the sensor and the output seen from the serial monitor. According to the dust sensor's datasheet, for a single scan, the LED must be turned on and off for 0.28ms and 40µs respectively. After turning off the LED, it must wait for another 9690µs at the minimum before another round of scanning can be ran again.

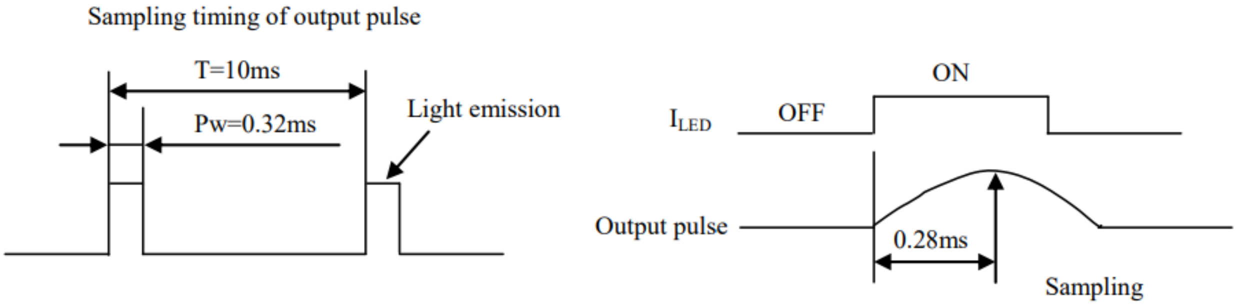
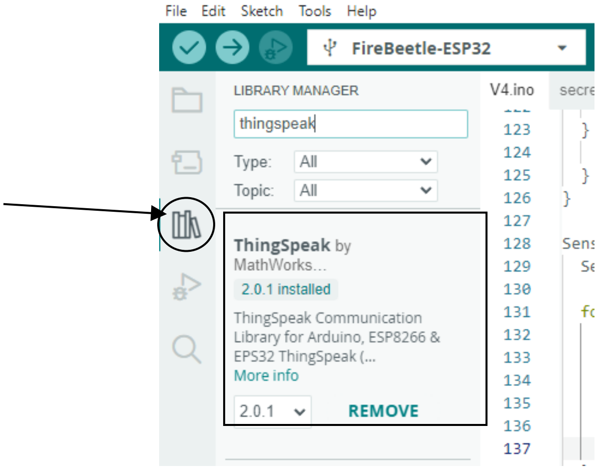


Figure 17: Sensor's recommended period

Parameter	Symbol	Specified condition	Recommended condition	Unit
Pulse cycle	T	10	10±1	ms
Pulse width	Pw	0.32	0.32±0.02	ms

Table 2: Sensor's pulse cycle

Next it would be to send the sensor data to ThingSpeak to enable remote monitoring and data logging. To ease the process on the coding, there is a library to be installed called "ThingSpeak" and it is made by MathWorks themselves. Without installing the library, one would still be able to send and retrieve data from the ThingSpeak platform, but one would need to manually utilize the ThingSpeak API for that.



thingspeak

Type: All

Topic: All

ThingSpeak by MathWorks...

2.0.1 installed

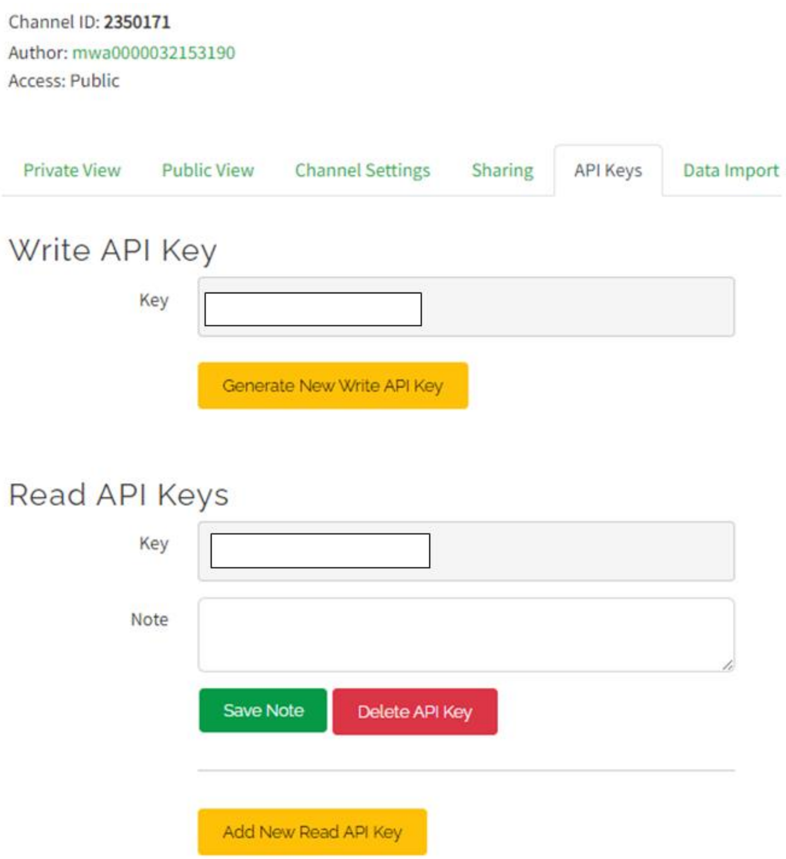
ThingSpeak Communication Library for Arduino, ESP8266 & EPS32 ThingSpeak (...

2.0.1

Figure 18: ThingSpeak Library

4.2.2 ThingSpeak setup

On ThingSpeak, you can create an account and subsequently your own channel. Each channel has its own unique read and write keys which would enable you to read and write to and from the channel that is associated to that set of unique keys.



Channel ID: **2350171**
Author: mwa0000032153190
Access: Public

Write API Key

Key [redacted]

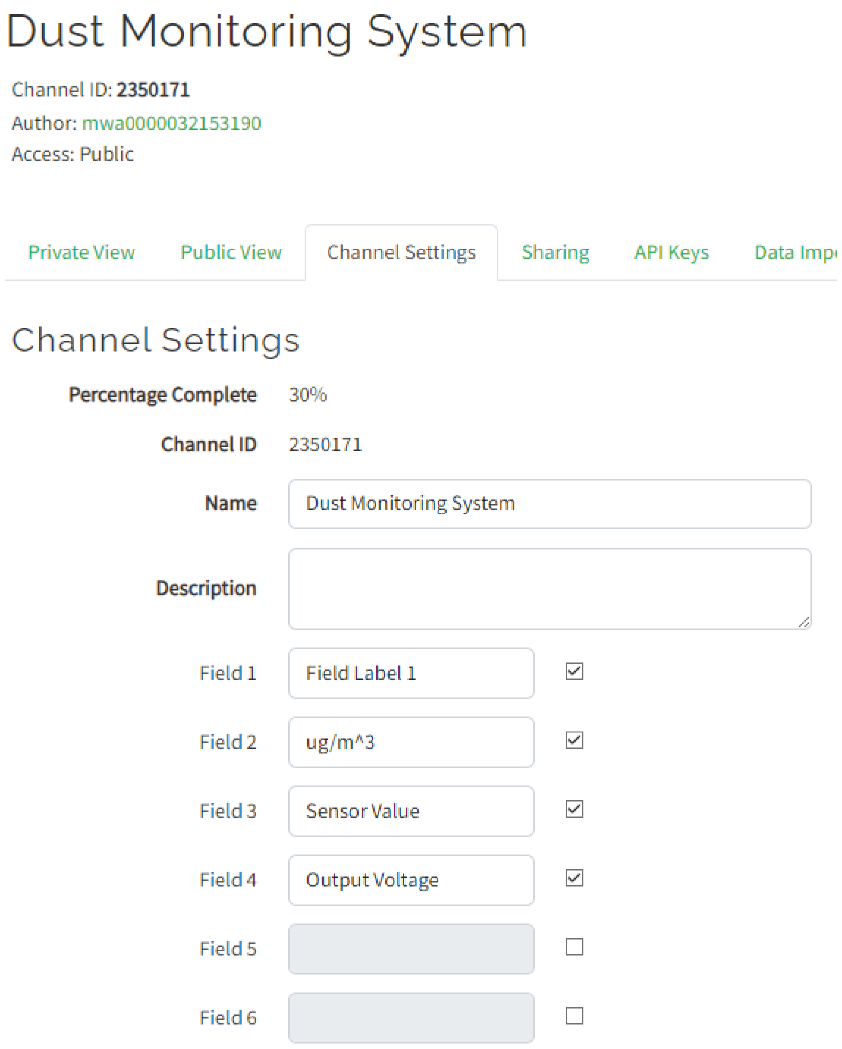
Read API Keys

Key [redacted]

Note

Figure 19: ThingSpeak API Keys

As seen in Figure 19, one is able to generate new keys if the key information is compromised or leaked. In order to keep the keys to yourself as anyone would be able to enter data to your channel without you knowing.



Dust Monitoring System

Channel ID: **2350171**
Author: mwa0000032153190
Access: Public

Channel Settings

Percentage Complete 30%

Channel ID 2350171

Name Dust Monitoring System

Description

Field 1 Field Label 1 ☑

Field 2 ug/m^3 ☑

Field 3 Sensor Value ☑

Field 4 Output Voltage ☑

Field 5 □

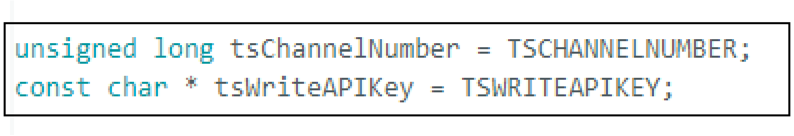
Field 6 □

Figure 20: ThingSpeak Channel Fields

On the channel settings page, it is able to view and edit the basic information of the channel. Fields are where data sent to the channel are being stored and tracked. In this example, Field 2 is where the calculated dust density level is being stored at. For error checking purposes, other values like the sensor output voltage which is required to calculate the dust density is also being stored.

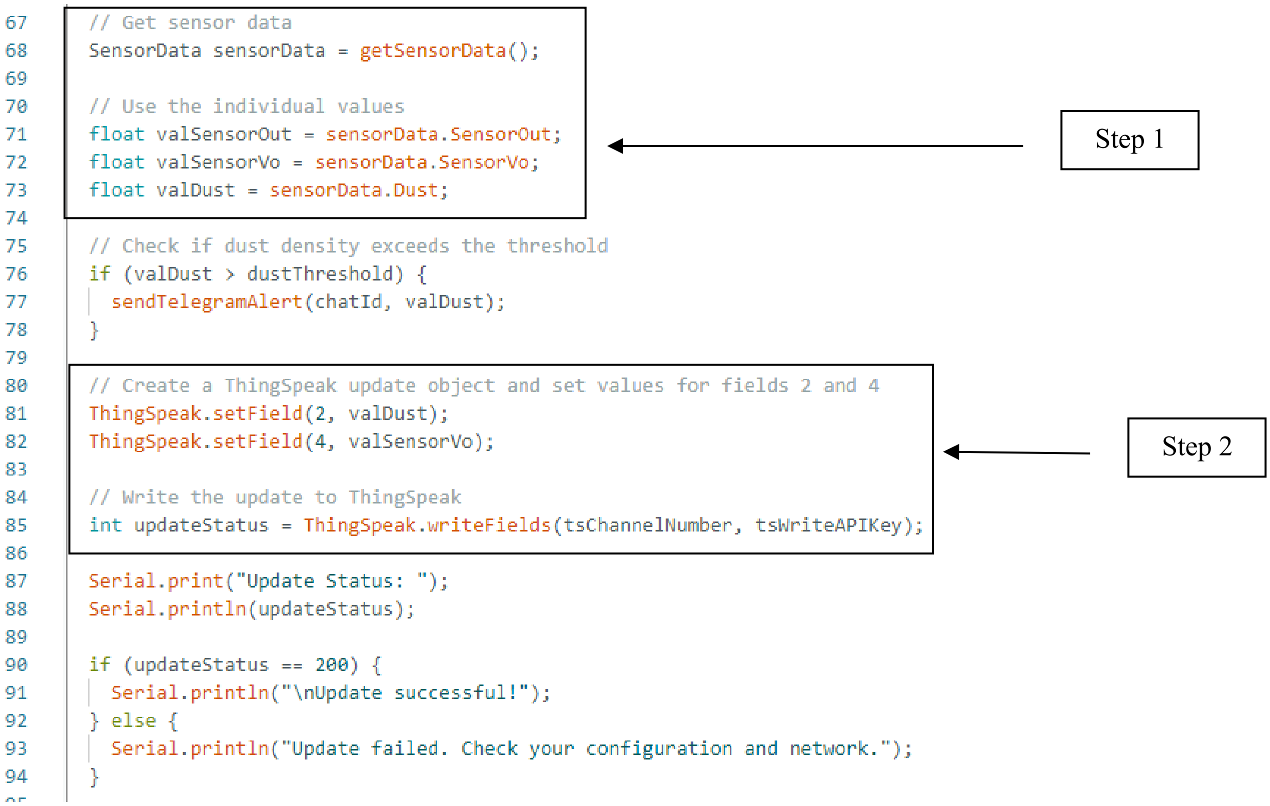
4.2.3 Integrating and sending data to ThingSpeak

In this section, we will be going through the process of sending the read sensor data to ThingSpeak with the help of the installed library.



```
unsigned long tsChannelNumber = TSCHANNELNUMBER;
const char * tsWriteAPIKey = TSWRITEAPIKEY;
```

Figure 21: ThingSpeak keys declared in code



```
// Get sensor data
SensorData sensorData = getSensorData();

// Use the individual values
float valSensorOut = sensorData.SensorOut;
float valSensorVo = sensorData.SensorVo;
float valDust = sensorData.Dust;

// Check if dust density exceeds the threshold
if (valDust > dustThreshold) {
  sendTelegramAlert(chatId, valDust);
}

// Create a ThingSpeak update object and set values for fields 2 and 4
ThingSpeak.setField(2, valDust);
ThingSpeak.setField(4, valSensorVo);

// Write the update to ThingSpeak
int updateStatus = ThingSpeak.writeFields(tsChannelNumber, tsWriteAPIKey);

Serial.print("Update Status: ");
Serial.println(updateStatus);

if (updateStatus == 200) {
  Serial.println("\nUpdate successful!");
} else {
  Serial.println("Update failed. Check your configuration and network.");
}
```

Figure 22: Sending to Channel

Figure 22 is a snippet of the code that highlights the key parts of sending the data to the channel. In step 1, the function getSensorData() is called to start off the sensor to read the air particles present. After the function has been executed, the data will then be stored in variables which will be used to send to the channel.

Step 2 shows how the ThingSpeak library is being used. ThingSpeak.setField() essentially is where you set the field number and which data that needs to be send. After setting up the object, the function ThingSpeak.writeFields(tsChannelNumber, tsWriteAPIKey) can be called to send the data. In Figure 21, the Channel ID or number and the write API Key has been included to be used in the loop. When the writeFields() function is called, it will then take all the declared variables, objects and their field number and execute the HTTP request.

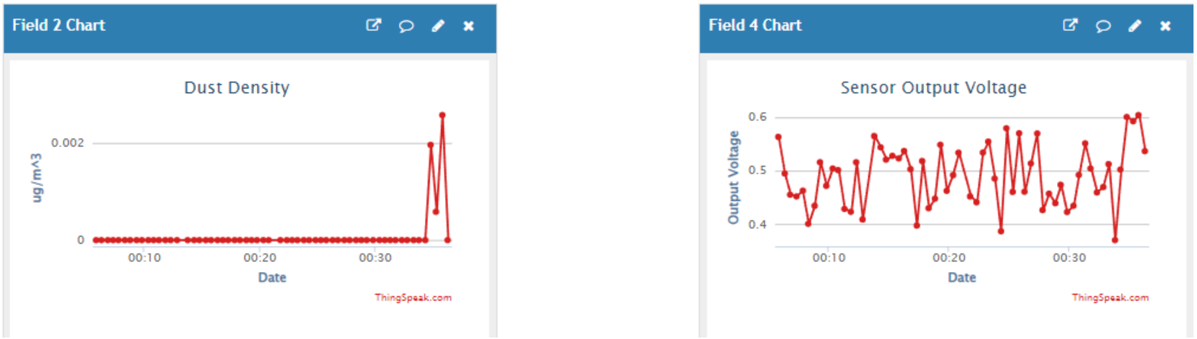
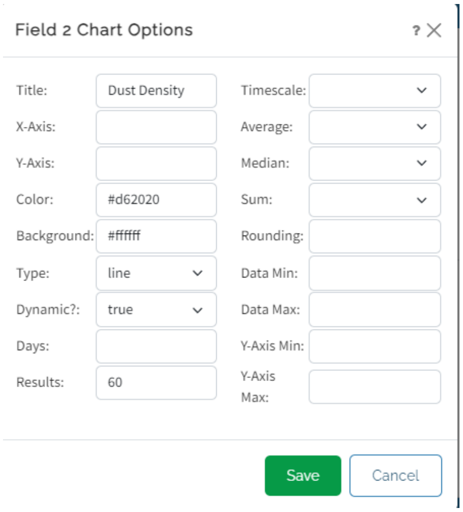


Figure 23: Data Chart of Dust Density and Output Voltage

If the data is sent successfully, dust density can be monitored from the channel's view thus, enabling us to remotely monitor the dust level remotely as we intended. The channel's field view can also be manipulated as well from the field settings.



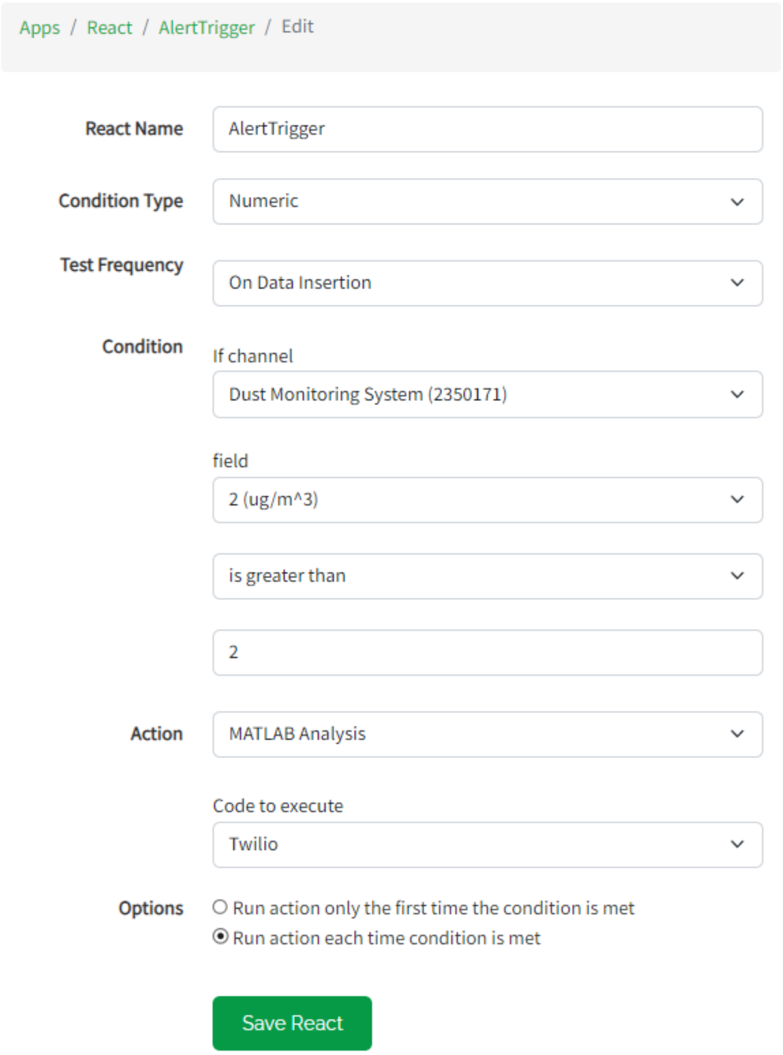
Field 2 Chart Options

Title:	Dust Density	Timescale:	
X-Axis:		Average:	
Y-Axis:		Median:	
Color:	#d62020	Sum:	
Background:	#ffffff	Rounding:	
Type:	line	Data Min:	
Dynamic?:	true	Data Max:	
Days:		Y-Axis Min:	
Results:	60	Y-Axis Max:	

Figure 24: Field Chart Settings

4.2.3 Alerts

To send out alerts, there are several ways to achieve that. One way will be through the ThingSpeak features and applications.



Apps / React / AlertTrigger / Edit

React Name: AlertTrigger

Condition Type: Numeric

Test Frequency: On Data Insertion

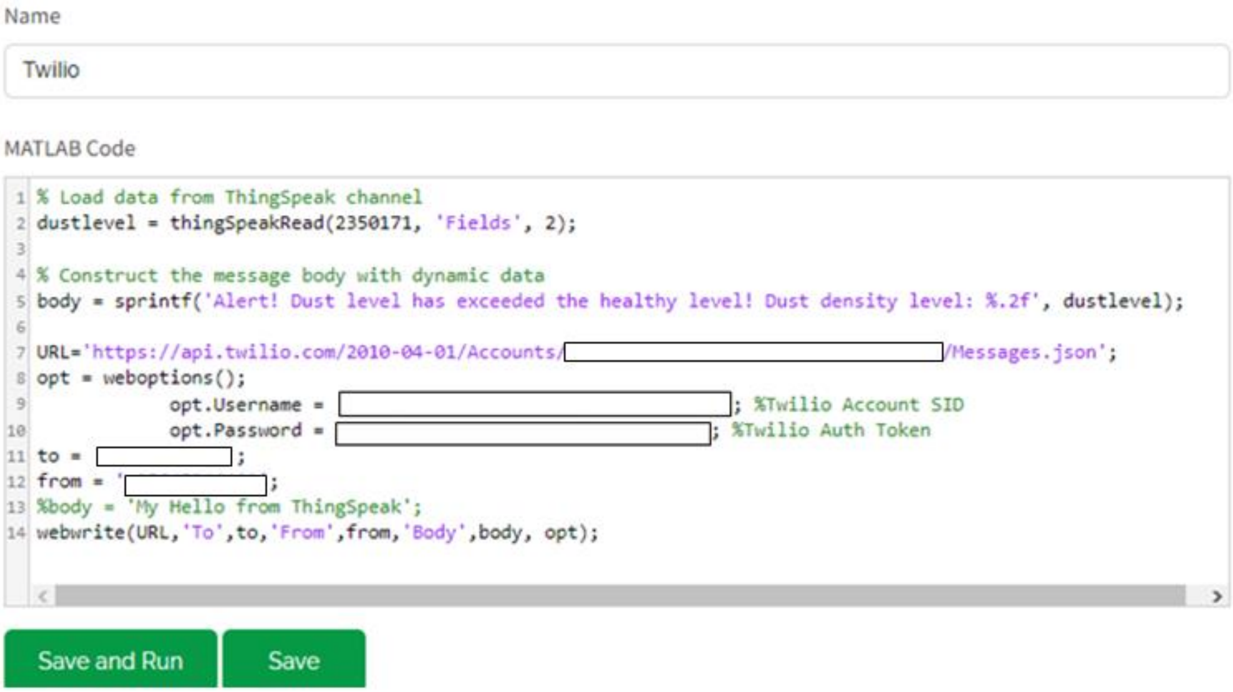
Condition:
If channel: Dust Monitoring System (2350171)
field: 2 (ug/m^3)
is greater than
2

Action: MATLAB Analysis
Code to execute: Twilio

Options:
○ Run action only the first time the condition is met
● Run action each time condition is met

Figure 25: React settings

On the react settings, we can set for this reaction to happen on certain conditions. For this purpose, it is set to check if the react should be executed upon data insertion. If the data inserted is greater than the threshold set, it will execute the action. In Figure 25, the action would be to execute the MATLAB Analysis called “Twilio”.

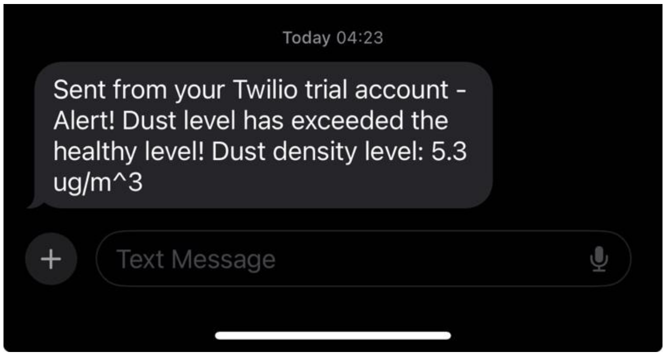


Twilio

```
% Load data from ThingSpeak channel
dustlevel = thingSpeakRead(2350171, 'Fields', 2);

% Construct the message body with dynamic data
body = sprintf('Alert! Dust level has exceeded the healthy level! Dust density level: %.2f', dustlevel);

URL='https://api.twilio.com/2010-04-01/Accounts/[redacted]/Messages.json';
opt = weboptions();
            opt.Username = [redacted]; %Twilio Account SID
            opt.Password = [redacted]; %Twilio Auth Token
to = [redacted];
from = '[redacted];
%body = 'My Hello from ThingSpeak';
webwrite(URL,'To',to,'From',from,'Body',body, opt);
```

Figure 26: MATLAB Analysis code

Once the react is triggered, the code above will be executed. It will take the data from the channel and insert it into the body of the alert message that will be sent. URL, account SID, Auth token and a virtual number can be gotten from Twilio after registration.

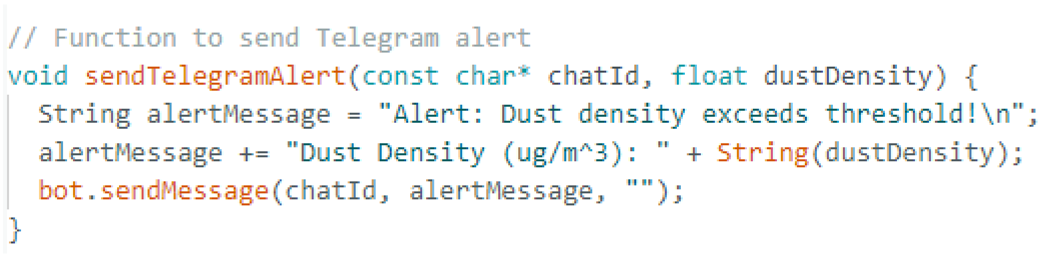

Today 04:23

Sent from your Twilio trial account - Alert! Dust level has exceeded the healthy level! Dust density level: 5.3 ug/m^3

Figure 27: Twilio SMS Alert

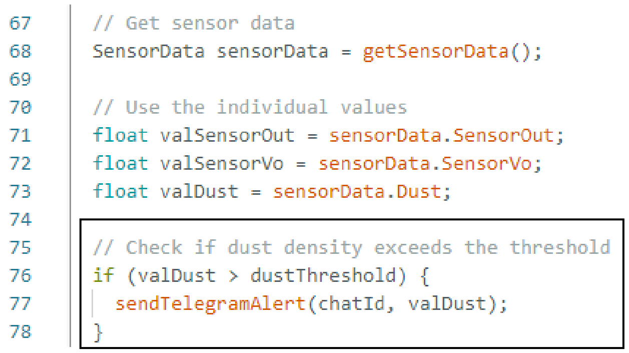
Figure 27 shows the alert message that is received once the MATLAB analysis is executed.

Another method of receiving alert messages would be with the usage of the telegram bot. To send alerts to the telegram bot, it will be done through the Arduino IDE itself.



```
// Function to send Telegram alert
void sendTelegramAlert(const char* chatId, float dustDensity) {
  String alertMessage = "Alert: Dust density exceeds threshold!\n";
  alertMessage += "Dust Density (ug/m^3): " + String(dustDensity);
  bot.sendMessage(chatId, alertMessage, "");
}
```

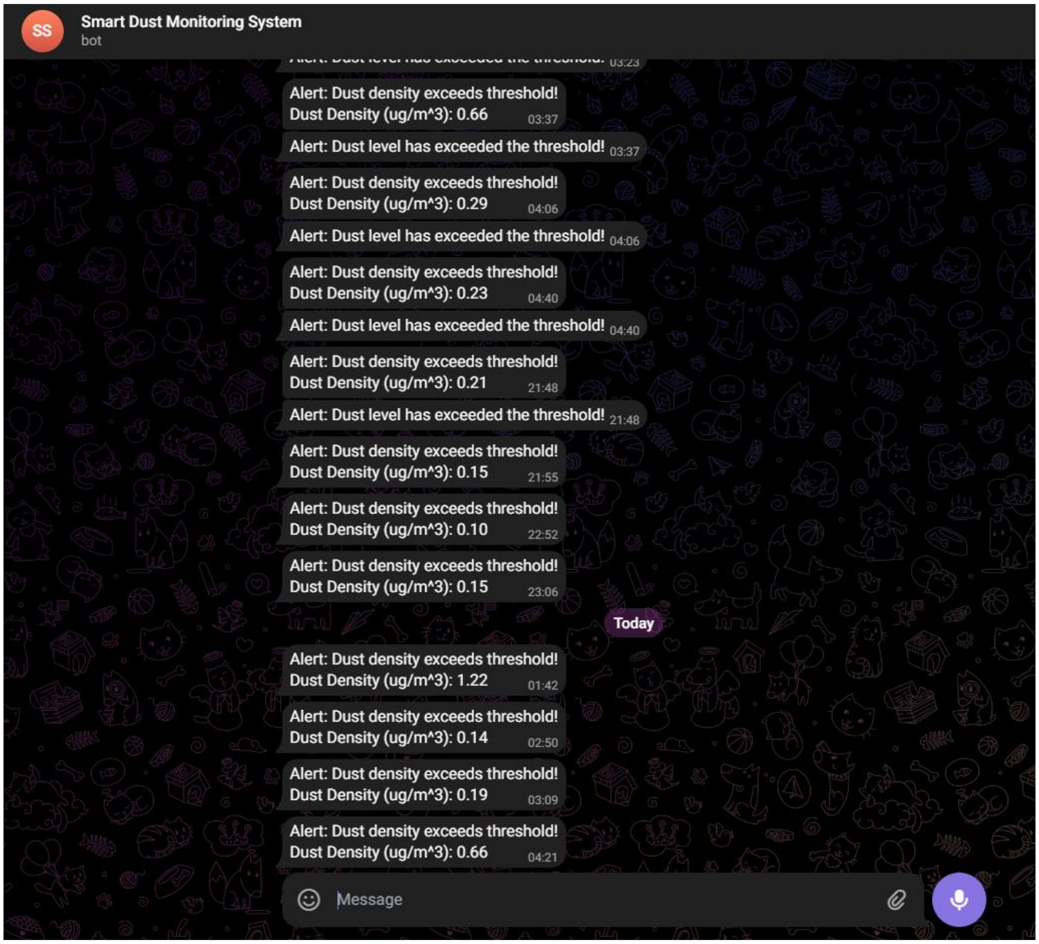
Figure 28: Function for telegram alert



```
// Get sensor data
SensorData sensorData = getSensorData();

// Use the individual values
float valSensorOut = sensorData.SensorOut;
float valSensorVo = sensorData.SensorVo;
float valDust = sensorData.Dust;

// Check if dust density exceeds the threshold
if (valDust > dustThreshold) {
  sendTelegramAlert(chatId, valDust);
}
```

Figure 29: Calling the function

As seen from Figure 29, once the sensor reading is taken, it will go through the if statement to check if the dust level is above the threshold. If it exceeds the threshold, it will proceed to call the function sendTelegramAlert( ).



Alert: Dust level has exceeded the threshold! 03:23

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.66 03:37

Alert: Dust level has exceeded the threshold! 03:37

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.29 04:06

Alert: Dust level has exceeded the threshold! 04:06

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.23 04:40

Alert: Dust level has exceeded the threshold! 04:40

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.21 21:48

Alert: Dust level has exceeded the threshold! 21:48

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.15 21:55

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.10 22:52

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.15 23:06

Today

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 1.22 01:42

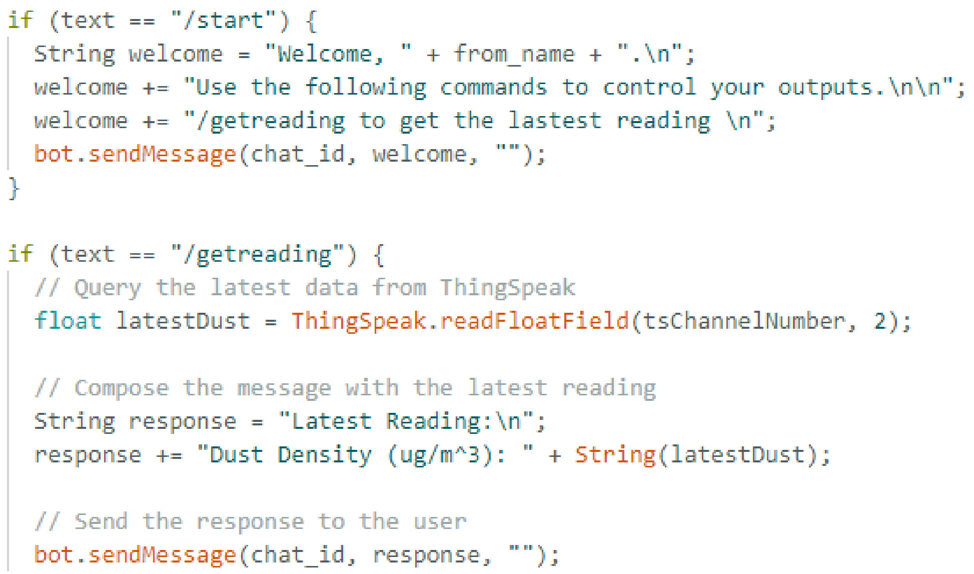
Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.14 02:50

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.19 03:09

Alert: Dust density exceeds threshold!
Dust Density (ug/m^3): 0.66 04:21

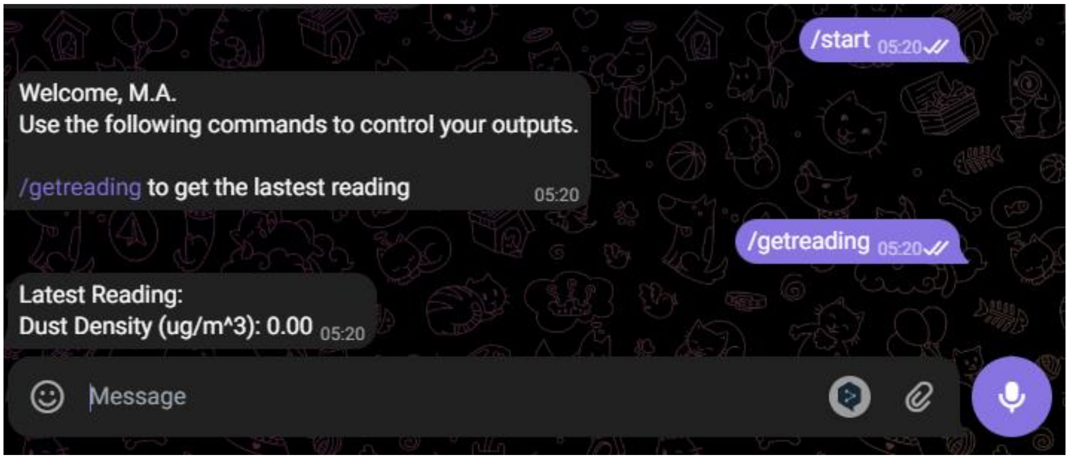
Figure 30: Telegram Bot Alert message

As seen in Figure 30, that is how the telegram alert will look like. For testing purposes, the threshold has been lowered to trigger the respond. The other advantage of using telegram bot would be the ability to set certain commands like getting latest reading although it does not exceed the alert threshold.



```
if (text == "/start") {
  String welcome = "Welcome, " + from_name + ".\n";
  welcome += "Use the following commands to control your outputs.\n\n";
  welcome += "/getreading to get the lastest reading \n";
  bot.sendMessage(chat_id, welcome, "");
}

if (text == "/getreading") {
  // Query the latest data from ThingSpeak
  float latestDust = ThingSpeak.readFloatField(tsChannelNumber, 2);

  // Compose the message with the latest reading
  String response = "Latest Reading:\n";
  response += "Dust Density (ug/m^3): " + String(latestDust);

  // Send the response to the user
  bot.sendMessage(chat_id, response, "");
```

Figure 31: Bot commands code



/start 05:20

Welcome, M.A.
Use the following commands to control your outputs.

/getreading to get the lastest reading 05:20

/getreading 05:20

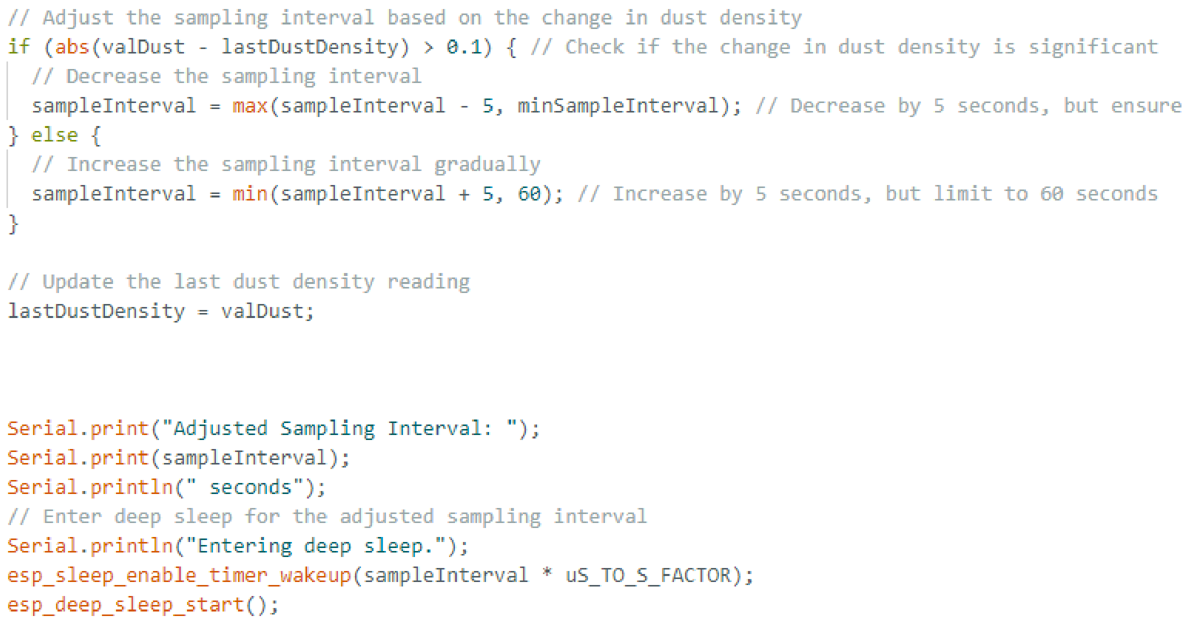
Latest Reading:
Dust Density (ug/m^3): 0.00 05:20

Figure 32: Bot commands

Figure 32 shows how the bot replies to the commands “/start” and “/getreading”. In this example, “/start” command will welcome the user and informs the user of what commands are available to be called. The command “/getreading” will fetch the latest reading from the ThingSpeak channel field and proceed to send the message to the user. The capabilities of the bot's command feature can be more extensively utilized.

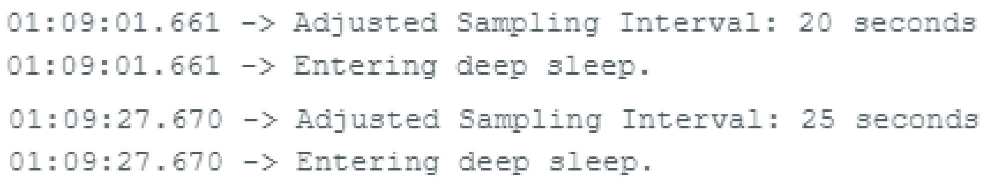
4.3 Dynamic Sampling Rate

Dynamic sampling rate is implemented to reduce the energy consumption of system. As transmitting data over wireless networks can consume a significant amount of energy, reducing the frequency of readings being taken can reduce the overall energy consumption and thus extending the battery life when running on a battery.



```
// Adjust the sampling interval based on the change in dust density
if (abs(valDust - lastDustDensity) > 0.1) { // Check if the change in dust density is significant
  // Decrease the sampling interval
  sampleInterval = max(sampleInterval - 5, minSampleInterval); // Decrease by 5 seconds, but ensure
} else {
  // Increase the sampling interval gradually
  sampleInterval = min(sampleInterval + 5, 60); // Increase by 5 seconds, but limit to 60 seconds
}

// Update the last dust density reading
lastDustDensity = valDust;



Serial.print("Adjusted Sampling Interval: ");
Serial.print(sampleInterval);
Serial.println(" seconds");
// Enter deep sleep for the adjusted sampling interval
Serial.println("Entering deep sleep.");
esp_sleep_enable_timer_wakeup(sampleInterval * uS_TO_S_FACTOR);
esp_deep_sleep_start();
```

Figure 33: Snippet of code for handling sampling rate



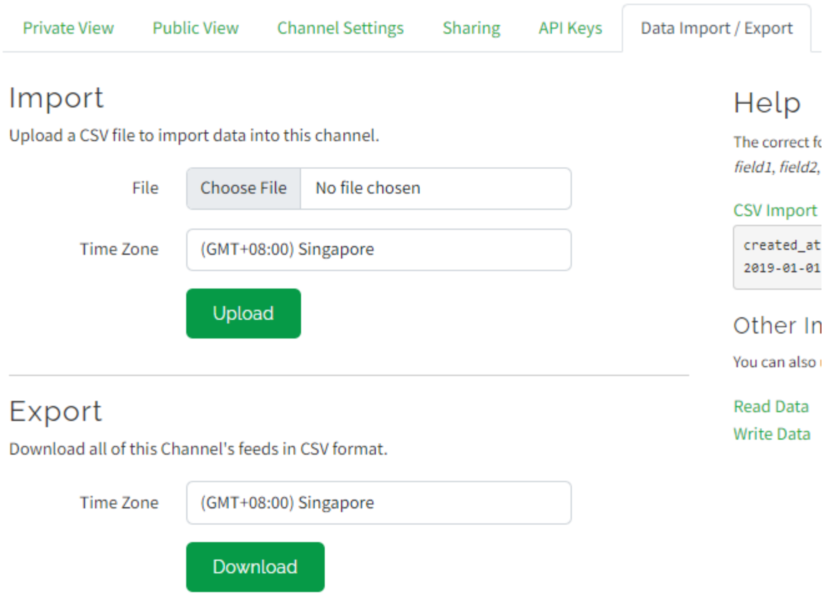
```
01:09:01.661 -> Adjusted Sampling Interval: 20 seconds
01:09:01.661 -> Entering deep sleep.
01:09:27.670 -> Adjusted Sampling Interval: 25 seconds
01:09:27.670 -> Entering deep sleep.
```

Figure 34: Example of dynamic timer

Figure 33 shows how the sampling rate is controlled. By making use of the ESP32's deep sleep mode where it will shut down the CPU and disable peripherals while persevering some states on its RAM during the deep sleep. While there are a few ways to wake it up from deep sleep, timer is being used here.

The global variable "sampleInterval" is used to control the duration of the timer. At the end of the loop before going into the deep sleep mode, the if-else statement determines if the sampling interval should be increased or decreased. There is also a maximum and minimum duration of 60 seconds and 10 seconds respectively to ensure that the sampling interval will not be too long or too short ensuring balance between being power efficient and maintaining its real-time monitoring purposes.

4.4 Data Analysis



Import

Upload a CSV file to import data into this channel.

File: No file chosen

Time Zone: (GMT+08:00) Singapore

Export

Download all of this Channel's feeds in CSV format.

Time Zone: (GMT+08:00) Singapore

Help

The correct fc

field1, field2,

created_at

2019-01-01

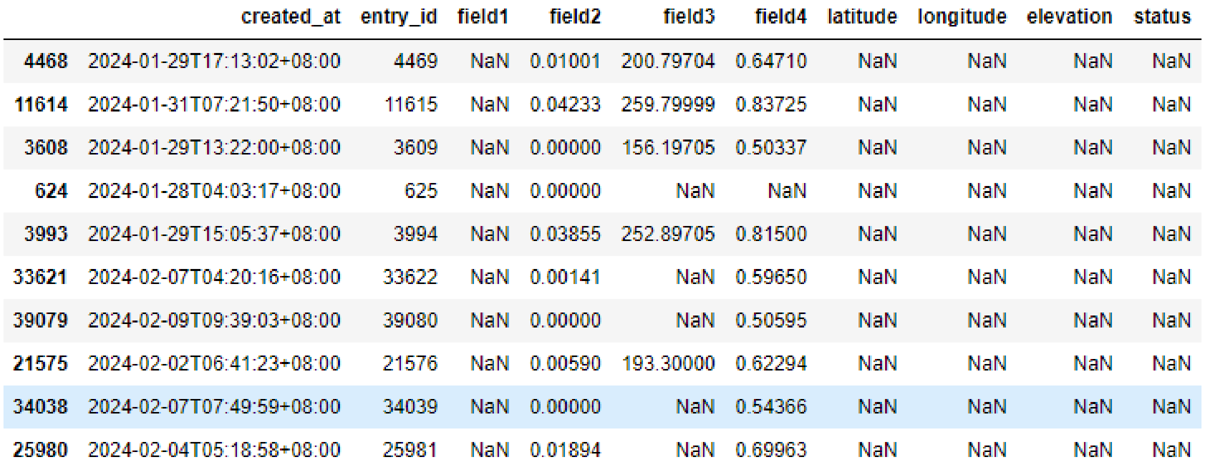
Other In

You can also

Figure 35: Exporting data from a channel

One of the other benefits of using ThingSpeak as mentioned earlier is the ability to have the historical data. With historical data, it enables data analysis to be done. This may be used to check for trends or any other correlation factors.

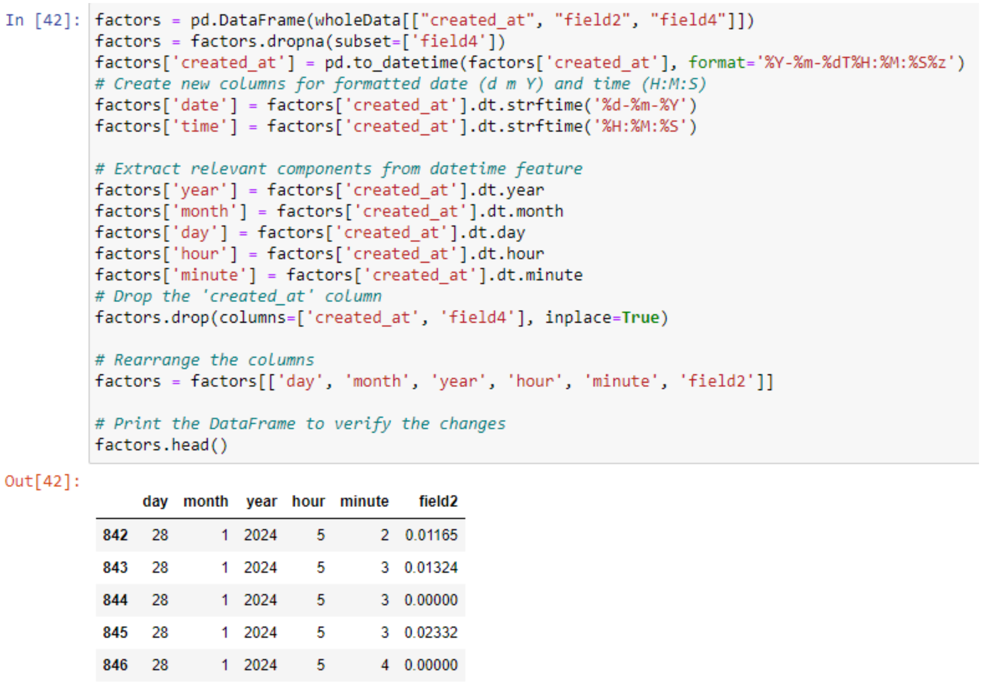
Exported data will be in a CSV format which will work well with “pandas” a python library for data analyzation. Data analysis will be done in the Jupyter Notebook environment with few other python libraries that will aid in data analysis and visualization.



	created_at	entry_id	field1	field2	field3	field4	latitude	longitude	elevation	status
4468	2024-01-29T17:13:02+08:00	4469	NaN	0.01001	200.79704	0.64710	NaN	NaN	NaN	NaN
11614	2024-01-31T07:21:50+08:00	11615	NaN	0.04233	259.79999	0.83725	NaN	NaN	NaN	NaN
3608	2024-01-29T13:22:00+08:00	3609	NaN	0.00000	156.19705	0.50337	NaN	NaN	NaN	NaN
624	2024-01-28T04:03:17+08:00	625	NaN	0.00000	NaN	NaN	NaN	NaN	NaN	NaN
3993	2024-01-29T15:05:37+08:00	3994	NaN	0.03855	252.89705	0.81500	NaN	NaN	NaN	NaN
33621	2024-02-07T04:20:16+08:00	33622	NaN	0.00141	NaN	0.59650	NaN	NaN	NaN	NaN
39079	2024-02-09T09:39:03+08:00	39080	NaN	0.00000	NaN	0.50595	NaN	NaN	NaN	NaN
21575	2024-02-02T06:41:23+08:00	21576	NaN	0.00590	193.30000	0.62294	NaN	NaN	NaN	NaN
34038	2024-02-07T07:49:59+08:00	34039	NaN	0.00000	NaN	0.54366	NaN	NaN	NaN	NaN
25980	2024-02-04T05:18:58+08:00	25981	NaN	0.01894	NaN	0.69963	NaN	NaN	NaN	NaN

Figure 36: Exported data before cleaning

Figure 36 shows the CSV file that was exported and before going through the data cleaning. The process of data cleaning is to improve accuracy of the analysis where errors and inconsistency in the data can lead to incorrect conclusions and decisions. Reliability will also be improved as it reduces the likelihood of biasedness or misleading conclusions caused by outliers, missing values, or incorrect data.



In [42]:
```
factors = pd.DataFrame(wholeData[["created_at", "field2", "field4"]])
factors = factors.dropna(subset=['field4'])
factors['created_at'] = pd.to_datetime(factors['created_at'], format='%Y-%m-%dT%H:%M:%S%z')
# Create new columns for formatted date (d m Y) and time (H:M:S)
factors['date'] = factors['created_at'].dt.strftime('%d-%m-%Y')
factors['time'] = factors['created_at'].dt.strftime('%H:%M:%S')

# Extract relevant components from datetime feature
factors['year'] = factors['created_at'].dt.year
factors['month'] = factors['created_at'].dt.month
factors['day'] = factors['created_at'].dt.day
factors['hour'] = factors['created_at'].dt.hour
factors['minute'] = factors['created_at'].dt.minute
# Drop the 'created_at' column
factors.drop(columns=['created_at', 'field4'], inplace=True)

# Rearrange the columns
factors = factors[['day', 'month', 'year', 'hour', 'minute', 'field2']]

# Print the DataFrame to verify the changes
factors.head()
```

Out[42]:

	day	month	year	hour	minute	field2
842	28	1	2024	5	2	0.01165
843	28	1	2024	5	3	0.01324
844	28	1	2024	5	3	0.00000
845	28	1	2024	5	3	0.02332
846	28	1	2024	5	4	0.00000

Figure 37: Example of output after basic cleaning

Data cleaning requires users to know what the CSV contains and for users to determine what data to retain or remove. After a few iterations of the cleaning, an example of the output can be seen in Figure 37. The column “created_at” has been split into “day”, “month”, “year”, “hour” and “minute” and “field2” is retained. Other factors like missing data etc. has also been dropped.

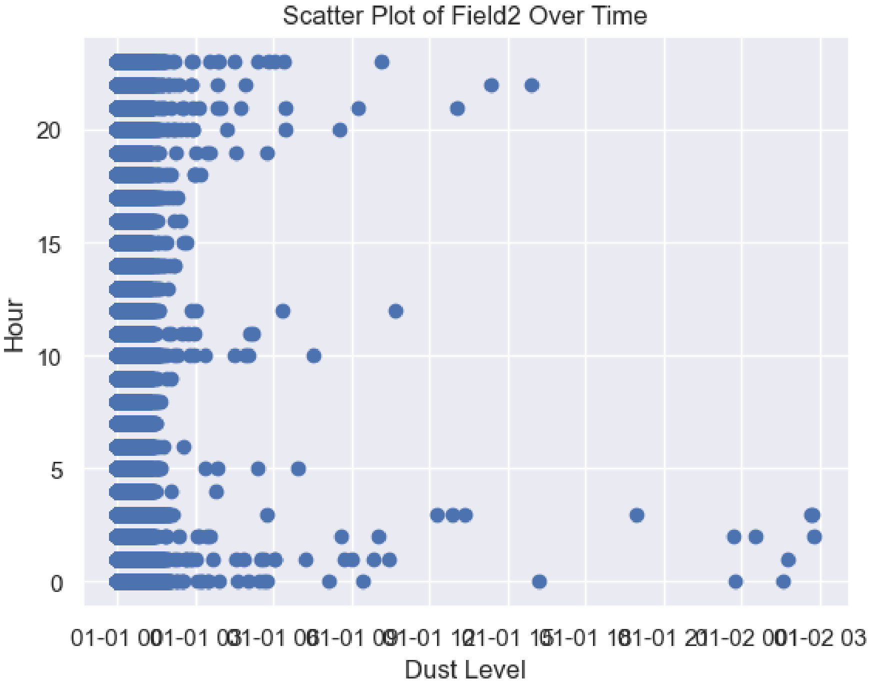


Figure 38: Scatter plot of cleaned data

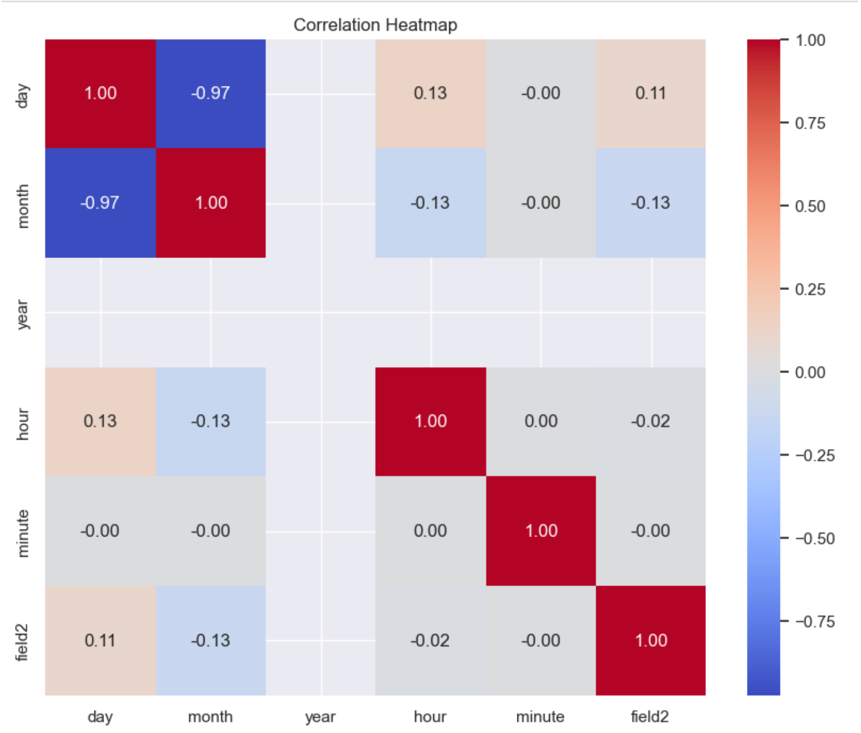


Figure 39: Correlation heatmap

Example of data visualization can be seen from Figure 38 and 39. As can been seen from the correlation heatmap, as of the data analyzation there seems to be no strong correlation between dust level and factors like date and time. As the sample size of the data is not large enough, no proper conclusions can be made as of now. Few solutions to better analyze will be to continue data logging a longer period or to add additional sensors that may factor into the dust level. Example of other sensors that may benefit the dust level analyzation would be temperature and humidity sensors. In conclusion, having a large sample data size and/or more sensors are important to better analyze the data. As mentioned earlier, data cleaning also has to be done to ensure that the analysis is done on a consistent basis and the conclusion of the analysis would not be misleading.

Chapter 5 Conclusion and Future Work

5.1 Conclusion

To meet the project's objectives, the Smart Dust Monitoring System with IoT Cloud was designed and developed. The two key features of the system are to remotely monitor the dust level through IoT Cloud and sending alerts. With the usage of IoT Cloud, the system not only provides continuous data collection but also has the ability to automatically alert users when the dust level has exceeded the healthy range. This also enables proactive measures to be taken to mitigate potential health risks that are associated with elevated dust levels.

The system was developed using the FireBeetle ESP32 micro-controller, Sharp GP2Y1010AU0F dust sensor and ThingSpeak as the IoT Cloud platform. As the popularity and growth of the usage of micro-controllers and IoT Cloud continues to develop, this system's scalability and adaptability makes it more flexible with detecting and dust levels in different locations or settings like industrial areas and housing estates.

5.2 Recommendation in Future Work

Future work on this project to further improve on the system can be to add additional sensors to the system. For example, including a temperature and humidity sensor could benefit this system as one of the main points is to alert users to the elevated dust level. So going on the same notion, users can be alerted to high temperatures or high humidity and proactive measures can be taken. Data from these two sensors will also be inserted into the IoT Cloud and can be considered when data analysis is being done. Also, with the growing interest and usage of Artificial Intelligence (AI), a machine learning model can be made and used using the multiple sensor data to reference from. Adding a sensor capable of detecting PM2.5 and PM10 particulate matter may also have a better correlation to the presence of dust level and be used when training a prediction model. Another consideration is to get a larger data sample size for better understanding and accuracy for conclusions to be made.

Reflection on Learning Outcome Attainment

As one would have expected, my journey through the FYP project was not a smooth journey. Early in the development stage, an obstacle appeared when trying to connect the micro-controller to the internet for the IoT purposes. Initially, the micro-controller chosen for the project was the Arduino Uno R3 but as that micro-controller has no in-built Wi-Fi capabilities, I had gotten an ESP-01 to work as the system's Wi-Fi module. Troubleshooting the issue resulted in failure thus, I had to make the decision to opt for another micro-controller that has in-built Wi-Fi capabilities to avoid such cases from happening again. Due to this, my project timeline has to be shifted to accommodate the procurement of newly selected component.

During the troubleshooting period, obstacles were also faced due to the large number of resources that are available online. Going through a large number of them trying to rectify the issue was a challenge as I had to pick and choose different solutions from different sources and to piece them together to try to solve the issues, that I was having with. Ultimately, I had to give up trying to troubleshoot it as it had already taken too much time and the decision to switch micro-controller was made.

Another challenge faced was integrating the micro-controller with the IoT platform. Due to this being my first experience dealing with said situation, software develop was quite the challenge. Although a library is available for use in Arduino IDE, there was still the learning journey needed to be made to implement functions and features that I have decided on.

Finally, throughout this FYP journey I had gained a better knowledge and skills on how to manage a project and how to handle obstacles when faced with one. Although there were challenging periods, I had enjoyed the process of overcoming them and the process of managing a project.

Acknowledging/Declaring the Use of GAI

Please refer to NTU's Current Policy & Guidelines on the Use of Generative AI available in NTUlearn home page and the link: https://entuedu.sharepoint.com/sites/Student/dept/ctlp/SitePages/Exploring-the-Impact-of-Generative-Artificial-Intelligence-(GAI)-Tools-on-Education.aspx

1. Complete the following declaration if applicable.
2. Create a Paper Trail to document the input prompt, output obtained, and how you have used it.

I Muhammad Ameerul Bin Azman (student name), AMEERUL001@e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission.

1. Name of course:
2. Course Code:
3. Instructor:
4. Title of Assignment/Project Submission:

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

i. Used GAI as permitted to assist in generating key ideas only □

ii. Used GAI as permitted to assist in generating a first text only. □
And/or

iii. Used GAI to refine syntax and grammar for correct language submission only. □
Or

iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project. ☑

I also declare that I have:

a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
c. Apart from the foregoing notices, the submission is wholly my own work.

Muhammad Ameerul Bin Azman

Student Name & Signature

01/04/2024

Date

References

[1] "How Dust Affects the World's Health." Accessed: Sep. 24, 2023. [Online]. Available: https://earthobservatory.nasa.gov/images/151100/how-dust-affects-the-worlds-health

[2] "Inhalable Particulate Matter and Health (PM2.5 and PM10) | California Air Resources Board." Accessed: Sep. 24, 2023. [Online]. Available: https://ww2.arb.ca.gov/resources/inhalable-particulate-matter-and-health

[3] Z. Wu, K. Qiu, and J. Zhang, "A Smart Microcontroller Architecture for the Internet of Things," *Sensors*, vol. 20, no. 7, Art. no. 7, Jan. 2020, doi: 10.3390/s20071821.

[4] Y. Güven, E. Coşgun, S. Kocaoğlu, H. Gezici, and E. Yilmazlar, "Understanding the Concept of Microcontroller Based Systems To Choose The Best Hardware For Applications," *Research Inventy: International Journal of Engineering And Science*, vol. 7, p. 38, Dec. 2017.

[5] "Ambient (outdoor) air pollution." Accessed: Jan. 05, 2024. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health

[6] D. Dee, "What is dust, and is it harmful to human health?," New Scientist. Accessed: Jan. 15, 2024. [Online]. Available: https://www.newscientist.com/lastword/mg24232351-500-what-is-dust-and-is-it-harmful-to-human-health/

[7] "1-hr PM2.5 Readings," Default. Accessed: Feb. 11, 2024. [Online]. Available: https://www.haze.gov.sg/resources/1-hr-pm2.5-readings

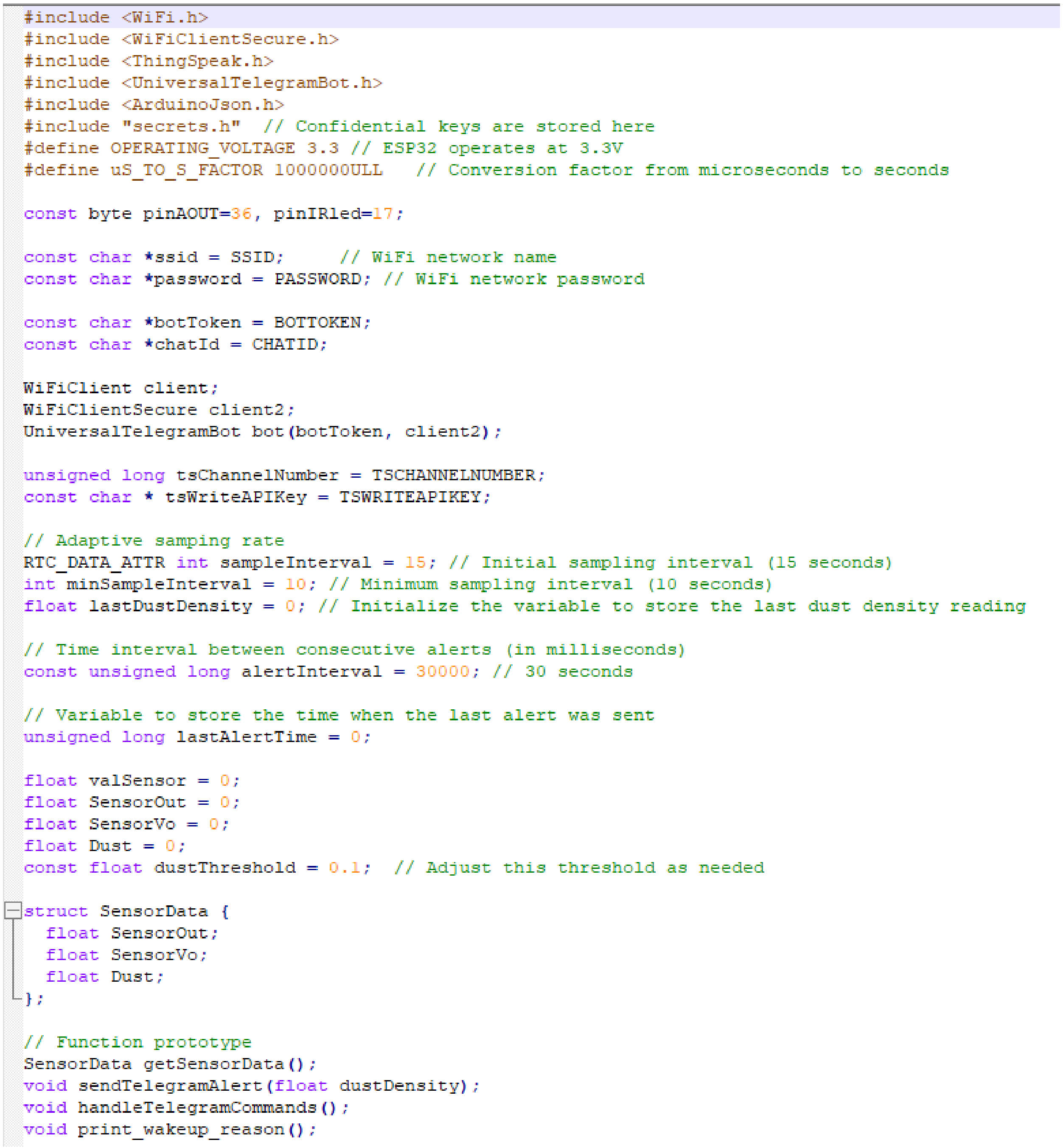
[8] J. A. K. Electronics, "Top microcontroller choices for your project." Accessed: Mar. 20, 2024. [Online]. Available: https://www.jakelectronics.com/blog/top-microcontroller-choices-for-your-project

[9] T. Agarwal, "Microcontrollers Types : Advantages, Disadvantages & Their Applications," ElProCus - Electronic Projects for Engineering Students. Accessed: Mar. 17, 2024. [Online]. Available: https://www.elprocus.com/microcontrollers-types-and-applications/

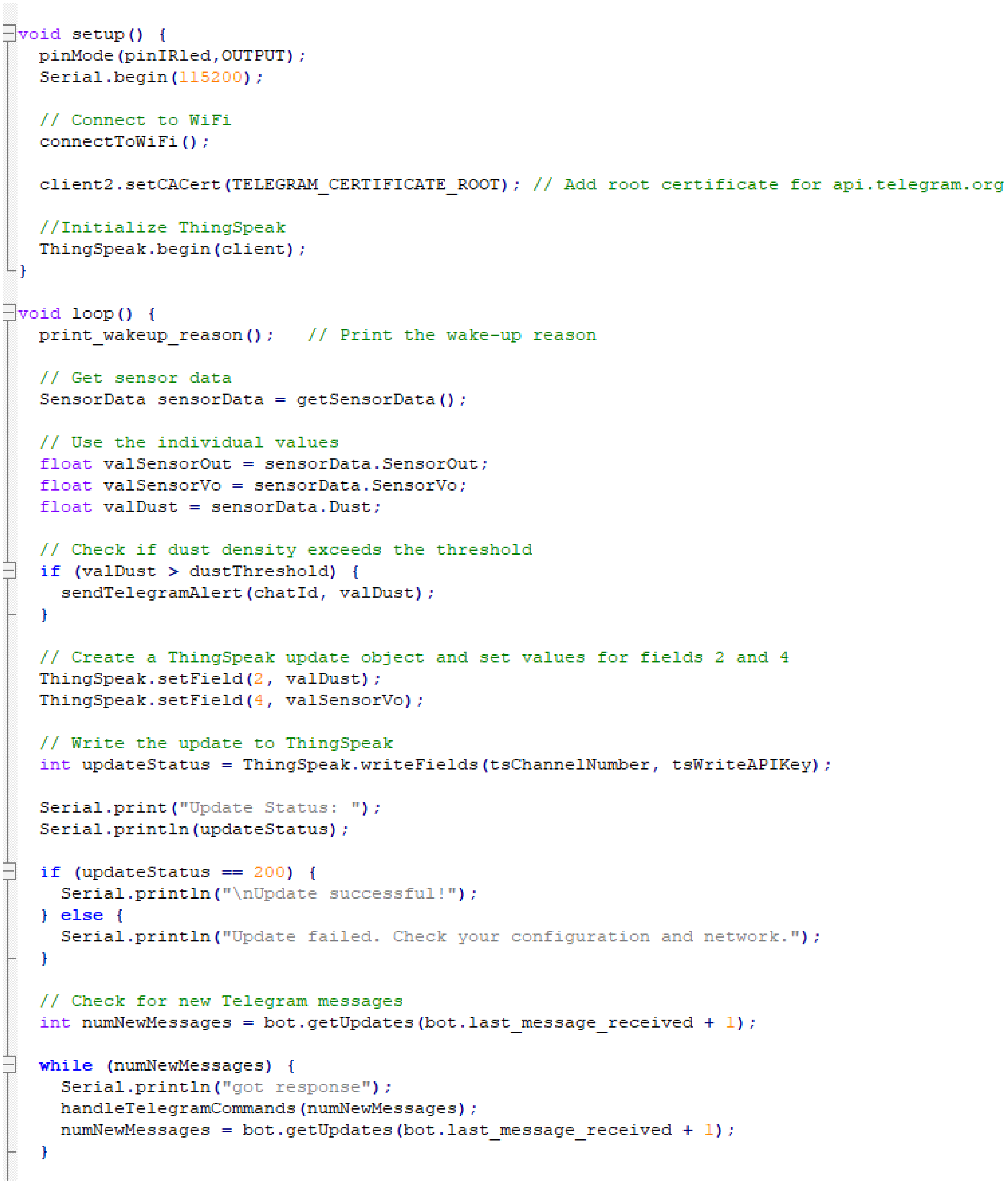
[10] A. Turner, "How Many Users Does Telegram Have? (Mar 2024)." Accessed: Mar. 26, 2024. [Online]. Available: https://www.bankmycell.com/blog/number-of-telegram-users/

Appendix

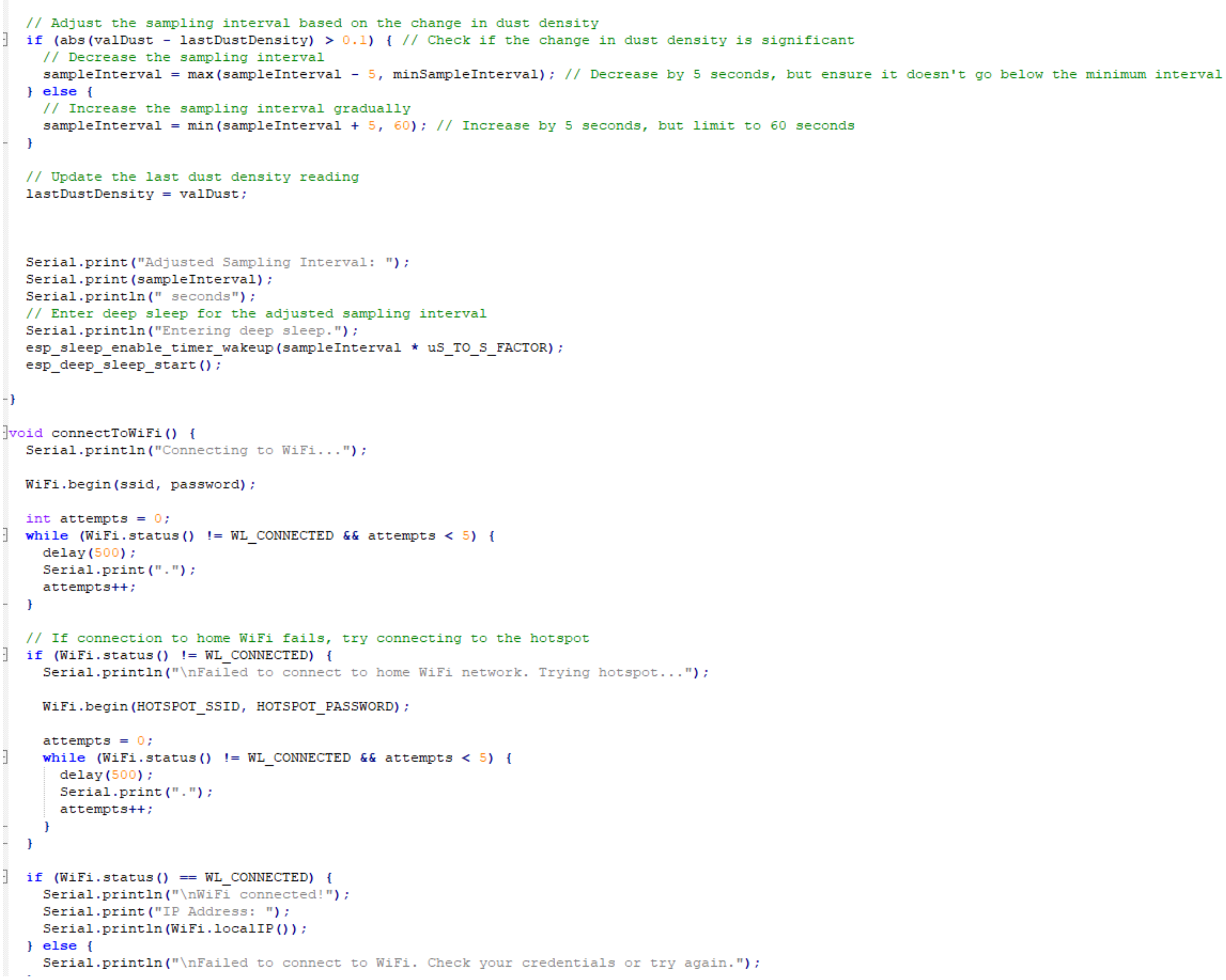
Arduino IDE code for Smart Dust Monitoring System with IoT Cloud



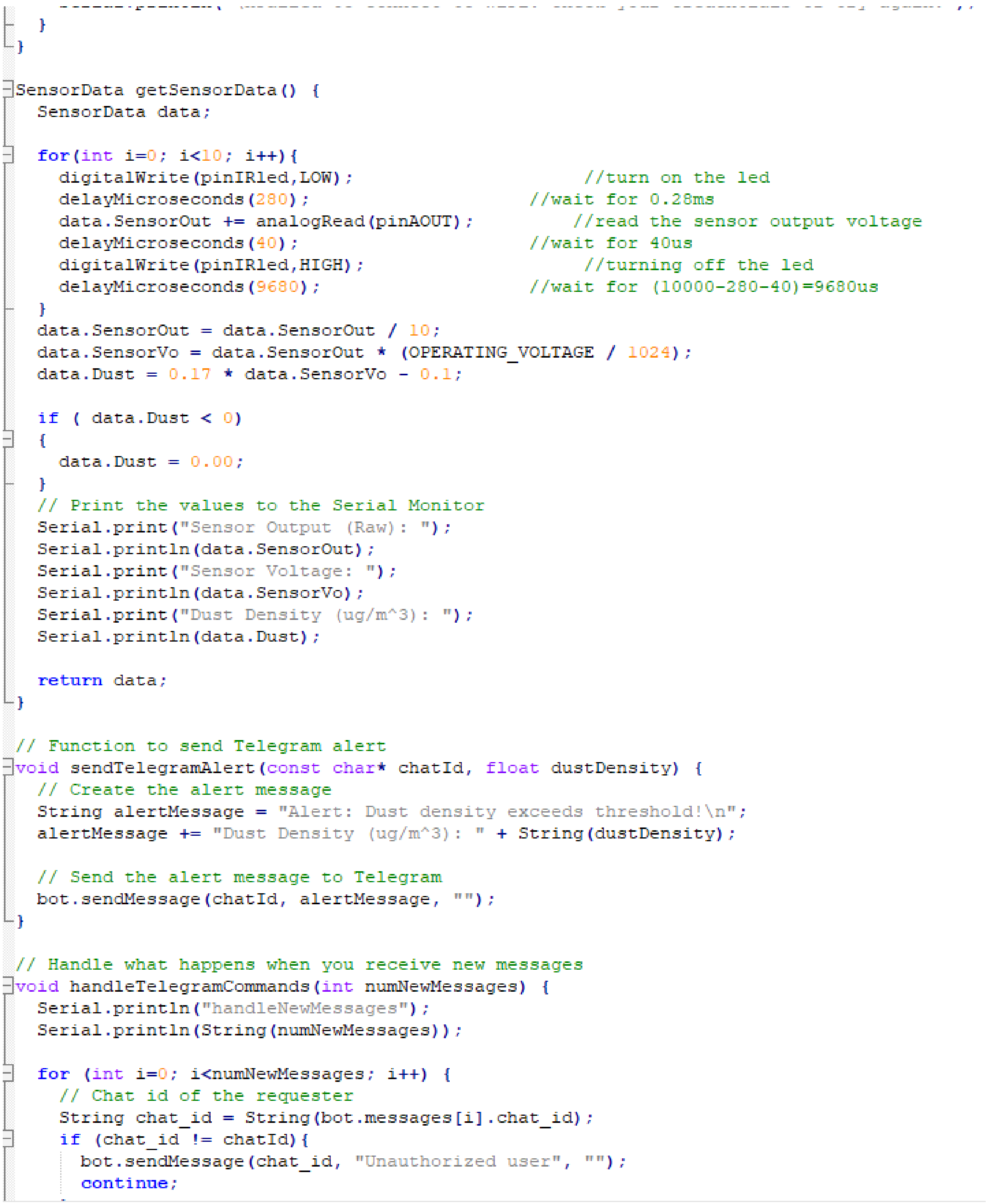
```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <ThingSpeak.h>
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>
#include "secrets.h"  // Confidential keys are stored here
#define OPERATING_VOLTAGE 3.3 // ESP32 operates at 3.3V
#define uS_TO_S_FACTOR 1000000ULL   // Conversion factor from microseconds to seconds

const byte pinAOUT=36, pinIRled=17;

const char *ssid = SSID;     // WiFi network name
const char *password = PASSWORD; // WiFi network password

const char *botToken = BOTTOKEN;
const char *chatId = CHATID;

WiFiClient client;
WiFiClientSecure client2;
UniversalTelegramBot bot(botToken, client2);

unsigned long tsChannelNumber = TSCHANNELNUMBER;
const char * tsWriteAPIKey = TSWRITEAPIKEY;

// Adaptive samping rate
RTC_DATA_ATTR int sampleInterval = 15; // Initial sampling interval (15 seconds)
int minSampleInterval = 10; // Minimum sampling interval (10 seconds)
float lastDustDensity = 0; // Initialize the variable to store the last dust density reading

// Time interval between consecutive alerts (in milliseconds)
const unsigned long alertInterval = 30000; // 30 seconds

// Variable to store the time when the last alert was sent
unsigned long lastAlertTime = 0;

float valSensor = 0;
float SensorOut = 0;
float SensorVo = 0;
float Dust = 0;
const float dustThreshold = 0.1;  // Adjust this threshold as needed

struct SensorData {
  float SensorOut;
  float SensorVo;
  float Dust;
};

// Function prototype
SensorData getSensorData();
void sendTelegramAlert(float dustDensity);
void handleTelegramCommands();
void print_wakeup_reason();
```



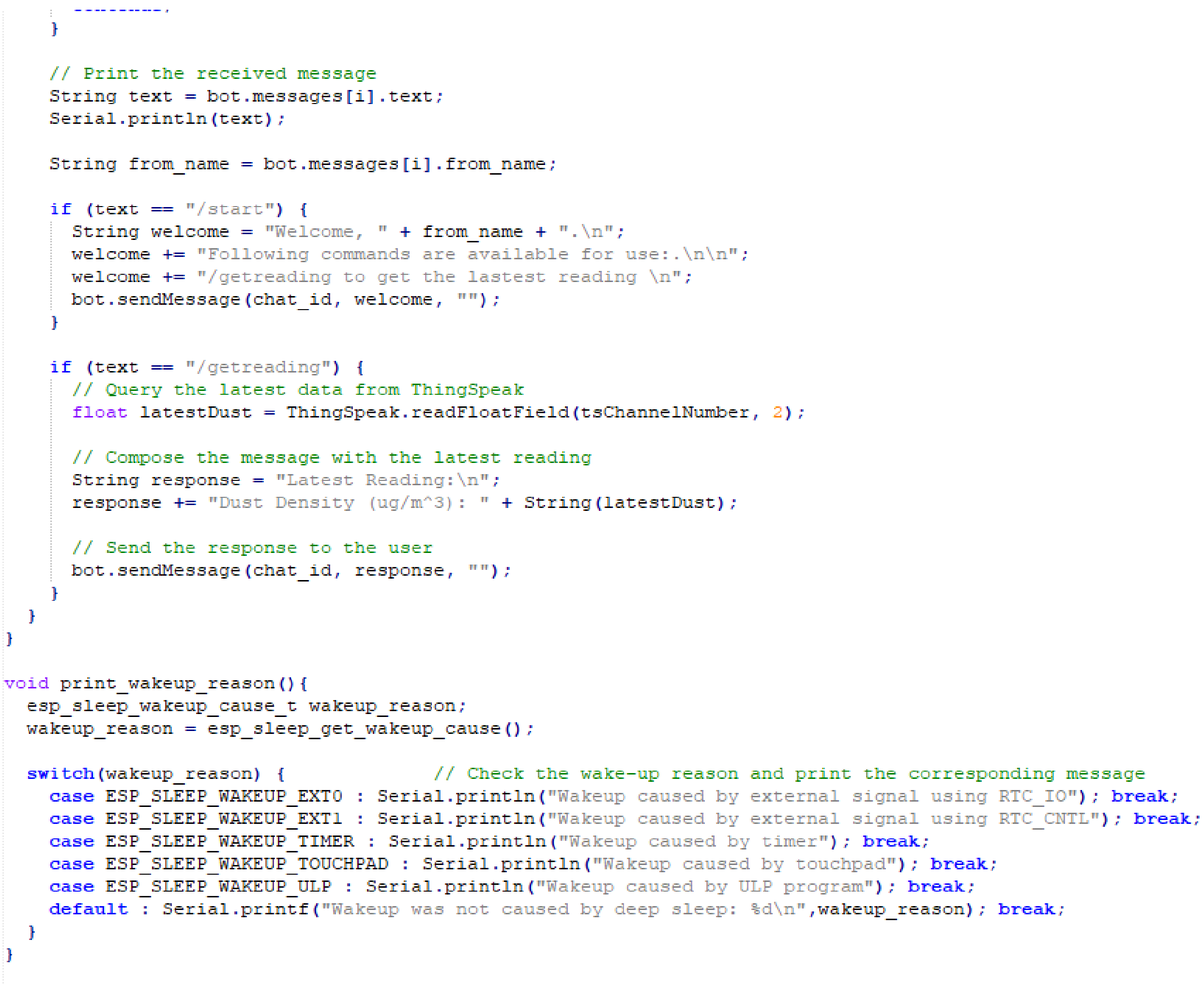
```
void setup() {
  pinMode(pinIRled,OUTPUT);
  Serial.begin(115200);

  // Connect to WiFi
  connectToWiFi();

  client2.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Add root certificate for api.telegram.org

  //Initialize ThingSpeak
  ThingSpeak.begin(client);
}

void loop() {
  print_wakeup_reason();   // Print the wake-up reason

  // Get sensor data
  SensorData sensorData = getSensorData();

  // Use the individual values
  float valSensorOut = sensorData.SensorOut;
  float valSensorVo = sensorData.SensorVo;
  float valDust = sensorData.Dust;

  // Check if dust density exceeds the threshold
  if (valDust > dustThreshold) {
    sendTelegramAlert(chatId, valDust);
  }

  // Create a ThingSpeak update object and set values for fields 2 and 4
  ThingSpeak.setField(2, valDust);
  ThingSpeak.setField(4, valSensorVo);

  // Write the update to ThingSpeak
  int updateStatus = ThingSpeak.writeFields(tsChannelNumber, tsWriteAPIKey);

  Serial.print("Update Status: ");
  Serial.println(updateStatus);

  if (updateStatus == 200) {
    Serial.println("\nUpdate successful!");
  } else {
    Serial.println("Update failed. Check your configuration and network.");
  }

  // Check for new Telegram messages
  int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

  while (numNewMessages) {
    Serial.println("got response");
    handleTelegramCommands(numNewMessages);
    numNewMessages = bot.getUpdates(bot.last_message_received + 1);
  }
```



```
  // Adjust the sampling interval based on the change in dust density
  if (abs(valDust - lastDustDensity) > 0.1) { // Check if the change in dust density is significant
    // Decrease the sampling interval
    sampleInterval = max(sampleInterval - 5, minSampleInterval); // Decrease by 5 seconds, but ensure it doesn't go below the minimum interval
  } else {
    // Increase the sampling interval gradually
    sampleInterval = min(sampleInterval + 5, 60); // Increase by 5 seconds, but limit to 60 seconds
  }

  // Update the last dust density reading
  lastDustDensity = valDust;



  Serial.print("Adjusted Sampling Interval: ");
  Serial.print(sampleInterval);
  Serial.println(" seconds");
  // Enter deep sleep for the adjusted sampling interval
  Serial.println("Entering deep sleep.");
  esp_sleep_enable_timer_wakeup(sampleInterval * uS_TO_S_FACTOR);
  esp_deep_sleep_start();

}

void connectToWiFi() {
  Serial.println("Connecting to WiFi...");

  WiFi.begin(ssid, password);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 5) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  // If connection to home WiFi fails, try connecting to the hotspot
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("\nFailed to connect to home WiFi network. Trying hotspot...");

    WiFi.begin(HOTSPOT_SSID, HOTSPOT_PASSWORD);

    attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 5) {
      delay(500);
      Serial.print(".");
      attempts++;
    }
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected!");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nFailed to connect to WiFi. Check your credentials or try again.");
```



```
  }
}

SensorData getSensorData() {
  SensorData data;

  for(int i=0; i<10; i++){
    digitalWrite(pinIRled,LOW);                     //turn on the led
    delayMicroseconds(280);                    //wait for 0.28ms
    data.SensorOut += analogRead(pinAOUT);         //read the sensor output voltage
    delayMicroseconds(40);                     //wait for 40us
    digitalWrite(pinIRled,HIGH);                    //turning off the led
    delayMicroseconds(9680);                   //wait for (10000-280-40)=9680us
  }
  data.SensorOut = data.SensorOut / 10;
  data.SensorVo = data.SensorOut * (OPERATING_VOLTAGE / 1024);
  data.Dust = 0.17 * data.SensorVo - 0.1;

  if ( data.Dust < 0)
  {
    data.Dust = 0.00;
  }
  // Print the values to the Serial Monitor
  Serial.print("Sensor Output (Raw): ");
  Serial.println(data.SensorOut);
  Serial.print("Sensor Voltage: ");
  Serial.println(data.SensorVo);
  Serial.print("Dust Density (ug/m^3): ");
  Serial.println(data.Dust);

  return data;
}

// Function to send Telegram alert
void sendTelegramAlert(const char* chatId, float dustDensity) {
  // Create the alert message
  String alertMessage = "Alert: Dust density exceeds threshold!\n";
  alertMessage += "Dust Density (ug/m^3): " + String(dustDensity);

  // Send the alert message to Telegram
  bot.sendMessage(chatId, alertMessage, "");
}

// Handle what happens when you receive new messages
void handleTelegramCommands(int numNewMessages) {
  Serial.println("handleNewMessages");
  Serial.println(String(numNewMessages));

  for (int i=0; i<numNewMessages; i++) {
    // Chat id of the requester
    String chat_id = String(bot.messages[i].chat_id);
    if (chat_id != chatId){
      bot.sendMessage(chat_id, "Unauthorized user", "");
      continue;
```



```
    }

    // Print the received message
    String text = bot.messages[i].text;
    Serial.println(text);

    String from_name = bot.messages[i].from_name;

    if (text == "/start") {
      String welcome = "Welcome, " + from_name + ".\n";
      welcome += "Following commands are available for use:.\n\n";
      welcome += "/getreading to get the lastest reading \n";
      bot.sendMessage(chat_id, welcome, "");
    }

    if (text == "/getreading") {
      // Query the latest data from ThingSpeak
      float latestDust = ThingSpeak.readFloatField(tsChannelNumber, 2);

      // Compose the message with the latest reading
      String response = "Latest Reading:\n";
      response += "Dust Density (ug/m^3): " + String(latestDust);

      // Send the response to the user
      bot.sendMessage(chat_id, response, "");
    }
  }
}

void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason) {             // Check the wake-up reason and print the corresponding message
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason); break;
  }
}
```