grEEEn: An IoT Platform for Indoor Plant Monitoring

Undergraduate Student Project

by

John Danielle T. Castor
[redacted]
BS Computer Engineering

Airick Miguel R. Gonzales
[redacted]
BS Electronics Engineering

Zylm M. Sabater
[redacted]
BS Electronics Engineering

Adviser:

Ramon Florentino Santos

University of the Philippines, Diliman

July 2023

Abstract

grEEEn: An IoT Platform for Indoor Plant Monitoring

The Internet of Things (IoT) is a network of interconnected objects through sensors and the Internet, enabling them to communicate. In this project, an IoT-based indoor plant monitoring system with virtual actuators was developed to monitor environmental parameters related to Golden Pothos health as well as indoor climate quality, via the STM32F411RE microcontroller. The readings from AM2320, BH1750 and SGP30 sensors were collected to the Thingspeak database and displayed to a web interface that also shows implications from the readings and the current actuator states. Furthermore, the room environment was also measured with and without Golden Pothos, and it was found out that the plant reduced the room temperature but had no significant effects to other factors. However, the project has limited generalizability due to limited resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 IoT Systems and its applications

The Internet of Things (IoT) refers to a conceptual framework where physical objects are interconnected through sensors and the Internet, enabling them to communicate and exchange data. While the notion of connecting objects to the Internet has existed for some time, recent technological advancements have facilitated the cost-effective and widespread integration of a greater number of devices. As a result, the IoT has gained significant traction and recognition. In essence, the IoT empowers various objects, such as appliances, sensors, and devices, to become ”smart” entities capable of seamless interaction with humans and among themselves, thereby enhancing efficiency and convenience in our daily lives.

In their report titled ”Unlocking the Potential of the Internet of Things,” the McKinsley Global Institute discusses the wide range of possible applications for the IoT. Different organizations have developed their own classifications for IoT applications, such as ”Industrial IoT” for manufacturing and utilities, or categorizing them by device type like wearables and appliances. The potential for IoT use cases is vast and can encompass various aspects of our lives.

With the increasing number of Internet-connected devices, there will be a significant surge in generated Internet traffic. Cisco predicts that non-PC devices will account for nearly 70% of Internet traffic by 2019. Additionally, the number of Machine-to-Machine (M2M) connections in different IoT verticals is expected to rise from 24% in 2014 to 45% in 2019.

These trends may lead to a transformation in how ”on the Internet” is being perceived. Currently, the World Wide Web is synonymous with the Internet experience, but the growth of IoT could shift the focus towards

passive interactions with connected objects like car components, home appliances, and self-monitoring devices.

The IoT encompasses various applications across different domains. In the human domain, IoT involves devices attached to or implanted within the human body, such as wearables and ingestibles, which monitor and maintain human health and wellness, aid in disease management, promote fitness, and enhance productivity. Within homes, IoT includes systems for controlling and securing the space, providing residents with convenience and safety. Retail environments utilize IoT for self-checkout options, in-store offers, and optimizing inventory management. In offices, IoT is employed for energy management, security, and improving productivity, especially for remote workers. Factories and standardized production environments use IoT to improve operating efficiencies, optimize equipment usage, and manage inventory. IoT in vehicles enables condition-based maintenance, usage-based design, and pre-sales analytics to enhance their performance and functionality. Overall, IoT applications bring connectivity and data-driven solutions to various aspects of people's lives, enhancing efficiency, convenience, and safety in a wide range of settings [1].

The study aims to utilize the IoT to monitor plant health and environmental variables, as plant maintenance is deemed difficult and worrisome [2]. By using IoT technology, a system can be created that collects and analyzes data from sensors placed in an environment. These sensors will measure crucial variables such as temperature, humidity, light intensity, and indoor air quality. The collected data will be transmitted wirelessly to a cloud platform, where it will be processed and analyzed in real-time. Through this approach, valuable insights will be gained into the environmental variables and understand how the introduction of the plants will impact the surroundings.

[3] has conducted a similar study and proposed the development of an IoT-based micro-climate monitoring system integrated with an automated intelligence system. The system aims to monitor soil, water, and air conditions in horticulture cultivation using IoT devices and capture equipment. The collected data is stored in a database and integrated with the Indonesian weather agency.

The paper proposed an IoT sensor board which consists of various sensors, including CO_2, light, air humidity and temperature, soil moisture and temperature sensors, as well as a camera and a wireless local-area network (WLAN) module. The sensor board was designed to collect data from the sensors and transmit it wirelessly to the servers.

The paper highlights the successful implementation of the proposed sensor board. The collected data was to be used to train an intelligent system using machine learning algorithms to automate horticulture control.

Actuators, such as fans, air conditioning, automated water systems, fertilizers, and lighting systems, was to be implemented to control micro-climate factors based on the machine learning system's judgements.

1.2 Related Work

[4] examined the effects of indoor plants on attention capacity in a controlled laboratory setting, with 34 student participants randomly assigned to either an office setting with four indoor plants or the same setting without plants. Attention capacity was assessed three times using a read span test, and the results showed that participants in the plant condition improved their performance from one to two, while no improvements was observed in the no-plant condition.

Similarly, [5] conducted a study in a classroom. Several Swedish schools were reported to have uncomfortable indoor environment due to poor thermal management and ventilation, overcrowding, and other factors. The solution implemented was the integration of green indoor plants to classrooms. The study involved the analyses of changes in indoor environment and improvement on the well-being of students and teachers. From the study, it was found out that the air quality and temperature improved, and student fighting decreased after several months. However, due to other factors such as COVID-19, more studies are needed to improve the validity of the conclusion of the study.

According to [6], it can be confirmed that poor air quality can lead to negative effects on students' typical schoolwork, such as performance of simple learning tasks. It was also observed that poor classroom air quality increased absenteeism, which impacted proper learning. Moreover, the study showed that poorly ventilated classrooms led to students being more likely to be less attentive, where the effects are more noticeable for tasks that require more complex skills such as spatial working memory and verbal ability to recognize words and non-word data.

400ppm	Normal outdoor air
400-1,000ppm	Typical CO2 levels found indoors
1,000-2,000ppm	Common complaints of drowsiness or poor air quality
2,000-5,000ppm	Associated with headaches, fatigue, stagnant, stuffiness, poor concentration, loss of focus, increased heart rate, nausea
> 5,000ppm	Toxicity or oxygen deprivation may occur. This is the permissible exposure limit of the daily workplace exposure
> 40,000ppm	Immediately harmful due to oxygen deprivation

Figure 1.1: ASHRAE CO_2 standard

Additionally, it was concluded that CO_2 concentration should be kept at or below 900 ppm. For reference, Fig. 1.1 shows the CO_2 level standard provided by the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE). A typical indoor environment should have less than 1000 ppm and more than that, it will lead to some form of negative impact to people.

[2] investigated the air temperature and humidity, CO_2 concentration, and particulate matter level due to various indoor plants such as Spider plant, Golden Pothos, and Boston fern, in classrooms. It was concluded that there is ”no clear winner” due to lack of reliable data, however, the Golden Pothos had the best performance based on the parameters followed by the Boston fern, and both proved to be suitable for classrooms, while the Spider plants were replaced by peace lily due to its need for high maintenance.

Furthermore, [7] provides numerical data for Golden Pothos' reduction of CO_2 level in an indoor environment. The study was conducted using two different values of light intensity, 300 lx and 700 lx. The plant's total CO_2 reduction were 60.67 ppm and 101.33 ppm for 300 lx and 700 lx, respectively.

These findings highlight the potential of incorporating specific plant species, such as the Golden Pothos, in indoor settings to naturally regulate CO_2 levels. It is worth noting that the choice and maintenance of indoor plants play a vital role in ensuring their viability and effectiveness in air quality improvement. Proper care, such as regular watering and monitoring of light exposure is essential for sustaining healthy plant growth and maximizing their air purifying capabilities.

[8] states that the Golden Pothos exhibits specific requirements to thrive successfully. Adequate lighting

conditions play a pivotal role in the development of plants, with recommended light levels ranging between 25 ft-c to over 200 ft-c. However, it is important to note that direct sunlight should be avoided as it may cause leaf scorching or damage. Maintaining a balance between optimal lighting intensity and protecting the plant from excessive direct sunlight is crucial for its overall health and growth.

Temperature regulation is another essential aspect to consider for Golden Pothos cultivation, as indicated by [8]. During nighttime, maintaining a temperature of approximately 65°F is recommended, while increasing the temperature to around 75°during daytime hours is advantageous for the plant's vitality. Relative humidity levels also impact the well-being of the Golden Pothos, as highlighted by [8]. Maintaining a relative humidity range of 25% to 49% contributes to a favorable growth environment for the plant.

In terms of watering requirements, [8] suggests a specific approach for the Golden Pothos. The surface of the soil mix should be allowed to dry before re-watering the plant.

1.3 Project Objectives

The primary objective of this project was to design, develop, and evaluate a plant monitoring system that leveraged sensor and IoT technology to monitor key environmental variables. The system aimed to provide real-time visualizations and virtual actuators, simulating optimal environmental condition maintenance for plant health and well-being.

With this objective in mind, the specific project goals were as follows:

1. To design a reliable IoT monitoring system that utilizes sensors to monitor key environmental variables such as temperature, humidity, light intensity, CO_2 levels, and TVOC concentration.

2. To collect and analyze data from the assigned sensors, AM2320, BH1750, and SGP30.

3. To transmit data to the cloud for real-time processing and analysis, as well as to assess data accuracy and scheduling reliability.

4. To simulate automatic plant cultivation through virtual actuators and the established optimal ranges for plant well-being, in addition to the developed interface for data and state visualizations.

5. To determine whether the integration of indoor plants positively affects a classroom environment by comparing pretest and post-test readings.

1.4 Scope and Limitations

The scope of this study encompassed the design, development, and implementation of a plant maintenance system that utilized the sensors assigned to monitor environmental variables, which were temperature, humidity, light intensity, CO_2 levels, and TVOC concentration. The study aimed to investigate the effectiveness of the system in simulating automatic plant monitoring and cultivation through virtual actuators and web interface.

However, it is important to acknowledge the limitations of this study. While the system aimed to provide displays based on the monitored variables, the accuracy and reliability of these data may be subjected to the performance of the provided sensors and the algorithms used for analysis. The study focused on providing preliminary insights into monitoring mechanisms for environmental conditions but may require further refinement for practical application.

Furthermore, the study was conducted within an assigned indoor environment, which limits the generalizability of the findings to different indoor settings. The effectiveness of the system in notifying users to specific plant needs or adjusting environmental conditions may vary depending on factors such as plant species, size, and individual plant requirements, which was not be fully captured in this study since only the Golden Pothos species was utilized. Additionally, it should be noted that the study was focused solely on the environmental aspects of plant maintenance and did not include the evaluation or monitoring of soil quality and nutrition as well as water level tracking. The study assumed the presence of suitable soil conditions for the plant and did not address any potential soil-related issues. Only water scheduling and suggestions were utilized and not real-time observations.

Moreover, the study duration was limited, and long term effects or seasonal variations on the effectiveness of the system was not be fully observed. Lastly, the project focused on indoor environment impacts, and improvements on human well-being was not covered.

Chapter 2

Methodology

2.1 Overview

The given task was to utilize the given three sensors, specifically the AM2320, BH1750, and SGP30 sensors, which respectively are temperature and humidity, light, and air quality sensors, to create an IoT platform that can be set up in a University of the Philippines - Electrical and Electronics Engineering Institute (UP-EEEI) room. From this, the idea was to set up an IoT platform for indoor plant monitoring by also utilizing the STM32F411RE Microcontroller and ESP8266 WiFi Module, as the sensors are aligned with most of the plant cultivation requirements, namely light, temperature, humidity, water, nutrition, and soil quality. Since the setup involved indoor plants inside a classroom, UP-EEEI Room 321, improvements on indoor temperature, humidity, CO_2 and TVOC levels were measured. Additionally, it was decided to indicate watering frequency on the developed system interface as well as to use a loam, garden and vermicast soil mixture, and *Epipremnum aureum* (Golden Pothos) was the chosen plant variable due to its robustness and indoor environmental effects [5].

In general, the project involved the construction of individual daughterboards for each sensor via PC-201 Printed Circuit Board (PCB), sensor and Wi-Fi module firmware development via STM32CubeIDE 1.12.0, sensor calibrations, web interface development via HTML, CSS and JavaScript, individual end-to-end communication tests, one-day pretesting and one-day posttesting, and data analyses on Microsoft (MS) Excel 365 [2]. Primarily, the pretest served as the control experiment, where initial indoor climate was measured. Posttest involved the placing of the Golden Pothos and indoor environmental improvement measurements, to be used for assessing the efficacy and possible recommendations for the experiment.

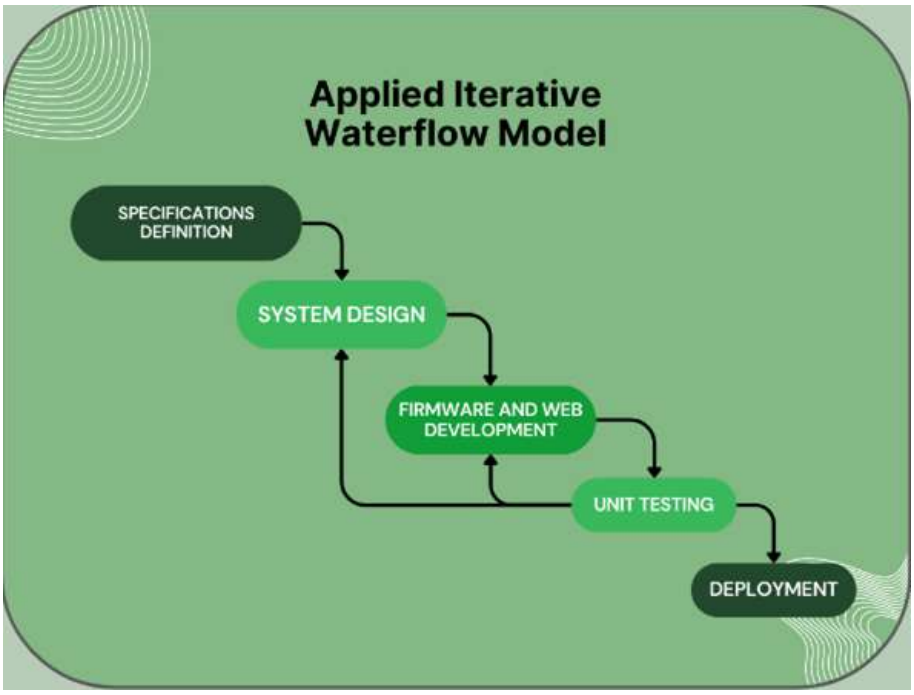


Figure 2.1: Applied iterative waterfall model

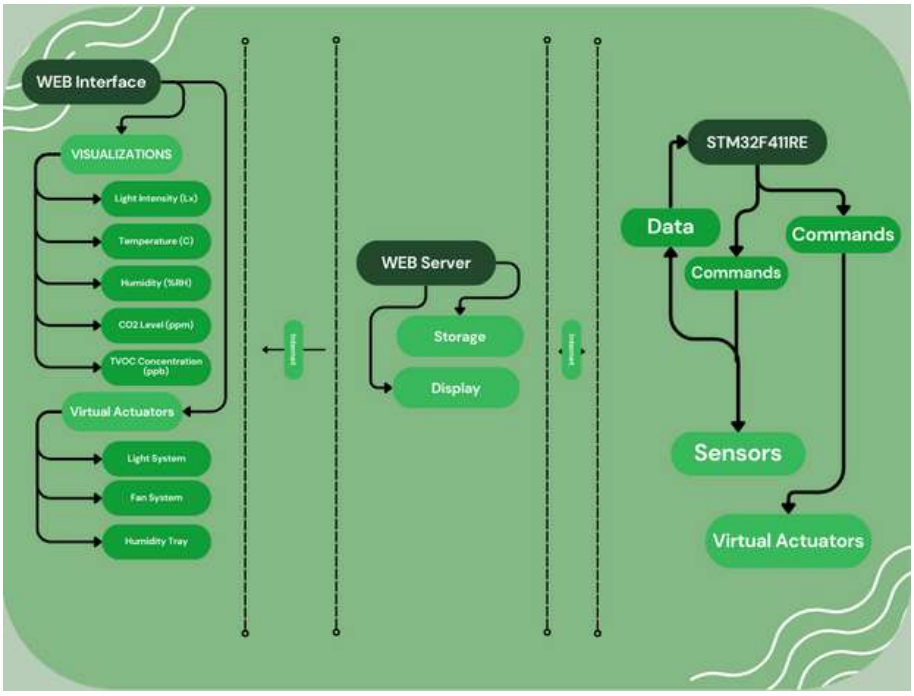


Figure 2.2: Overall software system design

2.2 Daughter board development

For all sensors, the development of daughterboards started with datasheet and schematic readings to understand the hardware requirements, connections, input and output data, and communication protocol. Paper layouts were designed to ease the soldering of components, shown in Appendix A. The soldered daughterboards were tested with respect to integrity and connectivity, via several drop and continuity tests, respectively. Prototyping was then done to verify if the modules are working, such as Serial Clock (SCL) waveform checking, as well as to confirm the pull-up resistor values and microcontroller pins to be used, via breadboard and resistors. Afterwards, the daughterboard solderings were finalized. Finally, the hardware used were powered via power banks and phone charger adapters that output $\sim$5V and $\sim$2A during the deployment.

All daughterboards utilized the same WiFi Module, ESP8266, a reliable and highly-integrated module which supports 2.4GHz connection and follows 802.11 b/g/n and Universal Asynchronous Receiver/Transmitter (UART) protocols, a serial communication protocol that requires receiver (RX) and transmitter (TX) wires [9][10]. It requires 3.3V power and 115.2kHz baud rate [9]. STM32F411RE controlled this module, which has six UART channels [11]. For the application, UART1 channel was used as well as pins PA10 and PA9 as the RX and TX pins, respectively. As for the connection, the VDD and chip enable pins of the ESP8266 modules were connected to the 3.3V pin of the microcontrollers, the RX and TX pins of the module were respectively connected to the TX and RX pins of the microcontrollers, and the modules were first connected to header pins then to the PCB.

2.2.1 AM2320

The AM2320 sensor based on datasheet reading, features stable, fast and calibrated technology for temperature and humidity sensing. It can sense temperatures and humidities, ranging from -40 °C to 80 °C and from 0%RH to 99.9%RH, with 0.1 °C and 0.1%RH resolutions, and less than five seconds response time. It requires 5V power and 4.7kΩ pull-up resistors, and uses Inter-Integrated Circuit (I2C) communication protocol, a bidirectional and open-drain serial bus protocol utilizing Serial Data (SDA) and SCL wires [11]. The sensor was controlled by STM32F411RE, which has three I2C channels that can run up to 50MHz [11]. The chosen channel was I2C1, and the chosen pins for SDA and SCL were PB9 and PB8, respectively. The sensor was connected to header pins connected to the PCB.

2.2.1.1

2.2.2 BH1750

The BH1750 light sensor based on the datasheet can detect from 0.001 to 100k lx. It requires a 3.3V power source and from testing it is determined that a 2.2kΩ pull-up resistors is best for its I2C protocol. Similar to the previous sensor the BH1750 sensor was also controlled by STM32F411RE using the I2C1 channel and the chosen pins are also PB9 and PB8 for the SDA and SCL pins respectively.

2.2.3 SGP30

The SGP30 sensor offers the capability to measure air quality parameters such as CO_2 and TVOC. According to [12], the sensor will measure the hydrogen (H2) gas and volatile organic compounds (VOCs) concentration and will use those measurements to calculate the values for the CO_2 and TVOC. It has the capability to measure the TVOC concentration within a range of 0 parts per billion (ppb) to 60000 ppb, as well as the CO_2 levels within a range of 400 ppm to 60000 ppm. It also provides the ability to read and write baseline values for baseline compensation. The sensor also offers raw signal measurements for part verification and testing purposes. The sensor communicates using I2C protocol and accepts a supply voltage between 1.62V and 1.98V which can be regulated by the microcontroller [12].The selected I2C channel was I2C1, and the designated pins for SDA and SCL were PB9 and PB8, respectively.

2.3 Sensor-MCU Integration

Integrating the sensors and MCUs required setting the STM32F411RE as master and the sensors as slaves [11]. We note that all sensors require I2C protocol, and that the software development was adapted from the repository of ControllersTech [13]. Firstly, the bare-metal project was created on STM32CubeIDE as well as the necessary inclusions such as the ”stm32f4xx.h” and ”math.h” modules were imported. The necessary General-Purpose Input/Output (GPIO) and I2C configurations were initialized by setting the appropriate register bits implied in the MCU reference manual preparing the pins, namely GPIOB, and hardware for I2C communication, The GPIO and I2C clocks and peripherals were enabled. The basic I2C functions were then implemented as instructed in the I2C Functional Description from the manual, such as I2C Start and Stop conditions, Address Transmission, and Master Transmission and Receiving [**empty citation**]. At this point the implementations differed for each sensor as they required different data processing and communication timings.

2.3.1 AM2320

The implementation for the AM2320 firmware was derived from the repository of Electrohobby [14]. The sensor utilizes 7-bit Addressing Mode, set at I2C initialization. The sensor was woken, as it uses passive mode, by sending a start signal then the slave address, 0xB8, then waiting for 1ms using HAL_Delay() function, then sending a stop signal, using the coded basic I2C functions. After, standard I2C communications can be used. To send the read command- the host must then start signal, then write the slave address 0xB8, the function code 0x03, the start address 0x00 and the register length 0x04, then a stop signal plus a 1ms delay. To read the sensor reading- the host must read the confirmation instruction by sending a start signal and the slave address + R, 0xB8+1, then reading the sensor data, then sending a stop bit.

The reading returned consists of eight bytes, the third and fourth bytes and the fifth and sixth bytes pertaining to the humidity and temperature readings in hexadecimal, respectively. The data were then converted to decimal and divided by 10 to obtain the %RH and °C readings [15]. The implementation involved containing the returned bytes to an 8-element buffer. The returned last two bytes are Cyclic Redundancy Check (CRC) checksum data, a protocol utilized by the sensor to segregate correct information and detect accidental changes, and the computation was programmed similar to the provided code in the datasheet. Host communication lasts for up to 3s, else the sensor becomes dormant requiring reawakening [15].

Virtual actuators were implemented to simulate how a fan and humidity system would respond from the obtained readings, via conditional statements and the time library of C. The ranges pertaining to Golden Pothos requirements were adapted from [8]. Along with the Read Data method, this set of code were looped infinitely while incorporating appropriate delays to continuously monitor the sensor readings and actuator states. The actuators virtualized were:

1. Desk fan that adaptively steadies or oscillates [16].
2. Ceiling fan that operates clockwise or counter-clockwise, to respectively circulate warm air or push cool air [17].
3. Pebble humidity tray that drains or pumps water when humidity is too high or low, respectively [18].
4. Baking Baking Soda Moisture Absorber Tray that closes or opens when humidity is too low or high, respectively [19].

In addition, the sensor was calibrated after the sensor code was completed, by simulating three scenarios - measuring a crowded room indoor environment, measuring a cloudy and cool outside environment, and spraying, fanning and directly blowing to the sensor, for approximately five minutes then analyzing whether

the readings were reasonable, as suggested by the instructor.

2.3.2 BH1750

The implementation for the BH1750 specific firmware was developed by determining the relevant op codes from its datasheet [20] which are the power on opcode (0x01), continously high resolution mode (0x10), and the slave address (0x23). The firmware first wakes the sensor by transmitting the slave address, wait for the acknowledge bit to set then sending the power on opcode. Then, to begin data collection the slave address is again transmitted and then the continuosly high resolution mode opcode after an acknowledge bit was receive. Lastly, after waiting for 200ms for the sensor to complete its data collection, the data is read by sending a start bit then the slave address increased by 1 bit (0x24) then after an acknowledge bit is received the high byte is read from the data register and an acknowledge bit was sent by the master. After that the low byte is read from the data register and an acknowledge bit and stop bit is finally sent. The read data from the sensor is then processed by applying the calibration equation to it created by comparing the raw data to a calibrated data from the light meter smart phone app. The read data is then passed through different conditions to determine the light level the sensor is currently reading either low light, medium light, or high light whcih occurs when the measured data is from 269-807 lx, 807-2153 lx, and 2153 lux and above respectively. The stm32 then sends the measure light intesity data and light condition data to the cloud. Then the stm32 waits for around 14 seconds before repeating the process from the second operation mentioned above.

2.3.3 SGP30

[12] shows a diagram of the structure for the measurement commands. The first eight bits hold the I2C address, in this case is the slave address, 0x58, followed by 16 bits, which may be a memory address or a command. Depending on whether it is write or read, the next data may differ. If it is write, the MCU will send 16 bits of data, partitioned to two parts, which makes it two 8-bit data, followed by a CRC. If it is read, the MCU will send the slave address, followed by the sensor sending a 16-bit data and CRC.

The integration of the sensor with the MCU involved an initial code obtained from [21] to assess the sensor's functionality. This code served as a starting point to verify if the sensor was operating as expected. However, the code relied on the usage of HAL, which was not it compliance with the project requirements. Hence, the code was discarded.

Despite discarding the code, its value as a reference for understanding the sensor's operation and integration

with the MCU was acknowledged. By utilizing the initial code as a reference, the final code was crafted to align with the project's specifications and adhere to the necessary guidelines, without relying on HAL.

The sensor makes use of the I2C functions from [13]. One of the most essential functions is the "SGP30_-Write" function which is responsible for writing a 16-bit command to the sensor. It starts by initiating I2C communication, followed by sending the slave address to the sensor. Next, it writes the upper 8-bits of the command followed by the lower 8-bits. Finally, the I2C communication is terminated.

Another important function is the "SGP30_Read" function which reads data from the sensor. Similar to the previous function, it starts by initiating I2C communication and waits for the Start Bit (SB) to be set, then sends the slave address. It reads the data in a loop by checking the Receive Data Register Not Empty (RxNE), stores the data in a buffer, and sets the ACK bit for all bytes except the last one. Finally, the ACK bit is cleared and I2C communication is stopped.

The "SGP30_Init" function is used to initialize the SGP30 sensor by calling SGP30_Write with the command for "sgp30_iaq_init" commmand, 0x2003 [12].

After initialization, an infinite for loop serves as the main code for the transmission of data. An "SGP30_-Measure" function is invoked to initialize air quality measurement. It uses the SGP30_Write function with the "sgp30_measure_iaq" command, 0x2008 [12].

Next, the SGP30_Read function is utilized to retreive the data from the sensor. To extract the values, the first and second bytes are combined for the CO_2 value while the fourth and fifth bytes are combined for the TVOC value. The third and sixth values are ignored since they only correspond to the CRC. The resulting CO_2 and TVOC values are assigned to the zeroth and first index of an array, which is to be send to the cloud, respectively.

A delay of 15 seconds is introduced before the loop iterates again, ensuring a regular interval between measurements and transmission.

The firmware were developed by applying Iterative Waterfall Model, a software method that involves planning, cyclic development and testing, then deployment, while verifying sensor readings by storing them to variables and monitoring them in the Live Expression window of STM32CubeIDE [22]. The sensor codes were then installed to the MCUs accordingly via the Debugging feature of STM32CubeIDE, and this feature was also used to visualize the program flow as well as to check current register values via the SFRs window.

2.4 Wi-Fi Module-MCU Integration

The implementation for the ESP8266-STM32F411RE firmware was similar across all boards, and was adapted from the LED Control project of ControllersTech [23]. The implementation utilized the Hardware Abstraction Layer (HAL) library, and two external libraries published by ControllersTech, UARTRingBuffer.c and ESPDataLogger.c, as well as their headers, providing ease to the program development. Basically, UARTRingBuffer.c is a collection of UART methods and uses ring buffer structure that efficiently and reliably handles data useful if the buffer size is fixed. The ESPDataLogger.c library involves ESP8266 Initialization and Send methods implemented by utilizing the UARTRingBuffer.c module and AT commands.

With these, WiFi access was employed to the project firstly by setting up the .ioc files for easy GPIO, UART, and clock configurations. Maximum clock speeds were utilized. The stated libraries were imported and another buffer was initialized to store the data to be sent to Thingspeak. Inside the main file were functions HAL, clock, GPIO, I2C, UART and ESP8266 Initialization, that configures the ESP8266 to connect to the Internet given the WiFi Service Set Identifier (SSID) and password, preceding the infinite loop included for continuous data receivings, transmissions, and processings. Inside the infinite while loop were the Read Data method, the virtual actuators processing the data readings, the appending of the data readings and actuator states to the buffer, sending each buffer element to Thingspeak given the Application Programming Interface (API) key of the channel utilized and using the Transmission Control Protocol (TCP) of Thingspeak, then a 15s-delay as Thingspeak requires 15s update intervals, deemed acceptable as a parallel study even utilized one hour intervals [24]. Furthermore, an automatic reconnection method was added to the program to address WiFi disconnections and instabilities via SysTick_Handler() and HAL_GetTick() functions, every 30-60s. And similar to sensor-MCU integration, cyclic developments and testings as well as finished firmware deployment to the MCU were applied through the Debugging function of STM32CubeIDE.

Setting Up the Cloud Interface

Thingspeak is a cloud platform that features live data analytics and visualizations via MATLAB [25]. It was utilized in the project for its simplicity and flexibility with respect to setup and platform compatibility. The channel setup was relatively easy and it started with creating accounts, setting new channel credentials such as Name and Description, adding visualization and widgets, and publicizing the channel for the web interface. The Channel ID and API key was then obtained from the channel settings and API keys tabs, used by ESP8266 to send data and for the web interface development. One Thingspeak channel was created per daughterboard and the data were merged on the web interface. Furthermore, each channel used different sets of visualizations and widgets depending on the sensor readings and virtual actuators used.

2.5 Web Interface Design

The web interface, grEEEn, features simple indoor environment monitoring and indications, showing the current climate conditions and actuator states, and whether they match with the Golden Pothos requirements. A website was created instead of an installed software since they do not require installation consuming data storage, save data on the cloud accessible everywhere, and that the system was already connected to the Internet. The interface was developed and designed by writing HTML, CSS, and JavaScript syntaxes on Notepad. The visualizations and widgets from the Thingspeak channel were accessed via the <iframe>syntax provided, and the latest readings were obtained via the fetch() and data.feeds[0] functions. The Golden Pothos requirement list and an overview of the overall system were also added.

2.6 Testing and deployment

Before the actual deployment, the end-to-end communication of each daughterboard setup from the sensor to the web interface was individually tested. The readings were also verified and calibrated by subjecting the hardware to three different environments and scenarios each. From testing, the networks used were relatively stable fiber-optic internet connections. All setups were able to continuously run for more than 12 hours.

2.6.1 Pretesting

Pretesting was done to serve as the control setup, when the current environmental conditions of the chosen room for deployment, Room 321, were initially measured. From the available resources, the setup was constructed by merging armchairs to elevate the plants and hardware, boxes to level the plants and boards, tapes to secure the boxes and the boards in place, and an extension cord for powering the daughterboards. The sensors were placed beside the plants as close as possible in order to sense the nearby climate conditions. This setup ran for a day, without the plants, in order to capture day and night readings. Note that the BH1750 unit was not included in this setup due to the plant's lack of integration into the room.

2.6.2 Posttesting

Posttesting was done to determine whether integrating indoor plants to a classroom improved indoor air quality, temperature and humidity, and how much the improvements were. The setup was similar to the setup in pretest with the addition of two Golden Pothos as well as the BH1750 unit. The placement of the

boards were unchanged and the setup ran continuously for another day. After the testing, the data were exported from Thingspeak in Comma-Separated Values (CSV) format, and were analyzed statistically and graphed using t-test: two-samples assuming unequal variances with 0.05 level of significance, through MS Excel 365.

Chapter 3

Results and Discussion

3.1 Sensor-Wi-Fi Module-MCU Integration

The group conducted a preliminary test by running the firmware for the SGP30, to validate the code implementation functionality and reliability for establishing a seamless UART connection between the ESP8266 and Thingspeak. This test was deemed sufficient as the code was shared among all members, ensuring consistency and accuracy in the implementation process.

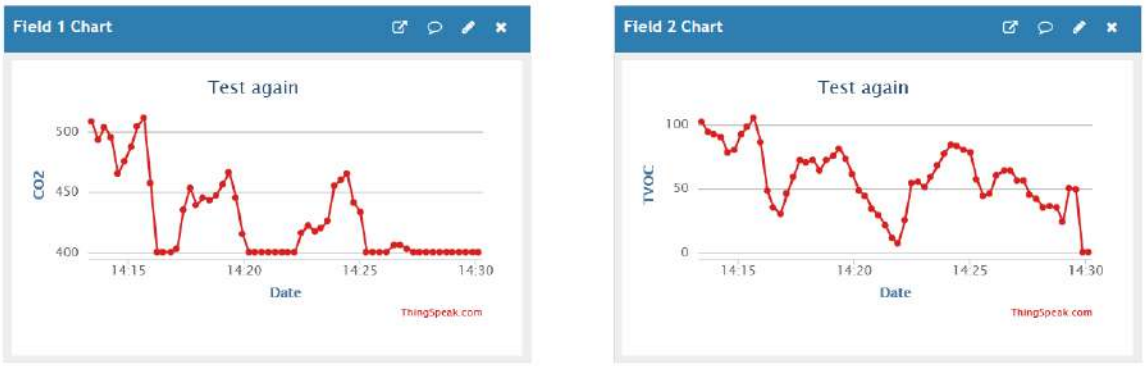


Figure 3.1: Wi-Fi module-MCU test

Fig.3.1 shows the results from the said preliminary test that used the SGP30. It is worth noting that the internet connection utilized during the test was established through a mobile data and phone hotspot. By conducting this initial test, the group aimed to assess the performance of the UART connection by comparing the data shown in the IDE's live expressions and the data shown in ThingSpeak's plot. Additionally, this allows the identification of any potential issues or bugs in the code, allowing for timely troubleshoot

and refinement. Ultimately, this approach of testing resulted in a more efficient development process and increased productivity.

Upon finalizing the firmware, deploying the program to the daughter boards, and confirming the UART communication, the sensor readings were verified and calibrated by subjecting the boards to three different settings each. The values were tracked via the visualizations on Thingspeak, and the plots were verified by comparing data via the Live Expressions window in STM32CubeIDE. Data can only be collected on Thingspeak and not on the Live Expression of the IDE thus the integration of the presentation of results for the Sensor, Wi-Fi Module, and MCU integration.

3.1.1 AM2320

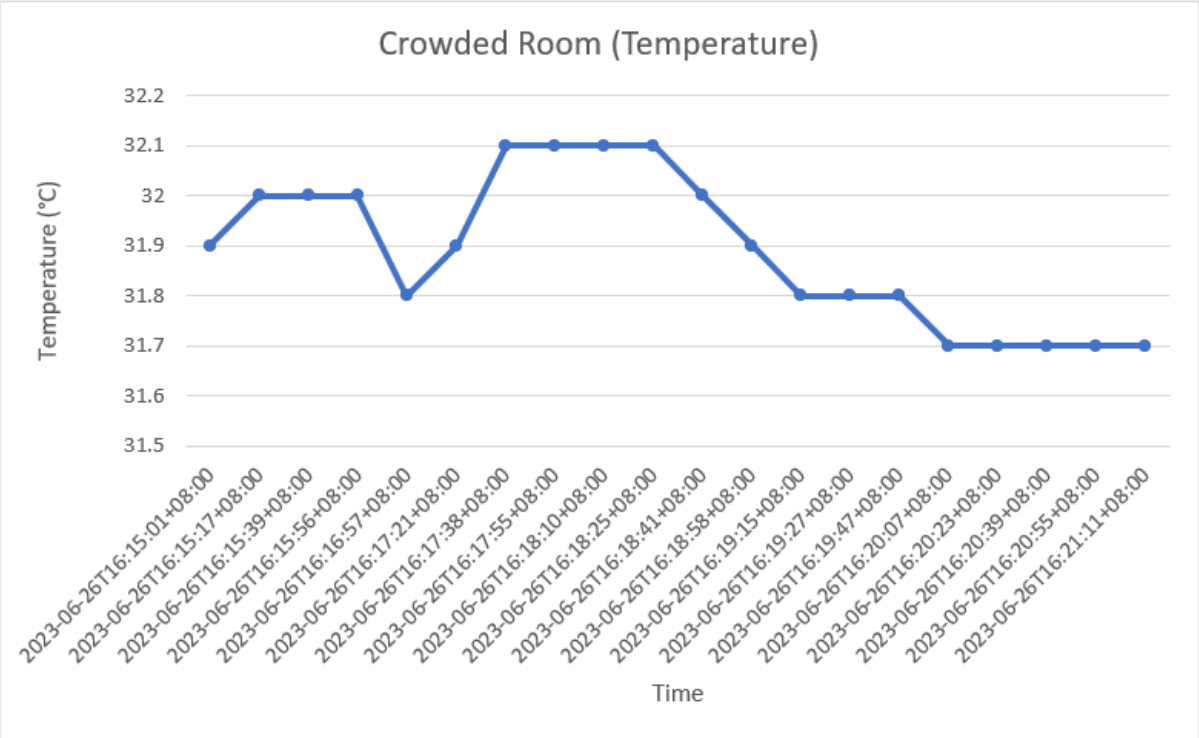


Figure 3.2: AM2320 temperature readings from the crowded room setting

Figure shows the temperatures obtained by AM2320 when subjected to a crowded room environment for five minutes on June 26, 2023, in degree Celcius (°C). The horizontal and vertical axes correspond to the data and time, and the temperature readings, respectively. Point intervals were roughly 15-20s, with an outlier between the fourth and fifth points on the crowded room measurements that reached a minute interval, due to Internet disconnection and reconnection. The obtained maximum value was 32.1°C at times

17:35 to 18:25, the minimum value was 31.7°C at times 20:07 to 21:11, and the mean was approximately 31.89°C.

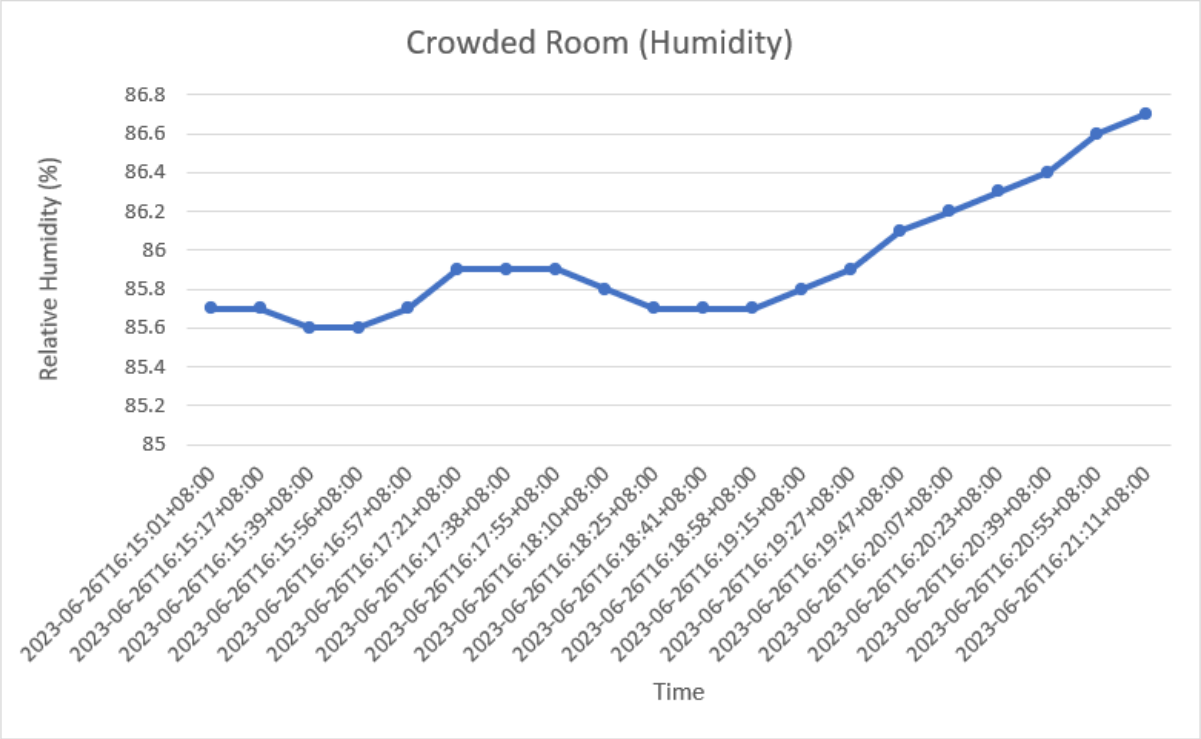


Figure 3.3: AM2320 humidity readings from the crowded room setting

-AM2320 humidity readings from the crowded room setting The figure above illustrates the humidity readings obtained from a crowded room setting for five minutes on June 26, 2023, in percent Relative Humidity (%RH). The horizontal axis pertains to the data and time, and the vertical axis corresponds to the humidity readings. The values peaked to 86.7%RH at the last point during 21:11, drops to 85.6%RH at 15:39 to 15:56, and the mean obtained was 85.945%RH.

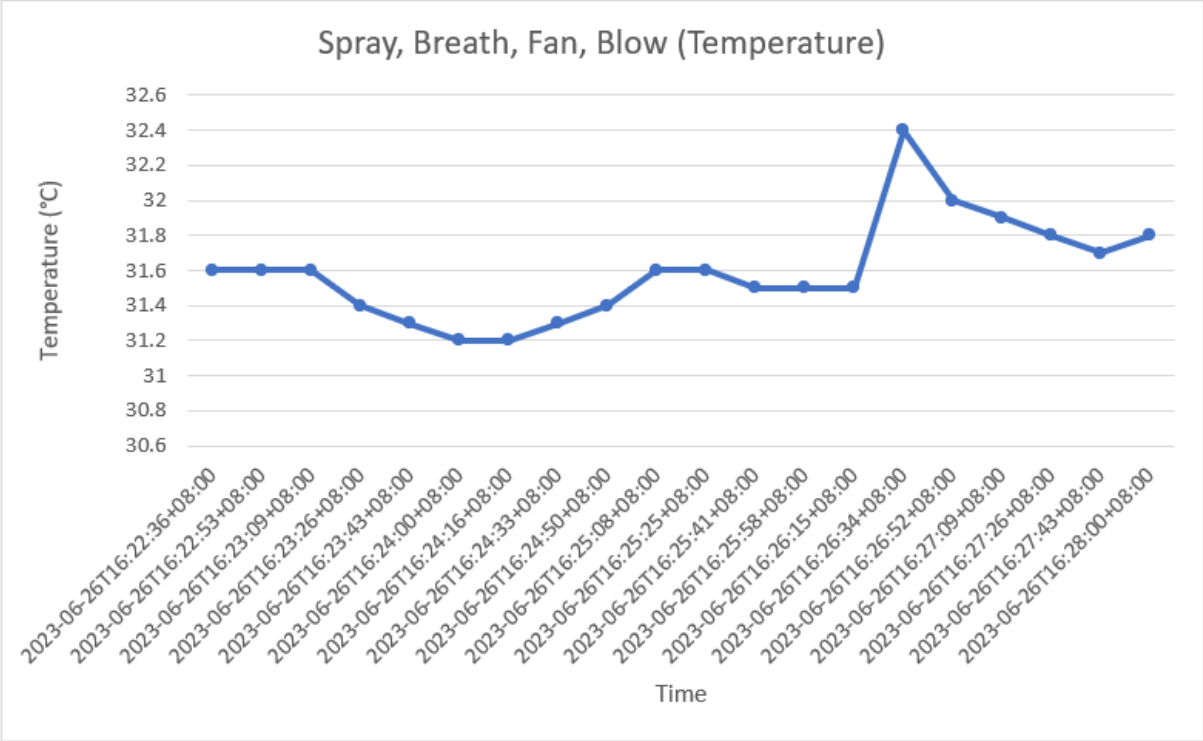


Figure 3.4: AM2320 temperature readings from the four experiments

-AM2320 temperature readings from the four experiments The second setting used involved four experiments, namely spraying alcohol, breathing, fanning, and very closely blowing to the sensor, done for a total of five minutes on June 26, 2023. As implied in Figure - regarding the temperature readings obtained from the said experiment, the highest, lowest, and mean values were respectively 32.4°C at time 26:34, 31.2°C at 24:00 to 24:16, and 31.595°C.

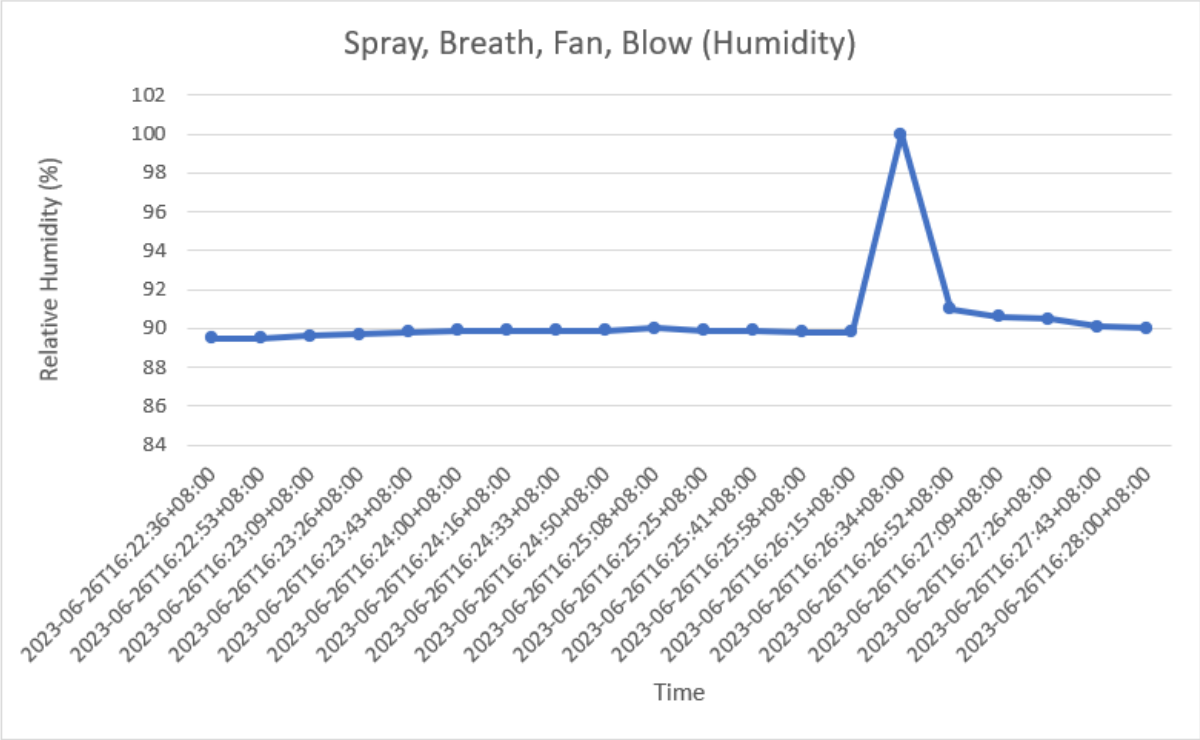


Figure 3.5: AM2320 humidity readings from the four experiments

-AM2320 humidity readings from the four experiments Shown in figure - are the humidity readings obtained from the second setting, in %RH. The x and y axes pertain to the date and time, and the AM2320 humidity readings in %RH. The peak was 100%RH at time 26:34, the trough was 89.5%RH at the first point on time 22:36, and the average was 90.465%RH.

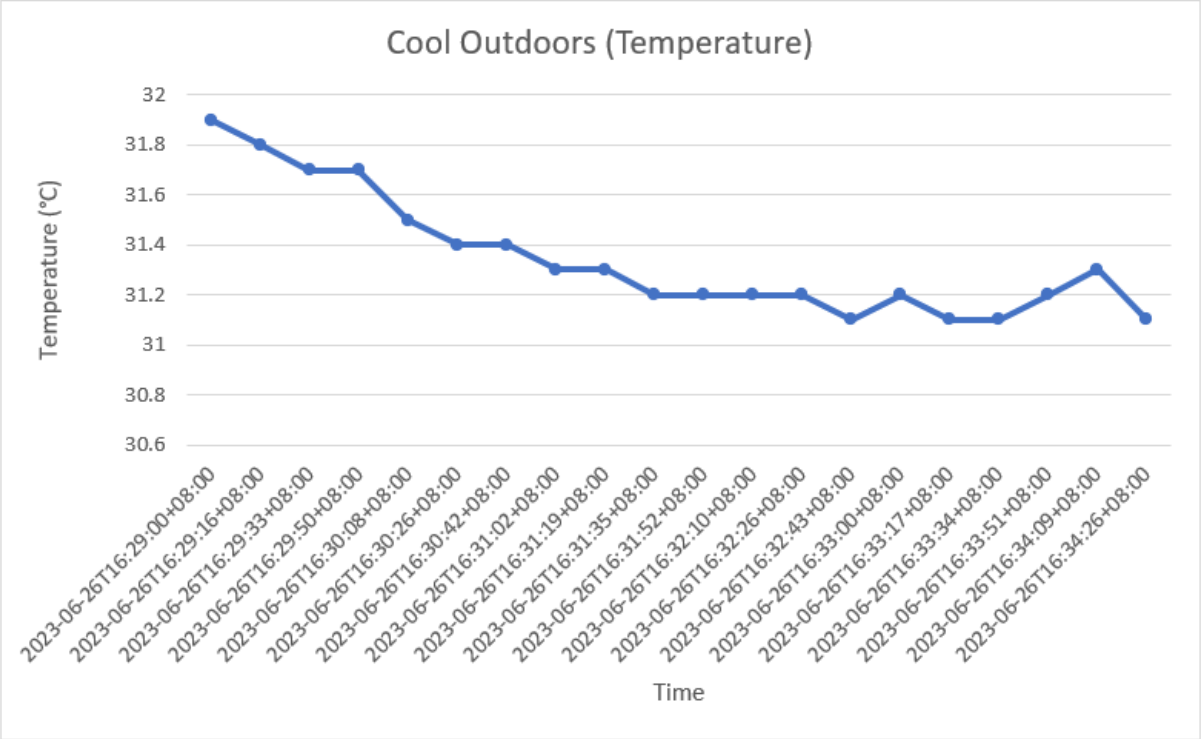


Figure 3.6: AM2320 temperature readings from the cool outside setting

-AM2320 temperature readings from the cool outside setting The last calibration was then done outdoors, on a cool and cloudy weather. Figure - illustrates the AM2320 temperature values obtained as it was subjected to the said setup. The horizontal and vertical axes were similar to the previous figures pertaining to temperature readings, as well as the date, duration, and point intervals. With these, the data peaked to 31.9°C at the first point on 29:00, the minimum value obtained was 31.1°C at times 32:43, 33:17-33:34 and 34:26 which approximately were the last points. The values averaged to approximately 31.345°C.

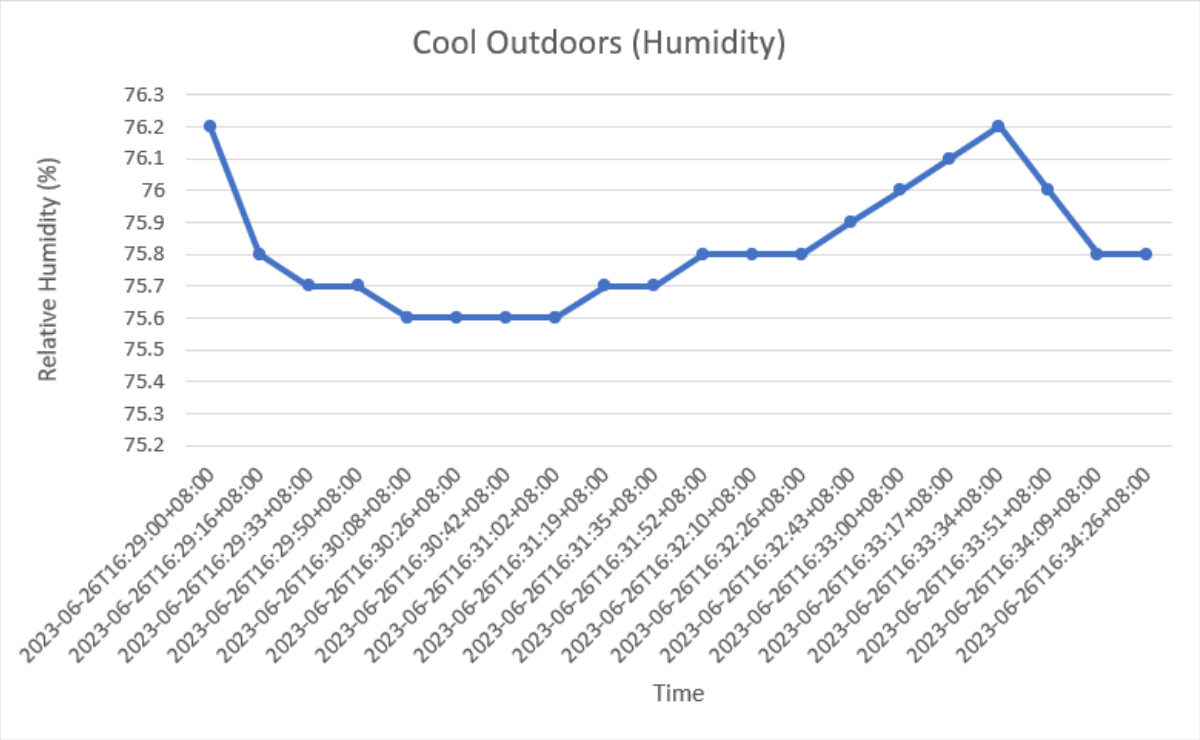


Figure 3.7: AM2320 humidity readings from the cool outside setting

-AM2320 humidity readings from the cool outside setting For the set of last calibration readings for AM2320, shown in figure - are the relative humidity data obtained. The maximum value was 76.2%RH at 29:00 which was the first datum. The minimum value was 75.6%RH at times 30:08 to 31:02 and the mean obtained was approximately 75.82%RH.

3.1.2 BH1750

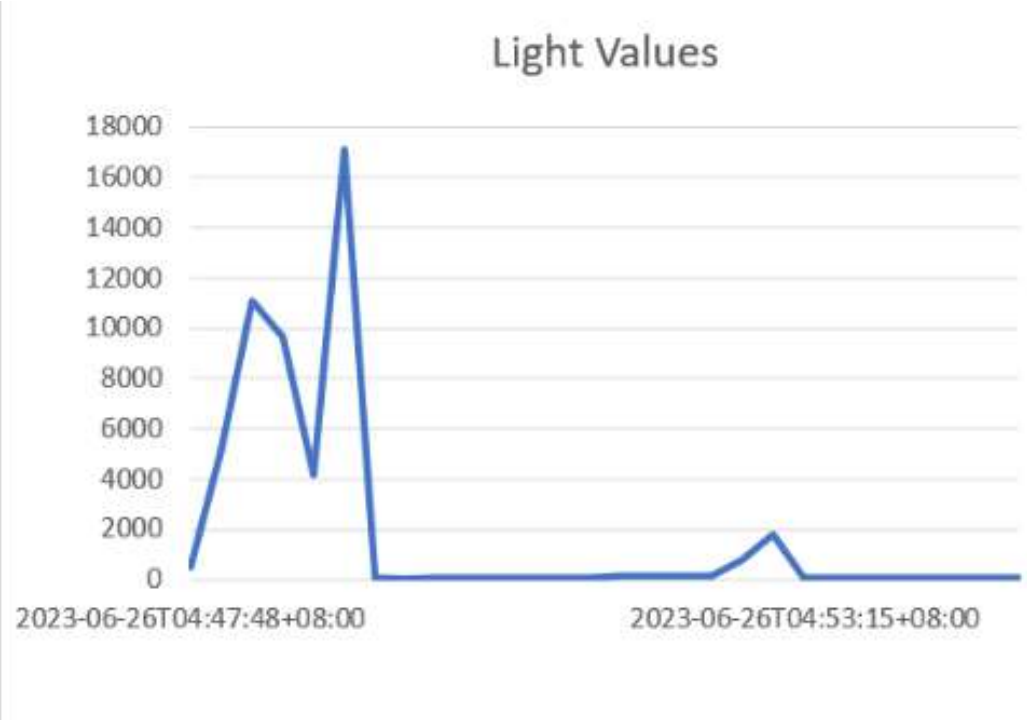


Figure 3.8: BH1750 calibration

Calibrating the BH1750 light sensor is done by exposing the diiferent light levels as seen in 3.8. The sensor is exposed to a flashlight causing the spikes shown in the plot. Then, the ambient room light is measured as well as the ambient room light but closer to the light bulb. The data is then compared to the output of lux meter and the calibration equation is derived from the assumption that the calibrated scale and the raw data is linear. Two points are sampled as shown in 3.9f and their slope is determined and was used to convert the raw data to a calibrated data.

raw	calibrated	slope
32	26	0.8
42	34	

Figure 3.9: BH1750 calibration (samples)

3.1.3 SGP30

Calibrating the SGP30 sensor was a crucial step in ensuring accurate measurements and reliable data analysis. There were three cases for the calibration of the sensor and only the CO_2 was observed as the TVOC

has a direct relationship with the CO_2, meaning if the CO_2 increased, the TVOC will also increase.

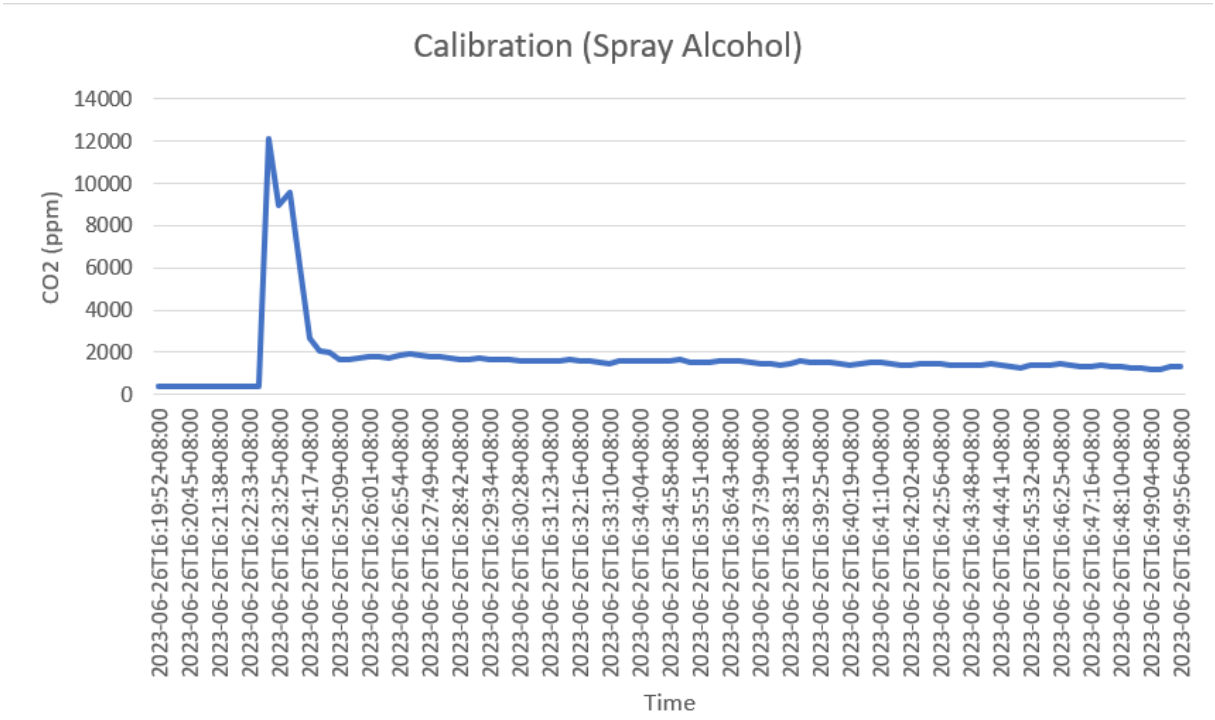


Figure 3.10: SGP30 calibration (spraying alcohol)

Shown in Fig. 3.10 is the first case for the calibration. It involved the controlled spraying of alcohol near the sensor. Prior to the event, the readings of the sensor were observed to hover around 400 ppm. However, upon spraying the alcohol, an instantaneous spike in readings to 12000 ppm was observed, highlighting the sensitivity and responsiveness of the sensor. As time progressed and the alcohol dispersed, the readings of the sensor gradually stabilized to a more reasonable and representative level of 2000 ppm.

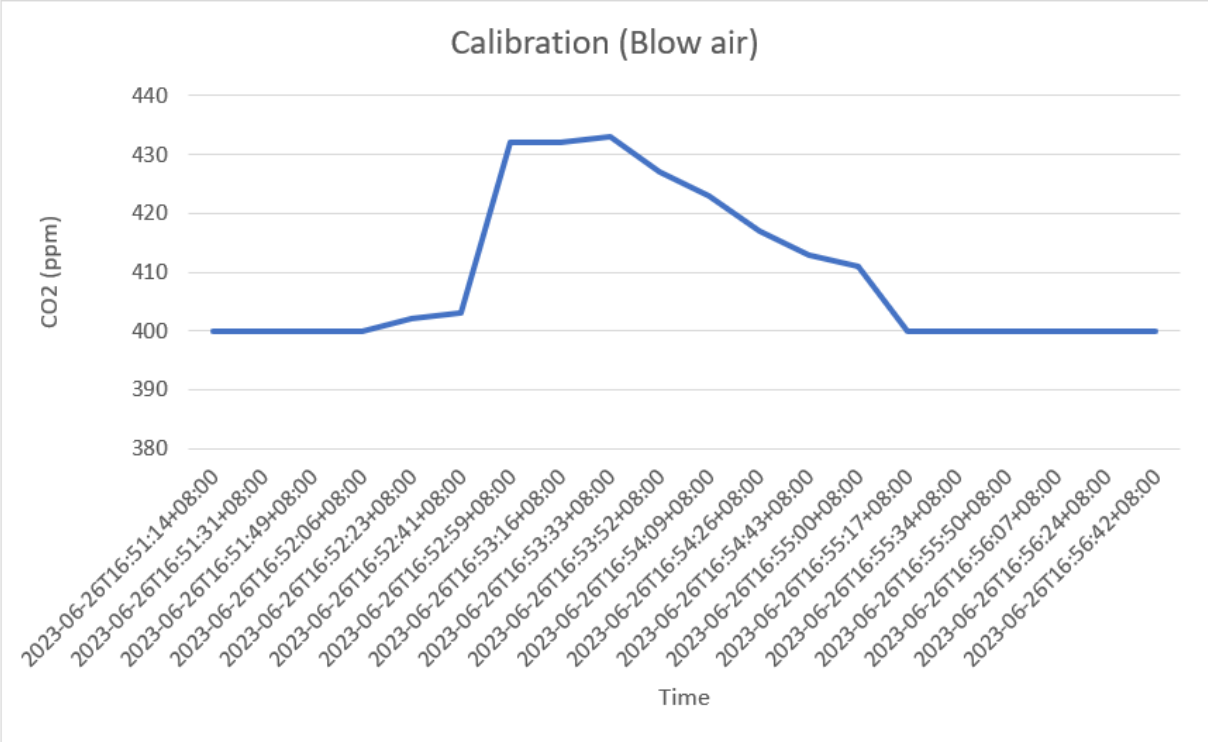


Figure 3.11: SGP30 calibration (blowing air)

Shown in Fig. 3.11 is the second case for the calibration. It involved blowing air from the mouth towards the sensor. During this calibration process, the maximum observed value reached around 430 ppm, with occasional variations ranging between 410 ppm and 430 ppm. However, as time progressed, a downward trend was observed, and the CO_2 levels gradually decreased to nearly the initial baseline level of 400 ppm. This highlights the importance of accounting for human respiration and its impact on CO_2 measurements.

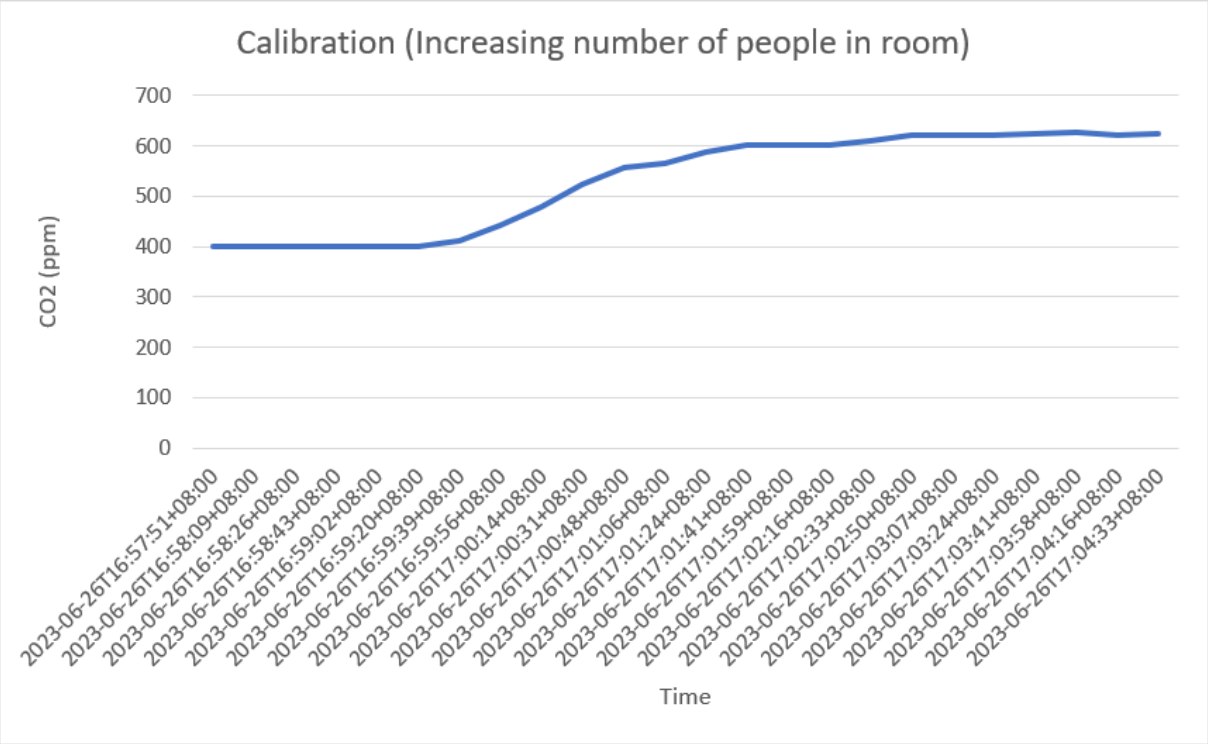


Figure 3.12: SGP30 calibration (increasing number of people in room)

The third and final case involved observing the impact of an increasing number of people in the room on the CO_2 levels detected by the sensor as shown in Fig. 3.12. It was observed that as the number of individuals in the room increased, the CO_2 levels also increased accordingly. The measured CO_2 value was observed to be around 600 ppm for around 10 people near the sensor. Notably, the CO_2 levels did not decrease as long as the number of people in the room remained constant.

This observation highlights the direct correlation between human occupancy and CO_2 levels, emphasizing the significance of proper ventilation in maintaining optimal air quality. These calibrations demonstrated the capability of the SGP30 to detect and respond to changes in the air composition.

3.2 Web Interface

To ensure that the developed web interface yields reliable and the correct data, as well as to assess scheduling accuracy, the web interface displays were compared with the Thingspeak database visualizations.

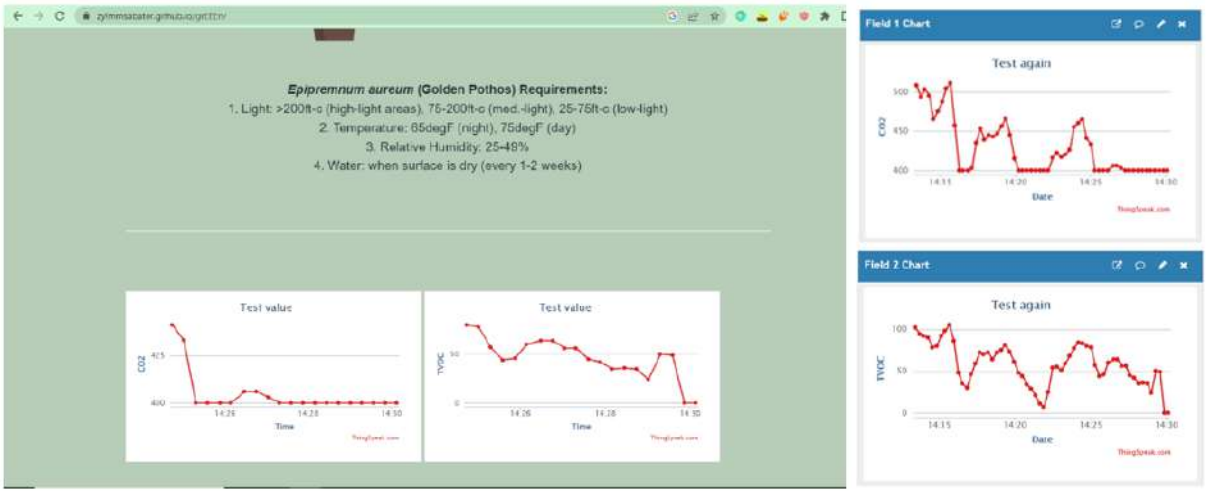


Figure 3.13: Initial web interface and ThingSpeak plots

Shown in Fig. 3.13 is the side-by-side initial grEEEn interface and Thingspeak display comparison. As shown, the plots obtained were exactly the same. Upon observations, the plots updated at about the same times for both grEEEn and Thingspeak, and the web interface even often updated a bit earlier. It should be noted that the interface was further developed with respect to the visuals and information provided after the deployment, based on the further useful applications the developers thought of after the experiments.

3.3 Testing and deployment

The overall system was then deployed for roughly two days, June 28-30, 2023, with one day each for pretest and posttest. Afterwards, the data in CSV format were exported from Thingspeak, then were plotted and analyzed using MS Excel 365.

3.3.1 Pretesting

3.3.1.1 AM2320

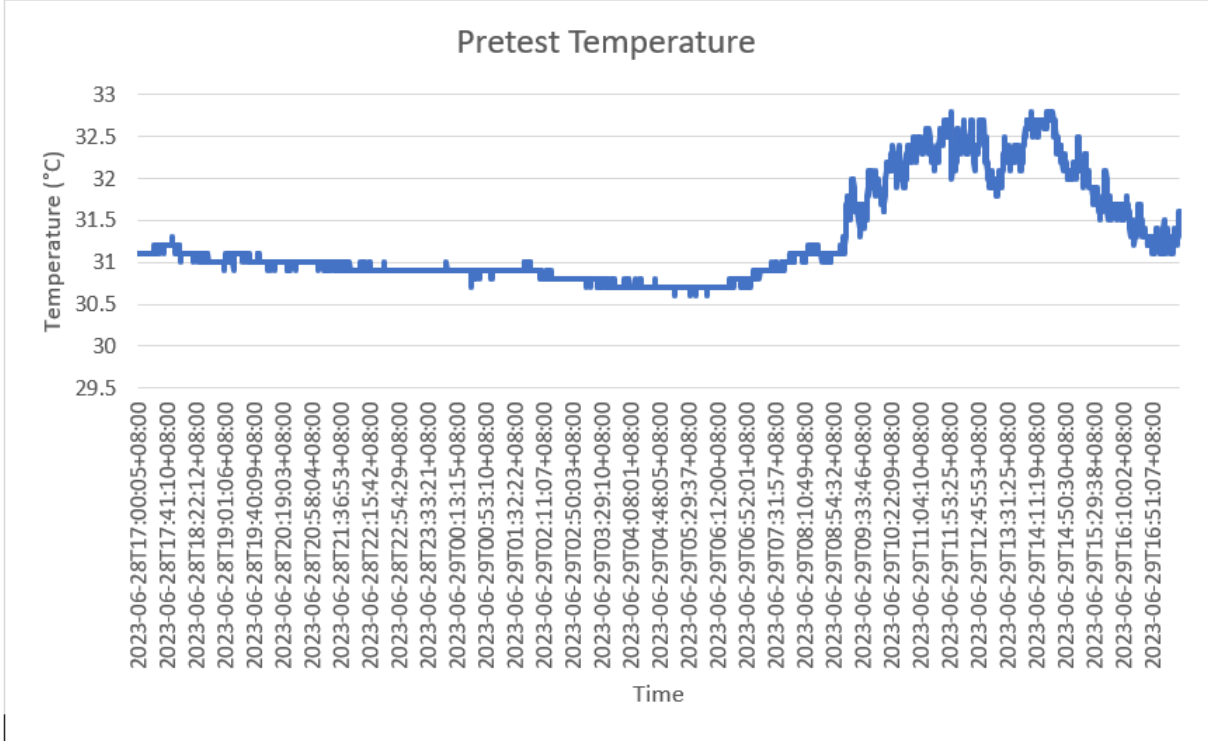


Figure 3.14: Pretest temperature

Shown in figure - are the temperature data obtained during the pretesting period. The abscissa and ordinate axes relate to the data and time as well as the temperature readings in °C, repectively. The point intervals were around 15s but there were longer intervals such as one minute whenever the AM2320 unit encountered internet disconnections. The maximum, minimum, mean and standard deviation values obtained were 32.8°C, 30.6°C, 31.28°C, and 0.35°C, respectively. At night, from 18:00 to 6:00, the mean and standard deviation were 30.89°C and 0.12°C while at day, the values were 31.71°C and 0.62°C.

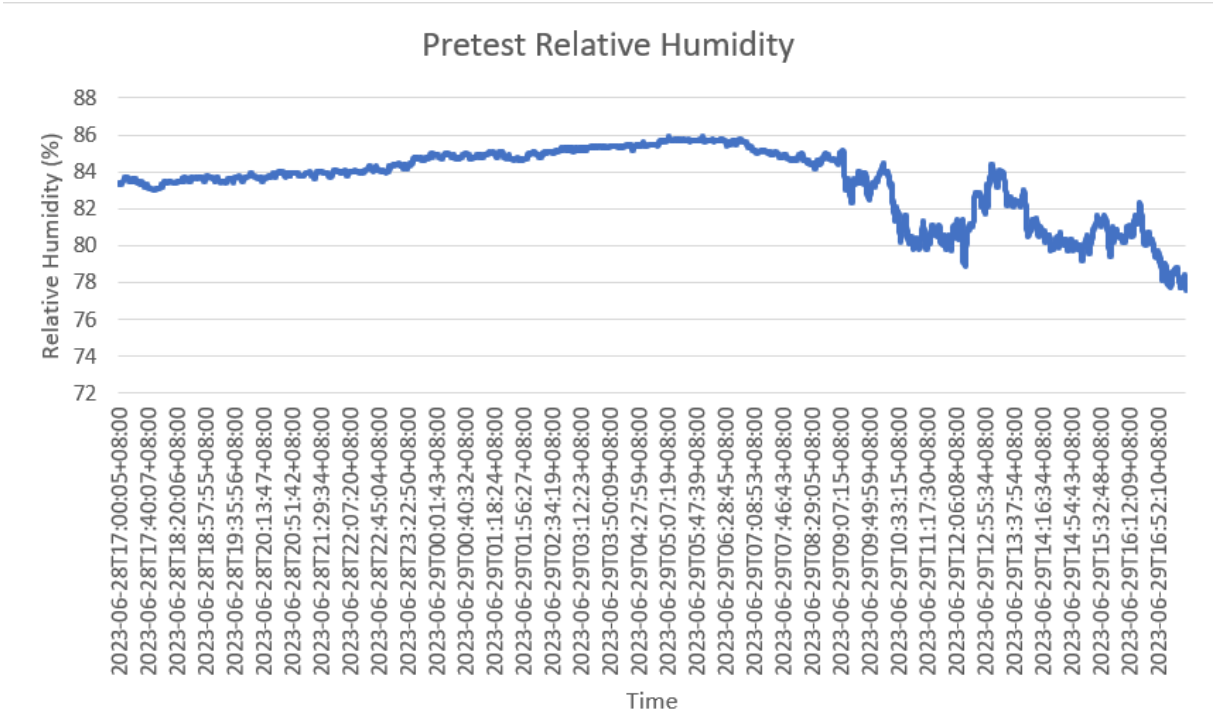


Figure 3.15: Pretest humidity

Shown in figure - are the AM2320 humidity readings during the posttest. The horizontal axis pertains to the corresponding date and time while the vertical pertains to the humidity measured in %RH. Similar point intervals were obtained as in the temperature measurements. The greatest measurement was 85.9%RH, drops up to 77.2%RH, and the data had a mean and standard deviation of 83.36%RH and 4.25%RH. The means and standard deviations were 84.54%RH and 0.73%RH at night, and 82.10%RH and 2.35%RH at day.

3.3.1.2 SGP30

This pretest served as a baseline measurement to establish the typical air quality conditions in an environment without plant influence.

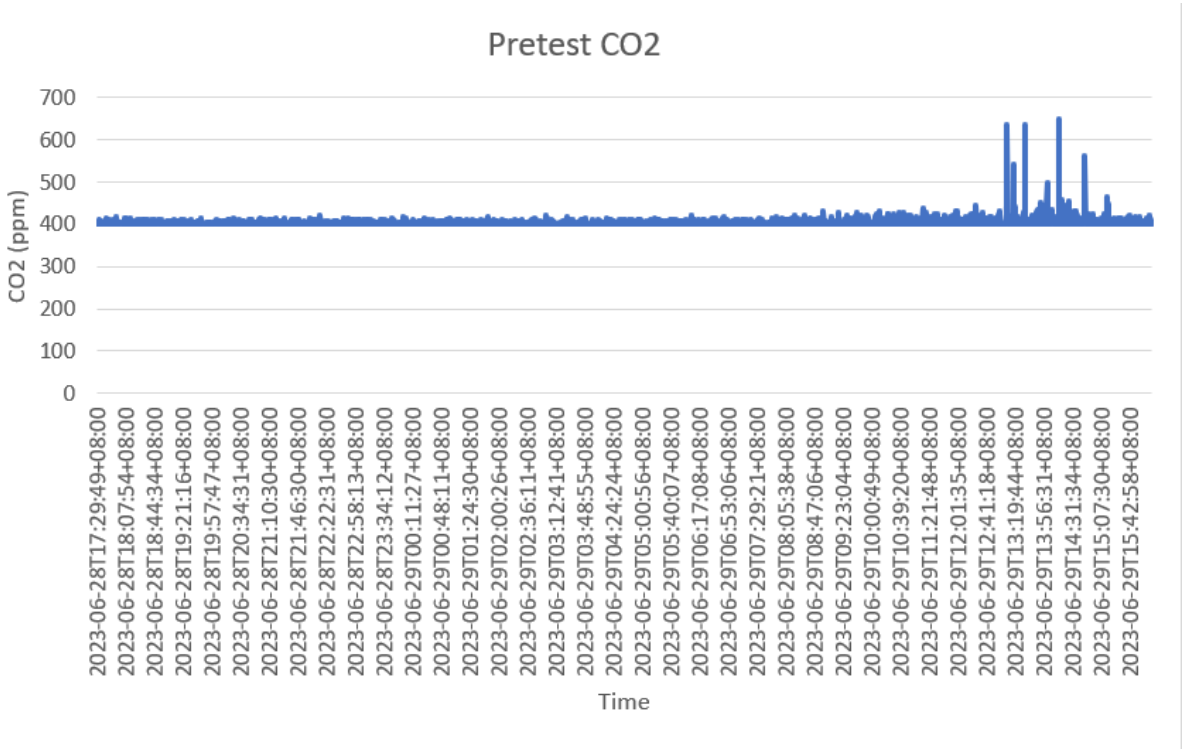


Figure 3.16: Pretest CO_2 level

As seen in Fig. 3.16, between 5:30 pm and approximately 1:30PM of the next day, the sensor recorded consistent CO_2 levels. The measurements indicated an average CO_2 level of approximately 450 ppm. However, after 1:30 pm, a notable change was observed as the CO_2 readings began to increase gradually. The levels reached approximately 650 ppm, with occasional variations where they either remained steady at this value or decreased back to around 400 ppm.

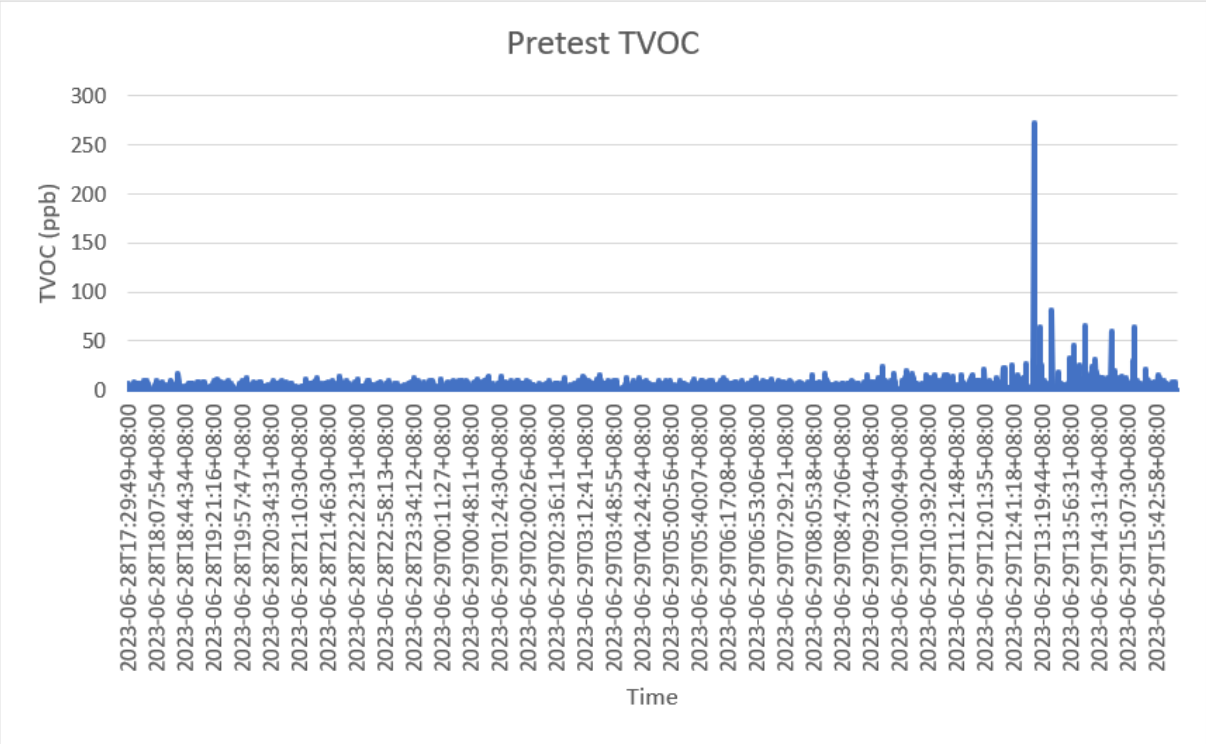


Figure 3.17: Pretest TVOC concentration

Similarly, the TVOC showed the same behavior as that of the CO_2. Between 5:30 pm and approximately 1:30 pm of the next day, the sensor recorded consistent TVOC concentration. The measurements indicated an average TVOC concentration of approximately 10 ppb. However, after 1:30 pm, a notable change was observed as the TVOC readings began to increase gradually. The levels reached approximately 50 ppb and 100 ppb, with occasional variations where they either remained steady at this value or decreased back to around 0 ppb. There is also one case that shows that the sensor measured 275 ppb around that time.

3.3.2 Posttesting

3.3.2.1 AM2320

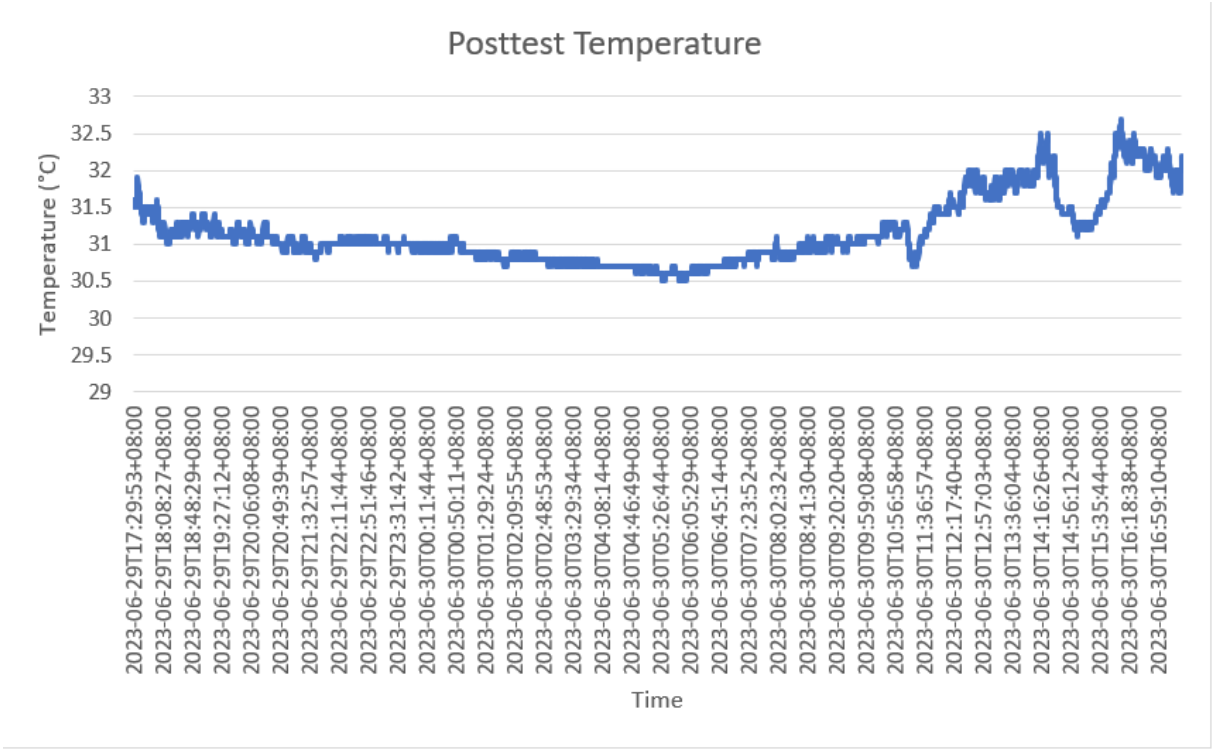


Figure 3.18: Posttest temperature

The posttest temperature readings obtained in °C were presented in figure -. The plot and data involved the same axes and intervals as in pretest. The highest temperature reading obtained was 32.7°C while the lowest was 30.5°C. The mean and standard deviation were respectively 31.15°C and 0.20°C. The night and day means and standard deviations were 30.93°C and 0.19°C, as well as 31.37°C and 0.51°C.

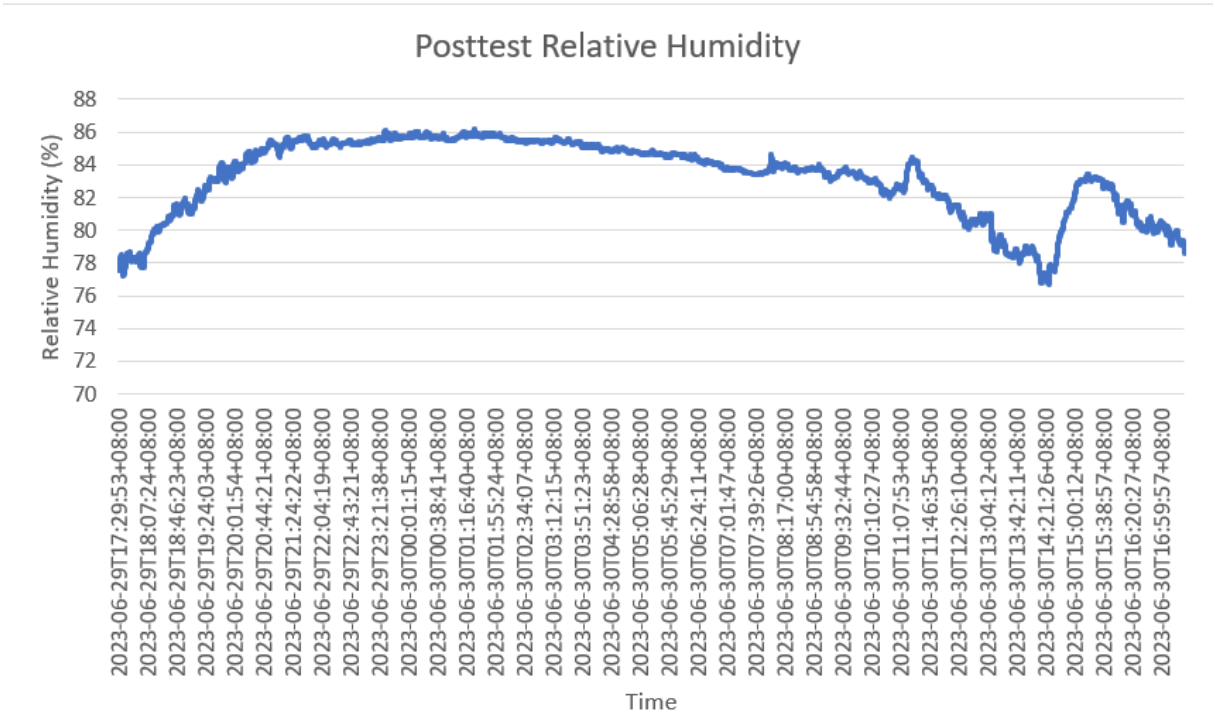


Figure 3.19: Posttest humidity

Figure - show then the posttest humidity measurements in %RH. Similarly, same axes and data intervals were involved as in pretest. The values reached up to 86.2%RH and dow to 76.7%RH. As per analyses, the mean was 83.38%RH and the standard deviation was 1.62%RH. At night, the mean was 84.64%RH and the standard deviation was 1.62%RH. During the day, the data had a mean of 82.02%RH with a standard deviation of 1..97%RH.

3.3.2.2 BH1750

The data from the calibrated light sensor during the post test is as shown in 3.20. From, the data it can be seen that it is logical as it increases starting from 5am and peaks around 11am and 4pm and dips during noon since the sun will be overhead. Furthermore, the recorded duration where the light level is considered in low light or above (>269 lx) is measured to be around 5hrs, thus the room meets the light requirement of the chose plant although not optimal. A light system can be implemented to complete the full 8hrs of low light or above.

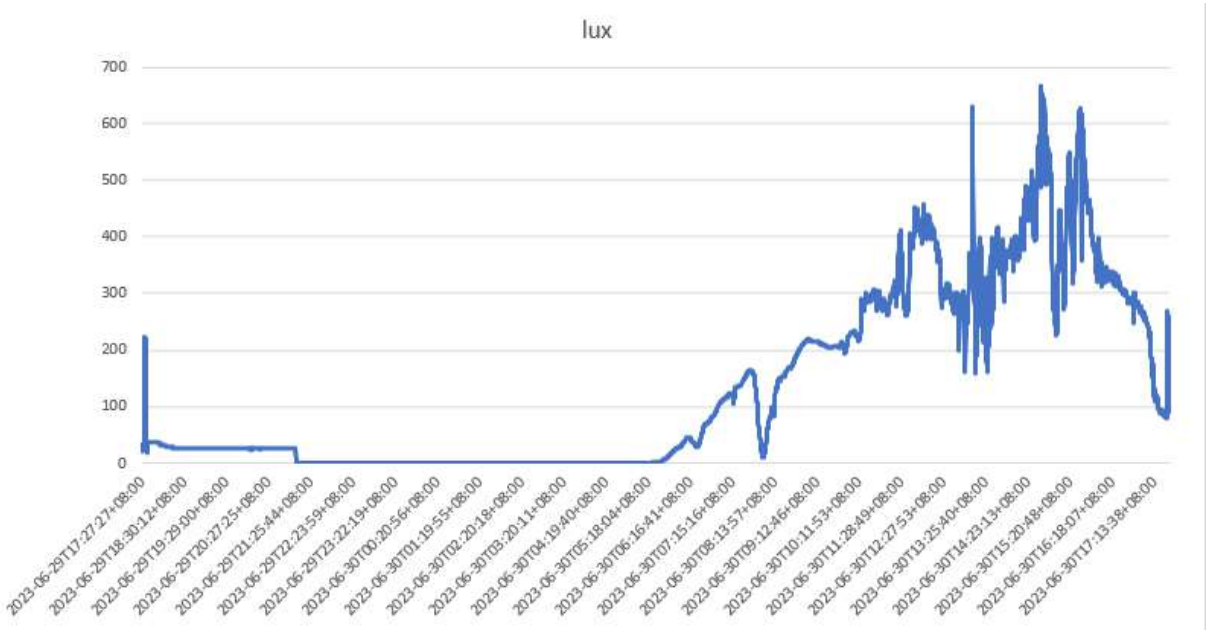


Figure 3.20: Posttest light levels

3.3.2.3 SGP30

This posttest entails evaluating the experimental group, a room where the plants are present. This aims to examine the impact of plants on indoor air quality. By introducing plants into the environment, the sensor's readings are monitored to detect and changes in CO_2 levels and TVOC concentration.

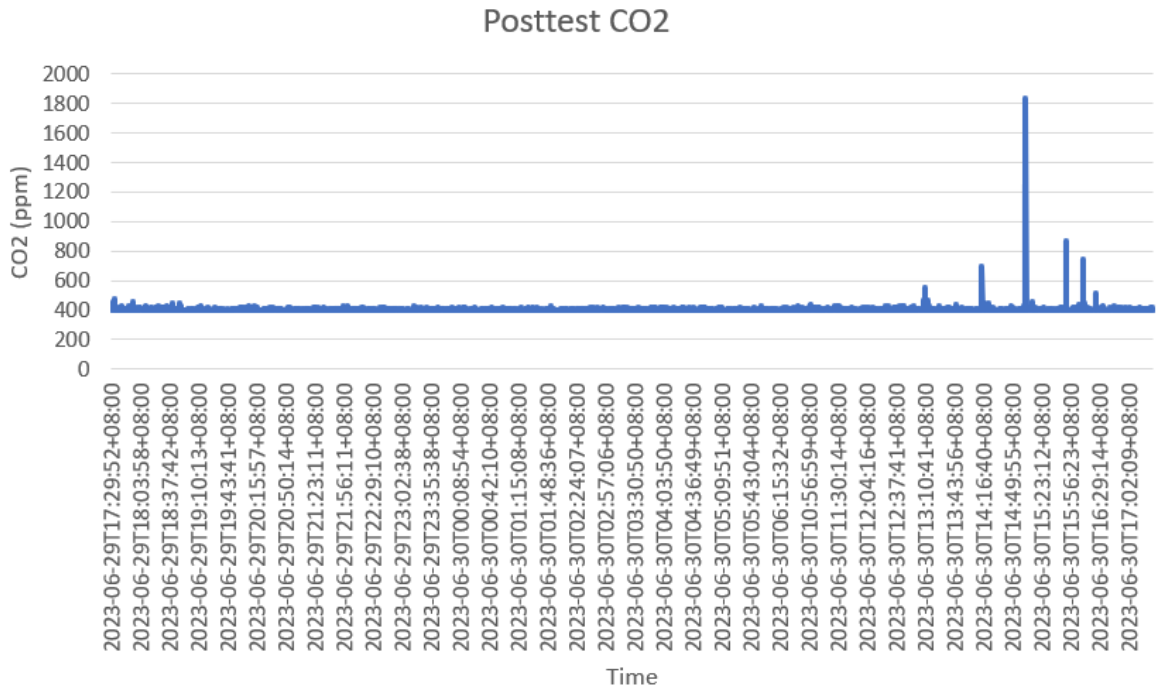


Figure 3.21: Posttest CO_2 level

In the posttesting phase, a similar trend to the pretest was observed as shown in Fig. 3.21. Between 5:30 pm and approximately 1:30 pm of the following day, the CO_2 levels was recorded at approximately 450 ppm. However, after 1:30 pm, there were instances where the CO_2 readings exhibited fluctuations. Some readings ranged from 600 ppm to 800 ppm, which then decreased to around 500 ppm. It is important to note that in one particular case, the CO_2 levels reached approximately 1800 ppm, surpassing the expected threshold. This finding raises concern since such elevated CO_2 levels should not be reached in a well-ventilated environment.

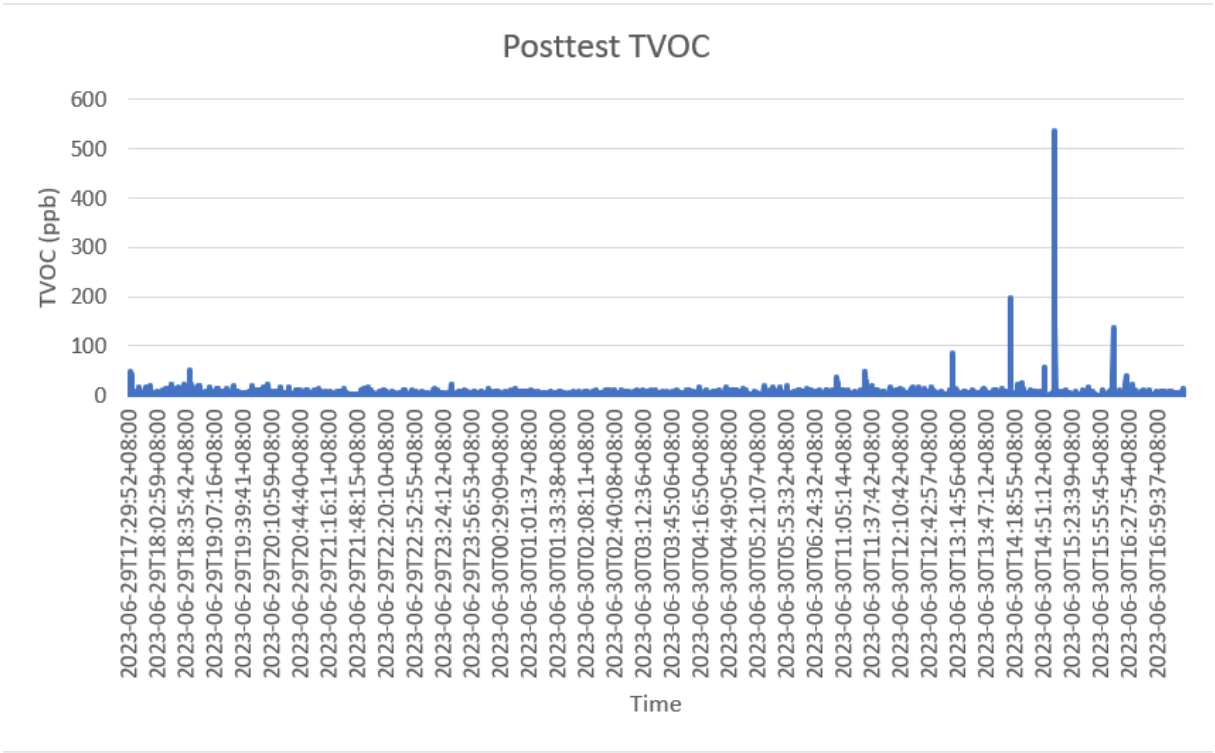


Figure 3.22: Posttest TVOC concentration

Much like the CO_2 readings, the TVOC readings for the posttest follow the same behavior as seen in Fig. 3.22. However, the values did not reach a surprising value, unlike the CO_2, and just had the same values as the one in the pretest.

Furthermore, it should be noted that during a specific timeframe between 6:00 am and 10:00 am, there was an instance where the sensor did not transmit any data. This temporary interruption in data transmission poses a challenge in comprehensively assessing the air quality parameters during that particular period. As a result, a gap in the data set exists, limiting the ability to capture the complete picture of air quality dynamics during that time frame.

3.4 Discussion

The discussion for the data obtained from the AM2320 sensor, namely the calibration and deployment data, involves determining the parameter differences, observations, recommendations, and statistical test. For the calibration, it was observed that the temperature lowered, and the humidity dropped by about 10 for the outside setup as compared to the crowded room setup. The readings gathered also matched with the

records from [26], where it was stated that the maximum temperature in Central Luzon is 32.2°C, the annual temperature in the Philippines is 28.1°C, and the relative humidity ranges from 75-85%RH. For the four experiment setup, it was observed that spraying caused a temperature decrease and a humidity increase, breathing caused a temperature increase and a humidity retention, fanning caused a temperature decrease and a slight humidity decrease due to air movements, and lastly, blowing really close to the sensor produced a spike to both temperature and humidity. All measurements were logical and made sense scientifically.

Observations for the temperature readings from the pretest were a decrease in the maximum and minimum values as well as in the mean and standard deviation implying that the data becomes closer. Night mean increased and day mean decreased. For both tests, the temperatures started to rise at around 06:30 and started to drop at around 15:30. For humidity, it was found out that the peak, mean and standard deviation values increased while the bottom value decreased. Night mean increased and day mean decreased slightly. The humidity levels started to rise at around 17:00 and started to drop at 07:00, and there was also a gradual drop at 22:00 during the comparison test. It can be seen by comparing the temperature and humidity graphs that they have somewhat an inverse relationship, which is reasonable as warmer air holds more moisture to saturate leading to lower relative humidity [27].

Variance	0.350444176	0.195990766	4.250438222	4.942109636
Observations	5436	5148	5436	5148
Hypothesized Mean Difference	0		0	
df	10049		10407	
t Stat	13.34749341		-0.274605258	
P(T<=t) one-tail	1.34848E-40		0.391812488	
t Critical one-tail	0		0	
P(T<=t) two-tail	2.70E-40		0.783624977	
t Critical two-tail	0.674514165		0.674513325	

Figure 3.23: t-Test for temperature and humidity

To determine the differences between the pretest and posttest data for the AM2320 sensor, t-Test: Two Sample Assuming Unequal Variances and a 0.05 level of significance were used via MS Excel 365, as shown in the figure above. From that, it was found out that the temperatures significantly decreased during the posttest and there was no significant difference between the pretest and posttest humidity readings.

Regarding the SGP30's pretesting, the significant shift of the air quality suggests a potential correlation with an increase in the number of people present in the room, possibly due to other groups utilizing the

same space.

In the posttesting, further analysis suggests that the unexpected spike in CO_2 levels may be attributed to other groups utilizing similar gas sensors in the vicinity. It is plausible that these groups were conducting experiments to test the measuring capabilities of their own sensors, thereby introducing additional CO_2 into the environment.

While most of the posttest results aligned with the pretest findings, the isolated instance of CO_2 levels reaching 1800 ppm indicates the need for vigilance in monitoring and maintaining optimal air quality conditions.

For the air quality, only the CO_2 will go through statistical analysis since [2] only uses CO_2 for comparison in their study. Using Excel, the t-Test: Two-Sample assuming Unequal Variances was utilized to determine if the introduction of plants had a significant effect on the CO_2 levels with a significance level of 0.05.

t-Test: Two-Sample Assuming Unequal Variances		
CO2		
	Pretest	*Posttest*
Mean	403.0131	404.22
Variance	93.50947	951.6567
Hypothesized Mean Difference	0	
df	4208	
t Stat	-2.24702	
P(T<=t) one-tail	0.012345	
t Critical one-tail	1.645216	
P(T<=t) two-tail	0.02469	
t Critical two-tail	1.960528	

Figure 3.24: t-Test for CO_2

Shown in Fig. 3.24 is the result of the t-Test. It can be seen that the mean of the pretest and posttest are 403.0131 ppm and 404.22 ppm, respectively. This contradicts on what is observed in [2] and [7] as the posttest should yield a much lower value than that of the pretest. The variance of the posttest is also much larger than that of the pretest and this may be due to the large recorded spike in Fig. 3.21. The p value for the two-tail is calculated to be 0.02469, which is less than the significance level. This means that there was a significant effect as the plant is introduced in the room.

The t-Test analysis revealed a statistically significant effect upon the introduction of plants into the room. However, it is important to acknowledge that the conclusive interpretation of this effect is hindered by external factors, rendering the findings inconclusive. Additionally, it can be observed that the values may

contradict the findings of related studies.

Chapter 4

Conclusion

Looking back at the project objectives, all were achieved during the duration of the project. An IoT based monitoring system was successfully developed and utilized to reliably monitor the environmental factors that are relevant to Golden Pothos health, as the data produced on the web interface coded were reliable, accurate, and involved the appropriate timings with little to no delays. The assigned sensors were strictly and effectively utilized and the data were reliable and logical, and they were sent and processed to a cloud database, Thingspeak. Furthermore, the data were shown using a simple web interface that also shows the list of requirements for Golden Pothos as well as the implications of the current readings from the sensor. The virtual actuators also simulated automatic plant cultivation, controlled using the assigned microcontrollers. Finally, the effects of the integration of Golden Photos were determined. The introduction of the plant had significantly reduced the overall temperature and increased the CO_2 levels, but have no significant effects on the CO_2 and TVOC levels. Although it should be noted that the CO_2 level measurements were likely affected by external factors such as other groups conducting their own experiments.

From the study, recommendations for parallel future project were determined such as the control and experiment to be conducted simultaneously and for a longer duration to improve the generalizability of the study. Future experiments could also be conducted in different room settings and plant varieties, which were not done due to limited resources. It is also recommended to have a stable internet connection since data were lost whenever the units are disconnected from the internet. Furthermore, future studies can also integrate actual actuators and plant care systems such watering, air, and light systems. For the AM2320 sensor, it is recommended to directly solder the unit to the PCB instead of using header pins, due to it having thin legs.

Bibliography

[1] K. Rose, S. D. Eldridge, and L. Chapin, "The internet of things : An overview understanding the issues and challenges of a more connected world", 2015.

[2] A. PichlhÃ¶fer, E. Sesto, J. Hollands, and A. Korjenic, "Health-related benefits of different indoor plant species in a school setting", *Sustainability*, vol. 13, no. 17, 2021, ISSN: 2071-1050. DOI: 10.3390/su13179566. [Online]. Available: https://www.mdpi.com/2071-1050/13/17/9566.

[3] E. R. Kaburuan, R. Jayadi, and Harisno, "A design of iot-based monitoring system for intelligence indoor micro-climate horticulture farming in indonesia", *Procedia Computer Science*, vol. 157, pp. 459–464, 2019, The 4th International Conference on Computer Science and Computational Intelligence (ICCSCI 2019) : Enabling Collaboration to Escalate Impact of Research Results for Society, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.09.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919311585.

[4] R. K. Raanaas, K. H. Evensen, D. Rich, G. SjÃžstrÃžm, and G. Patil, "Benefits of indoor plants on attention capacity in an office setting", *Journal of Environmental Psychology*, vol. 31, no. 1, pp. 99–105, 2011, ISSN: 0272-4944. DOI: https://doi.org/10.1016/j.jenvp.2010.11.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0272494410001027.

[5] I. Danielski, Ã. Svensson, K. Weimer, L. Lorentzen, and M. Warne, "Effects of green plants on the indoor environment and wellbeing in classroomsmdash;a case study in a swedish school", *Sustainability*, vol. 14, no. 7, 2022, ISSN: 2071-1050. DOI: 10.3390/su14073777. [Online]. Available: https://www.mdpi.com/2071-1050/14/7/3777.

[6] S. Sadrizadeh *et al.*, "Indoor air quality and health in schools: A critical review for developing the roadmap for the future school environment", *Journal of Building Engineering*, vol. 57, p. 104 908, 2022, ISSN: 2352-7102. DOI: https://doi.org/10.1016/j.jobe.2022.104908. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352710222009202.

[7] Suhaimi, Mohd Mahathir *et al.*, "Effectiveness of indoor plant to reduce co2 in indoor environment", *MATEC Web Conf.*, vol. 103, p. 05 004, 2017. DOI: 10.1051/matecconf/201710305004. [Online]. Available: https://doi.org/10.1051/matecconf/201710305004.

[8] B. Pennisi, "GROWING INDOOR PLANTS with Success", Dec. 2012.

[9] *ESP8266 WiFi Module (WRL-17146)*, ESP8266, SparkFun Electronics.

[10] *Universal Asynchronous Receiver Transmitter (UART) Protocol - GeeksforGeeks — geeksforgeeks.org*, https://www.geeksforgeeks.org/universal-asynchronous-receiver-transmitter-uart-protocol/, [Accessed 01-Jul-2023].

[11] *Ceiling Fan Design Guide*, Center for the Built Environment, Mar. 2020.

[12] *Datasheet SGP30: Indoor Air Quality Sensor for TVOC and CO2eq Measurements*, SGP30, Sensirion, May 2020.

[13] *STM32 I2C Configuration using Registers â» ControllersTech — controllerstech.com*, https://controllerstech.com/stm32-i2c-configuration-using-registers/, [Accessed 25-Jun-2023].

[14] ElectroHobby, *STM32 AM2320 I2C — youtu.be*, https://youtu.be/NM7RGOpcEw4, [Accessed 01-Jul-2023].

[15] *Digital Temperature and Humidity Sensor AM2320 Product Manual*, AM2320, AOSONG.

[16] *Temperature amp; Humidity 101 — acinfinity.com*, https://acinfinity.com/pages/grower-setup/temperature-and-humidity.html, [Accessed 29-Jun-2023].

[17] *Digital Temperature and Humidity Sensor AM2320 Product Manual*, AM2320, AOSONG.

[18] *How to Increase Houseplant Humidity with a Pebble Tray — gardening.org*, https://gardening.org/pebble-tray/, [Accessed 29-Jun-2023].

[19] A. Meehan, *The 75p baking soda hack to dehumidify your home and prevent condensation — glasgowlive.co.uk*, https://www.glasgowlive.co.uk/news/75p-baking-soda-hack-dehumidify-25771949, [Accessed 29-Jun-2023].

[20] *Digital 16bit serial output type ambient light sensor ic*, BH1750FVI, ROHM Semiconductors, Apr. 2009.

[21] M. Projekte, *STM32 Air Quality Sensor SGP30 — youtube.com*, https://www.youtube.com/watch?v=GbHgK9mSDfk&ab_channel=MikrocontrollerProjekte, [Accessed 29-Jun-2023], 2018.

[22] *Top 12 Software Development Methodologies - TatvaSoft Blog — tatvasoft.com*, https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/, [Accessed 28-Jun-2023].

[23] *How to Log Data into Thingspeak using STM32 and ESP8266 — controllerstech.com*, https://controllerstech.com/data-logger-using-stm32-and-esp8266/, [Accessed 04-Jun-2023], 2020.

[24] K. Rawal and G. Gabrani, "IoT based computing to monitor indoor plants by using smart pot", *SSRN Electronic Journal*, 2020. DOI: 10.2139/ssrn.3562964. [Online]. Available: https://doi.org/10.2139/ssrn.3562964.

[25] *Thingspeak*. [Online]. Available: https://thingspeak.com/.

[26] *That's how warm it is in the Philippines: 31.9 â°C on average per year and over 2100 hours of sunshine! — worlddata.info*, https://www.worlddata.info/asia/philippines/climate.php, [Accessed 30-Jun-2023].

[27] *Chapter 7 - Relationship between temperature and moisture — Animal amp; Food Sciences — afs.ca.uky.edu*, https://afs.ca.uky.edu/poultry/chapter-7-relationship-between-temperature-and-moisture, [Accessed 1-Jul-2023].

Appendix A

Sensor Interfacing Code

AM2320 main.c library

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************

  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************

  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the
     LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
```

```
  */
/* USER CODE END Header */
/* Includes
   ------------------------------------------------------------------*/

#include <ESP8266_Code.h>
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32f4xx.h"
#include "math.h"
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro
   -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
```

```
/* Private variables
   -----------------------------------------------------------*/
UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
float temp, humid;
float buf_data[6];
//HAL_StatusTypeDef status;
//#define LED_ONOFF(); GPIOC->ODR ^= GPIO_PIN_13;
#define AM2320_ADDRESS  0xB8
unsigned int tick;
volatile unsigned int auto_recon=60000;

char *am_pm;
float tempF;
float direct_fan=0.0;                                    //0- off, 1-
   direct, 2- oscillating
float ceiling_fan=0.0;                                   //0- off, 1- CW
    (Circulate warm air), 2- CC (Pushes cool air down)
float humidity_tray=0.0;                                 //0- off, 1-
   drain water, 2- pump water
float moisture_absorber_tray=0.0;                //0- close, 1- open;
   silica gel?, rock salt?, calcium chloride?
/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);

/* USER CODE BEGIN PFP */
```

```
/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
//Notes
/*
STM32F4 AM2320 No HAL (Hardware Abstraction Layer)
1. Configure GPIO Pins- connect AM2320 pins
2. I2C Init (Config)- speed, addressing mode, ack...
3. I2C Start Condition-
        -pull SDA low, SCL high
4. Send I2C Device Address- of AM2320, write bit- 0
        -0xB8 for write, followed by write bit, wait for ack?
5. Send I2C Register Address- from where you want to read the data (e.g
   . temp and humid... datasheet)
        -0x03, wait for ack
6. I2C Stop-
7. Delay- datasheet
8. I2C Start-
9. I2C Device Address- 1- read bit
        -0xB9 for read, read bit
10. Read Data- datasheet- register structure and data format
        -Receive data bytes from sensor
        -Generate ACK- more bytes are expected
        -NACK- last byte
        -Repeat ACK/NACK for each byte
11. I2C Stop-
        -by releasing SDA line while SCL is high
12. Process Data- accd. sensor's data format and ur application
*/

/*
```

```
AM2320 Specs
*5V, 4.7k for SCL and SDA, 5V-SDA-GND-SCL
*I2C bus mode, 2s per read, max 3s total comms
*1byte, function codes 1-127, output- 16bits (bit15 is sign)
*/

/*
I2C_Config
1. Enable the I2C CLOCK and GPIO CLOCK
2. Configure the I2C PINs for ALternate Functions
        a) Select Alternate Function in MODER Register
        b) Select Open Drain Output
        c) Select High SPEED for the PINs
        d) Select Pull-up for both the Pins
        e) Configure the Alternate Function in AFR Register
3. Reset the I2C
4. Program the peripheral input clock in I2C_CR2 Register in order to
   generate correct timings
5. Configure the clock control registers
6. Configure the rise time register
7. Program the I2C_CR1 register to enable the peripheral
*/


//Inits
void GPIO_Config(void) {
        RCC->AHB1ENR |= (1<<1);                    //GPIOB Clock Enable

        GPIOB->MODER &= ~(1<<16);                  //Alternate Function
           Mode- 10
        GPIOB->MODER |= (1<<17);
        GPIOB->MODER &= ~(1<<18);
        GPIOB->MODER |= (1<<19);
```

```
        GPIOB->OTYPER |= (1<<8);                    //Output Open-drain- 1
        GPIOB->OTYPER |= (1<<9);

        GPIOB->OSPEEDR |= (1<<16);                  //High Speed- 11, 50Mhz
            ?
        GPIOB->OSPEEDR |= (1<<17);
        GPIOB->OSPEEDR |= (1<<18);
        GPIOB->OSPEEDR |= (1<<19);

        GPIOB->PUPDR &= ~(1<<16);                   //Pull-up- 01, No Pull-
             00?
        GPIOB->PUPDR &= ~(1<<17);
        GPIOB->PUPDR &= ~(1<<18);
        GPIOB->PUPDR &= ~(1<<19);

        GPIOB->AFR[1] &= ~(1<<0);                   //AF6- 0110
        GPIOB->AFR[1] &= ~(1<<1);                   //Wrong ung una, AF4-
            0100 (for I2C1)
        GPIOB->AFR[1] |= (1<<2);
        GPIOB->AFR[1] &= ~(1<<3);

        GPIOB->AFR[1] &= ~(1<<4);                   //AF7- 0111
        GPIOB->AFR[1] &= ~(1<<5);
        GPIOB->AFR[1] |= (1<<6);
        GPIOB->AFR[1] &= ~(1<<7);
}

void I2C_Config(void) {
        RCC->APB1ENR |= (1<<21);                    //I2C Clock Enable

        RCC->APB1RSTR |= RCC_APB1RSTR_I2C1RST;
        RCC->APB1RSTR &= ~RCC_APB1RSTR_I2C1RST;
```

```
I2C1->CR1 |= (1<<15);                    //Reset- for errors or
    locked states
I2C1->CR1 &= ~(1<<15);

I2C1->CR2 |= (50<<0);                    //50MHz, 100kHz; used 8
    MHz as APB1 PCLK; 8

I2C1->CCR &= ~(1<<15);                   //SM
```

+

```
          ', PCLK- 8MHz? (>=2MHz for SM)
I2C1->CCR &= ~(1<<14);                   //Since not FM
I2C1->CCR |= (250<<0);                   //250, High,
    computation on CtrlsTech; 40

I2C1->TRISE |= (51<<0);                  //50+1, "; 9

//Configure I2C1 addr mode, dual addr mode, gen call mode, own
    addr
I2C1->CR1 &= ~(1<<0);                    //Disable I2C1
```

```
        I2C1->OAR1 &= ~(1<<15);
        I2C1->OAR1 |= (1<<15);                    //7-bit addr mode
        I2C1->OAR2 &= (1<<0);
        I2C1->CR1 &= ~(I2C_CR1_ENGC | I2C_CR1_NOSTRETCH);

//      I2C1->OAR1 &= ~(I2C_OAR1_ADDMODE | I2C_OAR1_OA1EN);
//      I2C1->OAR1 |= I2C_OAR1_ADDMODE_7BIT;
//      I2C1->OAR2 &= ~I2C_OAR2_OA2EN;
//      I2C1->CR1 &= ~(I2C_CR1_GCEN | I2C_CR1_NOSTETCH);

        I2C1->CR1 |= (1<<10);                     //ACK Enable
        I2C1->CR1 |= (1<<0);                      //Peripheral Enable
}

//Functions
void I2C_Start(void) {
        I2C1->CR1 |= (1<<8);                              //Start-
            repeated
        while (!(I2C1->SR1 & (1<<0)));            //Wait for SB bit to
            set (1)
}

void I2C_Stop(void) {
        I2C1->CR1 |= (1<<9);                              //Stop I2C
}

void I2C_Write(uint8_t data) {
        I2C1->DR = data;
        while (!(I2C1->SR1 & (1<<7)));            //Wait for TXE but to
            set, BTF?
}

void I2C_Wake(void) {
```

```
        I2C_Start();

        (void) I2C1->SR1;
        I2C1->DR = AM2320_ADDRESS;
        HAL_Delay(1);


        I2C_Stop();
}

void start_sequence(uint8_t dir) {
        I2C_Start();

        (void) I2C1->SR1;
                                //Addr
        I2C1->DR = dir == 0? AM2320_ADDRESS : (AM2320_ADDRESS + 1);

        while (!(I2C1->SR1 & (1<<1)));
                //Wait for ADDR bit to set
        (void) I2C1->SR1;
                                //For SB?
        (void) I2C1->SR2;
                                //For ADDR?
}

unsigned int crc16(uint8_t *ptr, uint8_t len) {         //Based on
   datasheet
        unsigned int crc = 0xFFFF;
        uint8_t i;

        while(len--) {
                crc ^= *ptr++;
                for(i=0; i<8; i++) {
```

```
                        if(crc & 0x01) {
                                crc >>= 1;
                                crc ^= 0xA001;
                        }
                        else {
                                crc >>= 1;
                        }
                }
        }

        return crc;
}

void AM2320_ReadData(void) {                                    //*t, *h
        uint8_t i;
        uint8_t buf[8];
        //uint8_t data_t[3];

        I2C_Wake();

        //Send Read Command
        start_sequence(0);
                //TX
        I2C_Write(0x03);
        I2C_Write(0x00);
        I2C_Write(0x04);
        I2C_Stop();
        HAL_Delay(1);

        //Read Data
        start_sequence(1);
                //RX
        for(i=0;i<8;i++) {
```

```
                while (!(I2C1->SR1 & (1<<6)));          //Wait for RXNE
                    bit to set
                buf[i] = I2C1->DR;
        }
        I2C_Stop();

        //CRC Check
//      data_t[0] = 0x03;
//      data_t[1] = 0x00;
//      data_t[2] = 0x04;
//
//      HAL_I2C_IsDeviceReady(&hi2c1, 0xB8, 2, 1);
//      HAL_I2C_Master_Transmit(&hi2c1, 0xB8, data_t, 3, 1);
//      HAL_I2C_Master_Receive(&hi2c1, 0xB9, buf, 8, 2);

        unsigned int Rcrc = buf[7] << 8;
        Rcrc += buf[6];
        if (Rcrc == crc16(buf, 6)) {
                unsigned int temperature = ((buf[4] & 0x7F) << 8) + buf
                    [5];
                temp = temperature / 10.0;
                temp = (((buf[4] & 0x80) >> 7) == 1) ? ((temp) * (-1))
                    : temp;
                unsigned int humidity = (buf[2] << 8) + buf[3];
                humid = humidity / 10.0;
                //return 0;
        }
        //return 2;
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
```

```
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

  /* MCU Configuration
     --------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();

  /* USER CODE BEGIN 2 */
  GPIO_Config();
  I2C_Config();
  ESP_Init("JEDEDCHICKEN", "ChickenZ");                    //"JEDEDCHICKEN
     ", "ChickenZ"; "HABIBI", "HABIBIBI";
  I2C1->CR1 |= I2C_CR1_ACK;                  //ACK Enable
  /* USER CODE END 2 */
```

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
        auto_recon = HAL_GetTick() + 60000;

        //LED_ONOFF();
        AM2320_ReadData();

        //Virtual Actuators
        //Fan system- 1 direct fan, 1 ceiling fan
        //Golden Pothos / "Epipremnum aureum" Requirements- T=2 (65degF
            night, 75degF day), H=2 (25%-49%), W=2 (Soil surface must
           be dry before re-watering)
        //1hr sensor data gathering...
        //Time
        time_t current_time = time(NULL);
        struct tm *local_time = localtime(&current_time);
        int hour = local_time->tm_hour;
        am_pm = (hour < 12) ? "AM" : "PM";

        //C-F conversion
        tempF = ((9.0/5) * temp) + 32.0;

        //
        if (strcmp(am_pm, "AM") == 0) {          //If equal
                if (tempF < 75.0) {
                        direct_fan=0.0;                          //Off
                        ceiling_fan=1.0;                         //CW
                }
                else if (tempF > 75.0) {
```

```
                direct_fan=1.0;                                 //
                    Direct
                ceiling_fan=2.0;                                //CC
        }
}
else {
        if (tempF < 65.0) {
                direct_fan=0.0;
                ceiling_fan=1.0;
        }
        else if (tempF > 65.0) {
                direct_fan=1.0;
                ceiling_fan=2.0;
        }
}

//Water system
if (humid < 25.0) {
        humidity_tray=2.0;                                      //Pump
        moisture_absorber_tray=0.0;             //Close
}
else if (humid > 49.0) {
        humidity_tray=1.0;                                      //Drain
        moisture_absorber_tray=1.0;             //Open
}

//
buf_data[0] = temp;
buf_data[1] = humid;
buf_data[2] = direct_fan;
buf_data[3] = ceiling_fan;
buf_data[4] = humidity_tray;
buf_data[5] = moisture_absorber_tray;
```

```
        ESP_Send_Multi("F7VIFLRMML37ATTC", 6, buf_data);
        HAL_Delay(15001);
                                                    //Must be >=15s
            delay
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) !=
     HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{
```

```
  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : USART_TX_Pin USART_RX_Pin */
  GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : PB8 PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

void SysTick_Handler(void)
```

```
{
        tick = HAL_GetTick();
        if (tick > auto_recon) { NVIC_SystemReset(); }

        HAL_IncTick();
}


#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
         line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

BH1750 main.c library

```
#include <stdint.h>
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "main.h"
#include "ESPDataLogger.h"
```

```
#define BH1750FVI_I2C_ADDRESS 0x23
#define BH1750FVI_I2C_PWRON 0x01


UART_HandleTypeDef huart1;
uint8_t check;
uint16_t Value_Buf[2];
uint16_t sending = 0;

void SystemClock_Config(void);
void Error_Handler(void);
static void GPIO_INIT(void);
void TIM2_CONFIG(void);
void TIM5_CONFIG(void);
void DELAY_us(uint16_t us);
void DELAY_ms(uint16_t ms);
void I2C_CONFIG(void);
void I2C1_Read(uint8_t SensorAddr, uint8_t* data, uint16_t size);
static void USART1_INIT(void);
void TIM2_IRQHandler(void);
void I2C1_Send(uint8_t address, uint8_t* data, uint16_t size);


void I2C1_Read(uint8_t SensorAddr, uint8_t* data, uint16_t size) {
    //Send the START COND
    I2C1->CR1 |= (0x1UL << (8U));
    while (!(I2C1->SR1 & (0x1UL << (0U))));

    //Send the ADDR + 1
    I2C1->DR = (SensorAddr << 1) | 1;
    while (!(I2C1->SR1 & (0x1UL << (1U))));
    (void)I2C1->SR2;
```

```
    for (uint16_t i = 0; i < size; i++) {
        //Enable acknowledge for all bytes except the last one
        if (i < size - 1) {
            I2C1->CR1 |= (0x1UL << (10U));
        } else {
            I2C1->CR1 &= ~(0x1UL << (10U));
        }

        //Wait for the byte to be received
        while (!(I2C1->SR1 & (0x1UL << (6U))));
        data[i] = I2C1->DR;
    }

    I2C1->CR1 |= (0x1UL << (9U)); // Stop Condition
}

int main(void){

  HAL_Init();
  SystemClock_Config();
  GPIO_INIT();
  USART1_INIT();
  TIM2_CONFIG();
  TIM5_CONFIG();
  I2C_CONFIG();
  //ESP_Init("TopShotZ", "12345678");
  ESP_Init("HABIBI", "HABIBIBI");
  TIM5->CR1 |= (1<<0);
  /* Initialize BH1750 */


  I2C1_Send(BH1750FVI_I2C_ADDRESS, BH1750FVI_I2C_PWRON, 1);
```

```
// Wait for 200ms for the sequence to complete
DELAY_ms(200);

while (1){
        uint8_t BH1750FVI_HResMode[1] = {0x10};
        I2C1_Send(BH1750FVI_I2C_ADDRESS, BH1750FVI_HResMode, 1);
        DELAY_ms(400);

        uint8_t SensorData[2] = {0};
        I2C1_Read(BH1750FVI_I2C_ADDRESS, SensorData, sizeof(
           SensorData));

        uint16_t light_lx = (SensorData[0] << 8) | SensorData[1];
        light_lx /= 1.2;
        uint16_t calibrated_lx = (light_lx -32)*0.8+26;
        light_lx = calibrated_lx;

    DELAY_ms(500);
        DELAY_ms(1000);
        Value_Buf[0] = light_lx;
        int light_level = 0;
        if (light_lx == 2152){
                light_level = 2;
        } else if ((light_lx <2152) & (light_lx > 807)) {
                light_level = 1;
        } else {
                light_level = 0;
        }
        Value_Buf[1] = light_level;
        sending = 1;
        ESP_Send_Multi("32BKAA8U9J3BZ824", 2, Value_Buf);
        TIM5->CNT = 0;
```

```
        HAL_Delay(14000);
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified
     parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 50;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
```

```
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
     HAL_OK)
  {
    Error_Handler();
  }
}

static void USART1_INIT(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

static void GPIO_INIT(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

}

void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */
  __disable_irq();
  while (1){
  }
  /* USER CODE END Error_Handler_Debug */
}

void TIM2_CONFIG(void){

// Enable Timer clock
```

```
        RCC->APB1ENR |= (1<<0);   // Enable clock for TIM2

//  Set the prescalar and the ARR
        TIM2->PSC = 50-1;   // 50Mhz/50 = 1 MHz ~~ 1 uS delay
        TIM2->ARR = 0xC350;   // MAX ARR value0 = 50,000

//  Enable the Timer, and wait for the update Flag to set
        TIM2->CR1 |= (1<<0); // Enable the Counter


        while (!(TIM2->SR & (1<<0)));   // UIF: Update interrupt flag..
            This bit is set by hardware when the registers are updated
}

void TIM5_CONFIG(void){

// enable Timer clock
        RCC->APB1ENR |= (1<<3);   // Enable clock for TIM5

// set the prescalar and the ARR
        TIM5->PSC = 30000-1;   // 50Mhz/30000 = 600 us delay
        TIM5->ARR = 0xC350;   // MAX ARR value0 = 50,000. 50000 * 600us
           = 30s

// enable the Timer, and wait for the update Flag to set
        TIM5->CR1 |= (1<<0); // Enable the Counter
        while (!(TIM5->SR & (1<<0)));   // UIF: Update interrupt flag..
            This bit is set by hardware when the registers are updated
}
void DELAY_us(uint16_t us) {
        TIM2->CNT = 0; //Reset the Counter
        while (TIM2->CNT < us); //the counter should increase by 1
           every us thus wait for the counter to reach the required
```

```
            delay
}

void I2C1_Send(uint8_t address, uint8_t* data, uint16_t size) {

    uint32_t counter;

    //Generate start cond.
    I2C1->CR1 |= I2C_CR1_START;
    counter = 10000;
    while (!(I2C1->SR1 & (0x1UL << (0U)))) {
        if (--counter == 0) {
            return 1;
        }
    }

    //Send the address
    I2C1->DR = (address << 1) & ~(0x1UL << (0U));
    counter = 10000;
    while (!(I2C1->SR1 & (0x1UL << (1U)))) {
        if (--counter == 0) {
            return;
        }
    }

    // Clear the ADDR
    (void)I2C1->SR1;
    (void)I2C1->SR2;

    //Send Data
    for (uint16_t i = 0; i < size; i++) {
        I2C1->DR = data[i];
        counter = 10000;
```

```
        while (!(I2C1->SR1 & (0x1UL << (7U)))) {
            if (--counter == 0) {
                return;
            }
        }
    }

    I2C1->CR1 |= (0x1UL << (9U)); //Stop
    return 0;
}



void DELAY_ms(uint16_t ms) {
        for (uint16_t i=0; i<ms; i++)
        {
                DELAY_us (1000); // delay of 1 ms
        }
}

void TIM5_IRQHandler(void){
                if (sending == 1) {
                NVIC_SystemReset();
                }
        }


void I2C_CONFIG(void) {
        // Enable I2C and GPIO clock
        RCC->APB1ENR |= (1 << 21); // enable I2C1 clock
        RCC->AHB1ENR |= (1<<1);  // SCL and SDA pins of Nucleo F411RE
           is located in PB8 and PB9, enable GPIOB clock
```

```
// Configure I2C pins
// set PB8 and PB9 as alternate functions
GPIOB->MODER |= (2<<16);
GPIOB->MODER |= (2<<18);

// set PB8 and PB9 as open drain output (default for I2C)
GPIOB->OTYPER |= (1<<8);
GPIOB->OTYPER |= (1<<9);

// set high speed for PB8 and PB9
GPIOB->OSPEEDR |= (3<<16);
GPIOB->OSPEEDR |= (3<<18);

// set PB8 and PB9 as internal pull up
GPIOB->PUPDR |= (1<<16);
GPIOB->PUPDR |= (1<<18);

// set PB8 and PB9 alternate function as I2C1 alternate
   function
GPIOB->AFR[1] |= (4<<0);
GPIOB->AFR[1] |= (4<<4);

// Reset and Set the I2C
I2C1->CR1 |= (1<<15);  // I2C under reset
I2C1->CR1 &= ~(1<<15);  // I2C not under reset

// Program peripheral input clock in I2C_CR2 to generate
   correct timings
I2C1->CR2 |= (50<<0);  // Set peripheral clock (f_PCLK) to 50
   Mhz

// Program I2C Clock Control Register (CCR)
I2C1->CCR = 240 <<0;  //Set CCR to 240
```

```
        // Set I2C Time Rise Register (TRISE) = ((TR_SCL)/(T_PCLK))+1 =
            ((1000 ns)/(1/50 Mhz))+1 = 51
        I2C1->TRISE = 51; //Set CCR to 51
        // Enable I2C
        I2C1->CR1 |= (1<<0);
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
         line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

SGP30 main.c library

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************

  * @file           : main.c
  * @brief          : Main program body
```

```
  ******************************************************************************

  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause
     license,
  * the "License"; You may not use this file except in compliance with
     the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************

  */
/* USER CODE END Header */

/* Includes
   ------------------------------------------------------------------*/
#include "main.h"

/* Private includes
   ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "ESPDataLogger.h"

/* USER CODE END Includes */

/* Private typedef
   -----------------------------------------------------------*/
```

```
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define
   ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define SGP30_I2C_ADDR 0x58
#define Init_air_quality 0x2003
#define Measure_air_quality 0x2008
/* USER CODE END PD */

/* Private macro
   ------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables
   ------------------------------------------------------------*/

UART_HandleTypeDef huart1;
uint16_t send_data[3];
unsigned int tick;
volatile unsigned int recon = 60000;
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes
   -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
static void MX_USART1_UART_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code
   ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
        recon = HAL_GetTick() + 60000;
  /* USER CODE END 1 */

  /* MCU Configuration
     ---------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
```

```
/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART1_UART_Init();
/* USER CODE BEGIN 2 */

ESP_Init("HABIBI", "HABIBIBI");
I2C_Init();
SGP30_Init();
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

        uint8_t signal[6];
        int alert;

        SGP30_Measure();
        HAL_Delay(10);
        SGP30_Read(signal, 6);

        int16_t CO2 = signal[0] << 8 | signal[1];
        int16_t TVOC = signal[3] << 8 | signal[4];
```

```
            send_data[1] = TVOC;
            send_data[0] = CO2;

            if(CO2 <= 1000){
                    alert = 0;
            }
            else if(CO2>1000 && CO2<=2000){
                    alert = 1;
            }
            else if(CO2>2000 && CO2<=5000){
                    alert = 2;
            }
            else if(CO2>5000 && CO2<=40000){
                    alert = 3;
            }
            else{
                    alert = 4;
            }

            send_data[2] = alert;

            ESP_Send_Multi("U0VX5BZ1ICYOUI7F", 3, send_data);
            HAL_Delay(15001);
    }
    /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
```

```
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

/** Configure the main internal regulator output voltage
*/
__HAL_RCC_PWR_CLK_ENABLE();
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
/** Initializes the CPU, AHB and APB busses clocks
*/
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 4;
RCC_OscInitStruct.PLL.PLLN = 100;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
RCC_OscInitStruct.PLL.PLLQ = 4;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
  Error_Handler();
}
/** Activate the Over-Drive mode
*/
if (HAL_PWREx_EnableOverDrive() != HAL_OK)
{
  Error_Handler();
}
/** Initializes the CPU, AHB and APB busses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
```

```
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) !=
      HAL_OK)
  {
    Error_Handler();
  }
}



/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

```
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void){
        GPIO_InitTypeDef GPIO_InitStruct = {0};

          /* GPIO Ports Clock Enable */
        __HAL_RCC_GPIOC_CLK_ENABLE();
        __HAL_RCC_GPIOH_CLK_ENABLE();
        __HAL_RCC_GPIOA_CLK_ENABLE();
        //   __HAL_RCC_GPIOB_CLK_ENABLE();

          /*Configure GPIO pins : USART_TX_Pin USART_RX_Pin */
        GPIO_InitStruct.Pin = USART_TX_Pin|USART_RX_Pin;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
        GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

```
/* USER CODE BEGIN 4 */
void I2C_Init(void){
        RCC->APB1ENR |= (1<<21);
        RCC->AHB1ENR |= (1<<1);

        GPIOB->MODER |= (2<<16);
        GPIOB->MODER |= (2<<18);

        GPIOB->OTYPER |= (1<<8);
        GPIOB->OTYPER |= (1<<9);

        GPIOB->OSPEEDR |= (3<<16);
        GPIOB->OSPEEDR |= (3<<18);

        GPIOB->PUPDR &= ~(1<<17);
        GPIOB->PUPDR |= (1<<16);
        GPIOB->PUPDR &= ~(1<<19);
        GPIOB->PUPDR |= (1<<18);

        GPIOB->AFR[1] |= (4<<0);
        GPIOB->AFR[1] |= (4<<4);

        I2C1->CR2 = 50;
        I2C1->CCR = 80;
        I2C1->TRISE = 51;

        I2C1->CR1 |= (1<<0);
}

void I2C_Start(void){
        I2C1->CR1 |= (1<<8); //Generate START
        while(!(I2C1->SR1 & (1 << 0))){} //Wait for START condition to
```

```
            be generated
}

void I2C_Stop(void){
        I2C1->CR1 |= (1<<9); //Generate STOP
        while(I2C1->CR1 & I2C_CR1_STOP){} //Wait for STOP condition to
            be generated
}

void I2C_Address(uint8_t address){
        I2C1->DR = address;
        while(!(I2C1->SR1 & (1<<1))){};
        uint8_t temp = I2C1->SR1 | I2C1->SR2;
}

void I2C_Write(uint8_t data){
        while(!(I2C1->SR1 & I2C_SR1_TXE)){} //Wait until I2C is ready
            for transmission
        I2C1->DR = data; //Write data to DR register
        while(!(I2C1->SR1 & I2C_SR1_BTF)){} //Wait until1 data
            transmission is complete
}

void I2C_Read(uint8_t address, uint8_t *buffer, uint8_t size){
        int remaining = size;

/**** STEP 1 ****/
        if (size == 1)
        {
                /**** STEP 1-a ****/
                I2C1->DR = address;
                                        //  send the address
                while (!(I2C1->SR1 & (1<<1)));                  // wait
```

```
                for ADDR bit to set

            /**** STEP 1-b ****/
            I2C1->CR1 &= ~(1<<10);
                        // clear the ACK bit
            uint8_t temp = I2C1->SR1 | I2C1->SR2;    // read SR1 and
                SR2 to clear the ADDR bit.... EV6 condition
            I2C1->CR1 |= (1<<9);  // Stop I2C

            /**** STEP 1-c ****/
            while (!(I2C1->SR1 & (1<<6)));                    // wait
                for RxNE to set

            /**** STEP 1-d ****/
            buffer[size-remaining] = I2C1->DR;       // Read the
               data from the DATA REGISTER

        }

/**** STEP 2 ****/
        else
        {
            /**** STEP 2-a ****/
            I2C1->DR = address;
                              //  send the address
            while (!(I2C1->SR1 & (1<<1)));                    // wait
                for ADDR bit to set

            /**** STEP 2-b ****/
            uint8_t temp = I2C1->SR1 | I2C1->SR2;    // read SR1 and
                SR2 to clear the ADDR bit

            while (remaining>2)
```

```
        {
                /**** STEP 2-c ****/
                while (!(I2C1->SR1 & (1<<6)));
                                // wait for RxNE to set

                /**** STEP 2-d ****/
                buffer[size-remaining] = I2C1->DR;  // copy the
                    data into the buffer

                /**** STEP 2-e ****/
                I2C1->CR1 |= 1<<10;  // Set the ACK bit to
                   Acknowledge the data received

                remaining--;
        }

        // Read the SECOND LAST BYTE
        while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
        buffer[size-remaining] = I2C1->DR;

        /**** STEP 2-f ****/
        I2C1->CR1 &= ~(1<<10);  // clear the ACK bit

        /**** STEP 2-g ****/
        I2C1->CR1 |= (1<<9);  // Stop I2C

        remaining--;

        // Read the Last BYTE
        while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
        buffer[size-remaining] = I2C1->DR;  // copy the data
           into the buffer
}
```

```
}

void SGP30_Init(void){
        SGP30_Write(Init_air_quality);
        HAL_Delay(20);
}

void SGP30_Write(uint16_t command){
        I2C_Start();
        I2C_Address(SGP30_I2C_ADDR<<1);
        I2C_Write((uint8_t)(command >> 8));
        I2C_Write((uint8_t)command);
        I2C_Stop();
}

void SGP30_Read(uint8_t* buffer, uint8_t size){
        I2C_Start();
        while (!(I2C1->SR1 & I2C_SR1_SB));
          I2C1->DR = (SGP30_I2C_ADDR << 1) | 1;
          while (!(I2C1->SR1 & I2C_SR1_ADDR));
          I2C1->SR2;
          for (uint16_t i = 0; i < size; i++) {
                  if (i < size - 1) {
                          I2C1->CR1 |= I2C_CR1_ACK;
                  }
                  else {
                          I2C1->CR1 &= ~I2C_CR1_ACK;
                  }
                  while (!(I2C1->SR1 & I2C_SR1_RXNE));
                  buffer[i] = I2C1->DR;
          }
        I2C_Stop();
```

```
}

void SGP30_Measure(void){
        SGP30_Write(Measure_air_quality);
}


/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void){
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return
      state */

  /* USER CODE END Error_Handler_Debug */
}

void SysTick_Handler(void){
        tick = HAL_GetTick();
        if(tick>recon){
                NVIC_SystemReset();
        }
        HAL_IncTick();
}
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
     number
  *         where the assert_param error has occurred.
```

```
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
     line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file
        , line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
    FILE****/
```

Appendix B

Wifi Module Interface Code

UartRingBuffer.c library

```
/*
 * UartRingbuffer.c
 *
 *  Created on: 10-Jul-2019
 *      Author: Controllerstech
 *
 *  Modified on: 11-April-2020
 */

#include "UartRingbuffer.h"
#include <string.h>

/**** define the UART you are using  ****/

extern UART_HandleTypeDef huart1;

#define uart &huart1

/* put the following in the ISR
```

```
extern void Uart_isr (UART_HandleTypeDef *huart);

*/

/**********==========>>>>>>>>>>> NO CHANGES AFTER
    THIS ==========>>>>>>>>>>>**********/


ring_buffer rx_buffer = { { 0 }, 0, 0};
ring_buffer tx_buffer = { { 0 }, 0, 0};

ring_buffer *_rx_buffer;
ring_buffer *_tx_buffer;

void store_char(unsigned char c, ring_buffer *buffer);


void Ringbuf_init(void)
{
  _rx_buffer = &rx_buffer;
  _tx_buffer = &tx_buffer;

  /* Enable the UART Error Interrupt: (Frame error, noise error,
     overrun error) */
  __HAL_UART_ENABLE_IT(uart, UART_IT_ERR);

  /* Enable the UART Data Register not empty Interrupt */
  __HAL_UART_ENABLE_IT(uart, UART_IT_RXNE);
}

void store_char(unsigned char c, ring_buffer *buffer)
{
  int i = (unsigned int)(buffer->head + 1) % UART_BUFFER_SIZE;
```

```
  // if we should be storing the received character into the location
  // just before the tail (meaning that the head would advance to the
  // current location of the tail), we're about to overflow the buffer
  // and so we don't write the character or advance the head.
  if(i != buffer->tail) {
    buffer->buffer[buffer->head] = c;
    buffer->head = i;
  }
}

void Uart_flush (void)
{

          _rx_buffer->head = _rx_buffer->tail;
}

int Look_for (char *str, char *buffertolookinto)
{
        int stringlength = strlen (str);
        int bufferlength = strlen (buffertolookinto);
        int so_far = 0;
        int indx = 0;
repeat:
        while (str[so_far] != buffertolookinto[indx]) indx++;
        if (str[so_far] == buffertolookinto[indx]){
        while (str[so_far] == buffertolookinto[indx])
        {
                so_far++;
                indx++;
        }
        }
```

```
        else
                {
                        so_far =0;
                        if (indx >= bufferlength) return -1;
                        goto repeat;
                }

        if (so_far == stringlength) return 1;
        else return -1;
}

int Uart_read(void)
{
  // if the head isn't ahead of the tail, we don't have any characters
  if(_rx_buffer->head == _rx_buffer->tail)
  {
    return -1;
  }
  else
  {
    unsigned char c = _rx_buffer->buffer[_rx_buffer->tail];
    _rx_buffer->tail = (unsigned int)(_rx_buffer->tail + 1) %
       UART_BUFFER_SIZE;
    return c;
  }
}

void Uart_write(int c)
{
        if (c>=0)
        {
                int i = (_tx_buffer->head + 1) % UART_BUFFER_SIZE;
```

```
                // If the output buffer is full, there's nothing for it
                    other than to
                // wait for the interrupt handler to empty it a bit
                // ???: return 0 here instead?
                while (i == _tx_buffer->tail);

                _tx_buffer->buffer[_tx_buffer->head] = (uint8_t)c;
                _tx_buffer->head = i;

                __HAL_UART_ENABLE_IT(uart, UART_IT_TXE); // Enable UART
                    transmission interrupt
        }
}

int IsDataAvailable(void)
{
  return (uint16_t)(UART_BUFFER_SIZE + _rx_buffer->head - _rx_buffer->
     tail) % UART_BUFFER_SIZE;
}

void Uart_sendstring (const char *s)
{
        while(*s) Uart_write(*s++);
}

void Uart_printbase (long n, uint8_t base)
{
  char buf[8 * sizeof(long) + 1]; // Assumes 8-bit chars plus zero byte
     .
  char *s = &buf[sizeof(buf) - 1];

  *s = '\0';
```

```
  // prevent crash if called with base == 1
  if (base < 2) base = 10;

  do {
    unsigned long m = n;
    n /= base;
    char c = m - base * n;
    *--s = c < 10 ? c + '0' : c + 'A' - 10;
  } while(n);

  while(*s) Uart_write(*s++);
}

int Uart_peek()
{
  if(_rx_buffer->head == _rx_buffer->tail)
  {
    return -1;
  }
  else
  {
    return _rx_buffer->buffer[_rx_buffer->tail];
  }
}


int Copy_upto (char *string, char *buffertocopyinto)
{
        int so_far =0;
        int len = strlen (string);
        int indx = 0;

again:
```

```
TIM5->CNT - 0;
while (!IsDataAvailable()){
        if (TIM5->CNT >= 50000){
                NVIC_SystemReset();
        }
}
while (Uart_peek() != string[so_far])
        {
                buffertocopyinto[indx] = _rx_buffer->buffer[
                    _rx_buffer->tail];
                _rx_buffer->tail = (unsigned int)(_rx_buffer->
                    tail + 1) % UART_BUFFER_SIZE;
                indx++;
                TIM5->CNT = 0;
                while (!IsDataAvailable()){
                        if (TIM5->CNT >= 50000){
                                NVIC_SystemReset();
                        }

        }}
while (Uart_peek() == string [so_far])
{
        so_far++;
        buffertocopyinto[indx++] = Uart_read();
        if (so_far == len) return 1;
        TIM5->CNT = 0;
        while (!IsDataAvailable()){
                if (TIM5->CNT >= 50000){
                        NVIC_SystemReset();
                }
        }
}
```

```
        if (so_far != len)
        {
                so_far = 0;
                goto again;
        }

        if (so_far == len) return 1;
        else return -1;
}

int Get_after (char *string, uint8_t numberofchars, char *buffertosave)
{

        while (Wait_for(string) != 1);
        for (int indx=0; indx<numberofchars; indx++)
        {
                TIM5->CNT = 0;
                while (!IsDataAvailable()){
                        if (TIM5->CNT >= 50000){
                                NVIC_SystemReset();
                        }
                }
                buffertosave[indx] = Uart_read();
        }
        return 1;
}


int Wait_for (char *string)
{
        int so_far =0;
        int len = strlen (string);
```

```
again:
        TIM5->CNT = 0;
        while (!IsDataAvailable()){
                if (TIM5->CNT >= 50000){
                        NVIC_SystemReset();
                }
        }
        if (Uart_peek() != string[so_far])
        {
                 _rx_buffer->tail = (unsigned int)(_rx_buffer->tail +
                    1) % UART_BUFFER_SIZE ;
                goto again;

        }
        while (Uart_peek() == string [so_far])
        {
                so_far++;
                Uart_read();
                if (so_far == len) return 1;
                TIM5->CNT = 0;
                while (!IsDataAvailable()){
                        if (TIM5->CNT >= 50000){
                                NVIC_SystemReset();
                        }
                }
        }

        if (so_far != len)
        {
                so_far = 0;
                goto again;
        }
```

```
        if (so_far == len) return 1;
        else return -1;
}

void Uart_isr (UART_HandleTypeDef *huart)
{
          uint32_t isrflags   = READ_REG(huart->Instance->SR);
          uint32_t cr1its     = READ_REG(huart->Instance->CR1);

    /* if DR is not empty and the Rx Int is enabled */
    if (((isrflags & USART_SR_RXNE) != RESET) && ((cr1its &
       USART_CR1_RXNEIE) != RESET))
    {
        /******************
                    *  @note   PE (Parity error), FE (Framing error),
                       NE (Noise error), ORE (Overrun
                    *          error) and IDLE (Idle line detected)
                       flags are cleared by software
                    *          sequence: a read operation to USART_SR
                       register followed by a read
                    *          operation to USART_DR register.
                    * @note   RXNE flag can be also cleared by a read
                       to the USART_DR register.
                    * @note   TC flag can be also cleared by software
                       sequence: a read operation to
                    *          USART_SR register followed by a write
                       operation to USART_DR register.
                    * @note   TXE flag is cleared only by a write to
                       the USART_DR register.

        *********************/
              huart->Instance->SR;                       /* Read
                 status register */
```

```
    unsigned char c = huart->Instance->DR;      /* Read data
       register */
    store_char (c, _rx_buffer);  // store data in buffer
    return;
}

/*If interrupt is caused due to Transmit Data Register Empty */
if (((isrflags & USART_SR_TXE) != RESET) && ((cr1its &
   USART_CR1_TXEIE) != RESET))
{
    if(tx_buffer.head == tx_buffer.tail)
        {
          // Buffer empty, so disable interrupts
          __HAL_UART_DISABLE_IT(huart, UART_IT_TXE);

        }

     else
        {
          // There is more data in the output buffer. Send the next
              byte
          unsigned char c = tx_buffer.buffer[tx_buffer.tail];
          tx_buffer.tail = (tx_buffer.tail + 1) % UART_BUFFER_SIZE;

          /******************
          *  @note   PE (Parity error), FE (Framing error), NE (
             Noise error), ORE (Overrun
          *          error) and IDLE (Idle line detected) flags are
              cleared by software
          *          sequence: a read operation to USART_SR
             register followed by a read
          *          operation to USART_DR register.
```

```
                * @note   RXNE flag can be also cleared by a read to the
                   USART_DR register.
                * @note   TC flag can be also cleared by software
                   sequence: a read operation to
                *          USART_SR register followed by a write
                   operation to USART_DR register.
                * @note   TXE flag is cleared only by a write to the
                   USART_DR register.

                **********************/

                huart->Instance->SR;
                huart->Instance->DR = c;

            }
        return;
    }
}




/*** Depreciated For now. This is not needed, try using other functions
    to meet the requirement ***/
/*
uint16_t Get_position (char *string)
{
  static uint8_t so_far;
  uint16_t counter;
  int len = strlen (string);
  if (_rx_buffer->tail>_rx_buffer->head)
  {
```

```
        if (Uart_read() == string[so_far])
                        {
                          counter=UART_BUFFER_SIZE-1;
                          so_far++;
                        }
        else so_far=0;
  }
  unsigned int start = _rx_buffer->tail;
  unsigned int end = _rx_buffer->head;
  for (unsigned int i=start; i<end; i++)
  {
        if (Uart_read() == string[so_far])
              {
                counter=i;
                so_far++;
              }
        else so_far =0;
  }

  if (so_far == len)
       {
         so_far =0;
         return counter;
       }
  else return -1;
}


void Get_string (char *buffer)
{
        int index=0;

        while (_rx_buffer->tail>_rx_buffer->head)
```

```
        {
                if ((_rx_buffer->buffer[_rx_buffer->head-1] == '\n')
                   ||((_rx_buffer->head == 0) && (_rx_buffer->buffer[
                   UART_BUFFER_SIZE-1] == '\n')))
                        {
                                buffer[index] = Uart_read();
                                index++;
                        }
        }
        unsigned int start = _rx_buffer->tail;
        unsigned int end = (_rx_buffer->head);
        if ((_rx_buffer->buffer[end-1] == '\n'))
        {

                for (unsigned int i=start; i<end; i++)
                {
                        buffer[index] = Uart_read();
                        index++;
                }
        }
}
*/
```

ESPDataLogger.c library

```
/*
 * ESPDataLogger.c
 *
 *  Created on: May 26, 2020
 *      Author: controllerstech
 */

#include "UartRingbuffer.h"
#include "ESPDataLogger.h"
#include "stdio.h"
```

```
#include "string.h"


void bufclr (char *buf)
{
        int len = strlen (buf);
        for (int i=0; i<len; i++) buf[i] = '\0';
}


void ESP_Init (char *SSID, char *PASSWD)
{
        char data[80];

        Ringbuf_init();

        Uart_sendstring("AT+RST\r\n");
        HAL_Delay(1000);

        Uart_flush();

        /********* AT **********/
        Uart_sendstring("AT\r\n");
        while(!(Wait_for("OK\r\n")));

        Uart_flush();


        /********* AT+CWMODE=1 **********/
        Uart_sendstring("AT+CWMODE=1\r\n");
        while (!(Wait_for("OK\r\n")));

        Uart_flush();
```

```
        /********* AT+CWJAP="SSID","PASSWD" **********/
        sprintf (data, "AT+CWJAP=\"%s\",\"%s\"\r\n", SSID, PASSWD);
        Uart_sendstring(data);
        while (!(Wait_for("GOT IP\r\n")));

        Uart_flush();

        /********* AT+CIPMUX=0 **********/
        Uart_sendstring("AT+CIPMUX=0\r\n");
        while (!(Wait_for("OK\r\n")));

        Uart_flush();

}

void ESP_Send_Data (char *APIkey, int Field_num, uint16_t value)
{
        char local_buf[100] = {0};
        char local_buf2[30] = {0};

        Uart_sendstring("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n
            ");
        while (!(Wait_for("OK\r\n")));

        sprintf (local_buf, "GET /update?api_key=%s&field%d=%u\r\n",
           APIkey, Field_num, value);
        int len = strlen (local_buf);

        sprintf (local_buf2, "AT+CIPSEND=%d\r\n", len);
        Uart_sendstring(local_buf2);
        while (!(Wait_for(">")));
```

```
        Uart_sendstring (local_buf);
        while (!(Wait_for("SEND OK\r\n")));

        while (!(Wait_for("CLOSED")));

        bufclr(local_buf);
        bufclr(local_buf2);

        Ringbuf_init();

}

void ESP_Send_Multi (char *APIkey, int numberoffileds, uint16_t value
    [])
{
        char local_buf[500] = {0};
        char local_buf2[30] = {0};
        char field_buf[200] = {0};


        Uart_sendstring("AT+CIPSTART=\"TCP\",\"184.106.153.149\",80\r\n
            ");
        while (!(Wait_for("OK\r\n")));

        sprintf (local_buf, "GET /update?api_key=%s", APIkey);
        for (int i=0; i<numberoffileds; i++)
        {
                sprintf(field_buf, "&field%d=%u",i+1, value[i]);
                strcat (local_buf, field_buf);
        }

        strcat(local_buf, "\r\n");
```

```
        int len = strlen (local_buf);

        sprintf (local_buf2, "AT+CIPSEND=%d\r\n", len);
        Uart_sendstring(local_buf2);
        while (!(Wait_for(">")));

        Uart_sendstring (local_buf);
        while (!(Wait_for("SEND OK\r\n")));

        while (!(Wait_for("CLOSED")));

        bufclr(local_buf);
        bufclr(local_buf2);

        Ringbuf_init();

}
```

Appendix C

IoT Platform Code

```
<!DOCTYPE html>
<html>
<head>
<title>Project grEEEn</title>
<style>
  .bg {
    background-image: url("Golden-Pothos.png");
    background-repeat: no-repeat;
    background-size: cover;
    background-position: center;
  }
</style>
</head>


<body style="background-color: #b7cdb8;">


<p style="font-family: arial black; font-size:500%; text-align: center;
    position: relative; top: -0.5em;">
<img src="https://i.ibb.co/sqcxFXJ/Golden-Pothos.png" alt="Golden
```

```
    Pothos.png" width="153.6" height="153.6" style="vertical-align:
    middle">

gr<span style="color: #5370fd;">E</span><span style="color: #f40600;">E
    </span><span style="color: #4f8c46;">E</span>n
</p>

<p style="font-family: arial; font-size: 120%; text-align: center;">
<strong><i>Epipremnum aureum</i> (Golden Pothos) Requirements: </strong
    ><br>
1. Light: >200ft-c (high-light areas), 75-200ft-c (med.-light), 25-75ft
    -c (low-light), minimize exposure to direct sunlight<br>
2. Temperature: 65degF (night), 75degF (day) <br>
3. Relative Humidity: 25-49% <br>
4. Water: when surface is dry (every 1-2 weeks) <br>

<br><br>
<style>
hr {color: white; border-style: solid;}
</style>
<hr>
<br><br>
</p>

<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Light Sensor (BH1750) Readings</b> <br><br>

<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
    ="https://thingspeak.com/channels/2149974/charts/1?bgcolor=%23ffffff
    &amp;color=%235370fd&amp;dynamic=true&amp;results=60&amp;title=Light
    +level&amp;type=line&amp;xaxis=Time&amp;yaxis=Lux"></iframe>
<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
    ="https://thingspeak.com/channels/2149974/widgets/678763"></iframe>
```

```
<br><br>
</p>

<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Light Sensor (BH1750) Indicator</b> <br>
<span id="lightimply"></span> <br>
</p>


<script>
<!-- Make an HTTP GET request to ThingSpeak API -->
fetch('https://api.thingspeak.com/channels/2149974/feeds.json?results
    =1')
  .then(response => response.json())
  .then(data => {
    <!-- Extract the latest readings from the response -->
    const latestReadingField1 = data.feeds[0].field1;
    const latestReadingField2 = data.feeds[0].field2;

    <!-- Check Time -->
    const lightimply = document.getElementById('lightimply');
    const currentTime = new Date();
    const currentHour = currentTime.getHours();

    <!-- Conditions2 -->
    if (parseFloat(latestReadingField1) > 32000) {
      lightimply.innerHTML = '<span style="color: #fdec53;"><strong>THE
          PLANT IS IN DIRECT SUNLIGHT!.</strong></span>'
    }
    else if (parseFloat(latestReadingField2) < 1) {
      lightimply.innerHTML = '<span style="color: #5370fd;"><strong>The
          plant is in low light.</strong></span>' }
```

```
    else if (parseFloat(latestReadingField2) > 1) {
      lightimply.innerHTML = '<span style="color: #fdec53;"><strong>The
          plant is in high light.</strong></span>' }
    else {
      lightimply.innerHTML = '<span style="color: black;"><strong>The
         plant is in medium light.</strong></span>' }

  })
  .catch(error => console.error(error));
</script>
<br>

<br><br>
<style>
hr {color: white; border-style: solid;}
</style>
<hr>
<br><br>
</p>


<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Temperature and Humidity Sensor (AM2320) Readings</b> <br><br>

<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/charts/1?bgcolor=%23ffffff
   &color=%23f40600&dynamic=true&results=60&title=Temperature&type=line
   &yaxis=Degree+Celcius"></iframe>
<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/charts/2?bgcolor=%23ffffff
   &color=%23f40600&dynamic=true&results=60&title=Humidity&type=line&
   yaxis=%25+Relative+Humidity"></iframe>
<br><br>
```

```
</p>

<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Temperature and Humidity Sensor (AM2320) Indicators</b> <br>
<span style="font-size: 80%;"><strong>Legend: </strong><br>
1. Direct Fan: 0 (off), 1 (direct), 2 (oscillating) <br>
2. Ceiling Fan: 0 (off), 1 (CW, circulates warm air), 2 (CC, pushes
   cool air) <br>
3. Humidity Tray: 0 (off), 1 (drain), 2 (pump) <br>
4. Moisture Absorber Tray: 0 (close), 1 (open) <br>
</span>
<br>

<iframe width="260" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/widgets/676179"></iframe>
<iframe width="260" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/widgets/676180"></iframe>
<iframe width="260" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/widgets/676181"></iframe>
<iframe width="260" height="260" style="border: 1px solid #cccccc;" src
   ="https://thingspeak.com/channels/2166518/widgets/676182"></iframe>


<!-- Get data from thingspeak -->
<br><br><br>
<strong>Implications: </strong> <br>
Current Temperature: <span id="field1"></span> degree Celcius <br>
 <span id="tempimply"></span> <br>
Current Humidity: <span id="field2"></span> % Relative Humidity <br>

<span id="humidimply"></span> <br>
```

```
<script>
<!-- Make an HTTP GET request to ThingSpeak API -->
fetch('https://api.thingspeak.com/channels/2166518/feeds.json?results
    =1')
  .then(response => response.json())
  .then(data => {
    <!-- Extract the latest readings from the response -->
    const latestReadingField1 = data.feeds[0].field1;
    const latestReadingField2 = data.feeds[0].field2;

    <!-- Display the latest readings on the HTML page -->
    document.getElementById('field1').textContent = latestReadingField1
        ;
    document.getElementById('field2').textContent = latestReadingField2
        ;

    <!-- Check Time -->
    const tempimply = document.getElementById('tempimply');
    const currentTime = new Date();
    const currentHour = currentTime.getHours();

    <!-- Conditions, converted F to C -->
    if ((parseFloat(latestReadingField1) < 18.333333) && ((currentHour
       >= 18 && currentHour <= 24) || currentHour < 6)) {
      tempimply.innerHTML = '<span style="color: #5370fd;"><strong>THE
         TEMPERATURE IS TOO LOW!</strong></span>' }
    else if ((parseFloat(latestReadingField1) > 18.333333) && ((
       currentHour >= 18 && currentHour <= 24) || currentHour < 6)) {
      tempimply.innerHTML = '<span style="color: #f40600;"><strong>THE
         TEMPERATURE IS TOO HIGH!</strong></span>' }
    else if ((parseFloat(latestReadingField1) == 18.333333) && ((
       currentHour >= 18 && currentHour <= 24) || currentHour < 6)) {
      tempimply.innerHTML = '<span style="color: black;"><strong>The
```

```
        temperature is just right.</strong></span>' }
    else if ((parseFloat(latestReadingField1) < 23.888889) && (
       currentHour >= 6 && currentHour < 18)) {
      tempimply.innerHTML = '<span style="color: #5370fd;"><strong>THE
        TEMPERATURE IS TOO LOW!</strong></span>' }
    else if ((parseFloat(latestReadingField1) > 23.888889) && (
       currentHour >= 6 && currentHour < 18)) {
      tempimply.innerHTML = '<span style="color: #f40600;"><strong>THE
        TEMPERATURE IS TOO HIGH!</strong></span>' }
    else if ((parseFloat(latestReadingField1) == 23.888889) && (
       currentHour >= 6 && currentHour < 18)) {
      tempimply.innerHTML = '<span style="color: black;"><strong>The
        temperature is just right.</strong></span>' }

    <!-- Conditions2 -->
    if (parseFloat(latestReadingField2) < 25) {
      humidimply.innerHTML = '<span style="color: #5370fd;"><strong>THE
         HUMIDITY IS TOO LOW!</strong></span>' }
    else if (parseFloat(latestReadingField2) > 49) {
      humidimply.innerHTML = '<span style="color: #f40600;"><strong>THE
         HUMIDITY IS TOO HIGH!</strong></span>' }
    else {
      humidimply.innerHTML = '<span style="color: black;"><strong>The
        humidity is just right.</strong></span>' }

  })
  .catch(error => console.error(error));
</script>


<br><br>
<style>
hr {color: white; border-style: solid;}
```

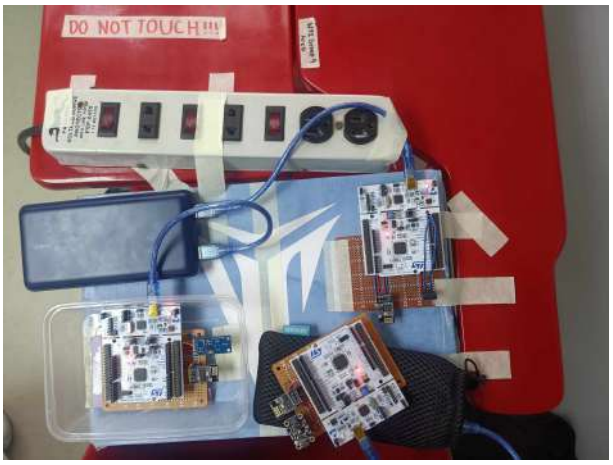
```
</style>
<hr>
<br><br>
</p>


<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Air Quality Sensor (SGP30) Readings</b> <br><br>

<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
    ="https://thingspeak.com/channels/2203260/charts/1?bgcolor=%23ffffff
    &color=%23508C46&dynamic=true&results=60&title=CO2+Reading&type=line
    &yaxis=CO2+[ppm]"></iframe>
<iframe width="450" height="260" style="border: 1px solid #cccccc;" src
    ="https://thingspeak.com/channels/2203260/charts/2?bgcolor=%23ffffff
    &color=%23508C46&dynamic=true&results=60&title=TVOC+Reading&type=
    line&yaxis=TVOC+[ppb]"></iframe>
<br><br>
</p>

<p style="font-family: arial; font-size:150%; text-align: center;">
<b>Air Quality Sensor (SGP30) Indicators</b> <br>


Current CO2 Level: <span id="field3"></span> ppm <br> <span id="
    co2imply"></span><br>
Current TVOC Level: <span id="field4"></span> ppb <br>
</p>

<script>
<!-- Make an HTTP GET request to ThingSpeak API -->
fetch('https://api.thingspeak.com/channels/2203260/feeds.json?results
    =1')
```

```
.then(response => response.json())
.then(data => {
  <!-- Extract the latest readings from the response -->
  const latestReadingField1b = data.feeds[0].field1;
  const latestReadingField2b = data.feeds[0].field2;

  <!-- Display the latest readings on the HTML page -->
  document.getElementById('field3').textContent =
     latestReadingField1b;
  document.getElementById('field4').textContent =
     latestReadingField2b;

  <!-- Check Time -->
  const co2imply = document.getElementById('co2imply');
  const currentTime = new Date();
  const currentHour = currentTime.getHours();

  <!-- Conditions2 -->
  if (parseFloat(latestReadingField1b) < 1000) {
    co2imply.innerHTML = '<span style="color: #338a4a;"><strong>
       Normal CO2 levels.</strong></span>' }
  else if (parseFloat(latestReadingField1) < 2000) {
    co2imply.innerHTML = '<span style="color: #ffffff;"><strong>
       Drowsiness might be experienced.</strong></span>' }
  else if (parseFloat(latestReadingField1) < 5000) {
    co2imply.innerHTML = '<span style="color: #fdd553;"><strong>CO2
       LEVELS ARE TOO HIGH!.</strong></span>' }
  else if (parseFloat(latestReadingField1) < 40000) {
    co2imply.innerHTML = '<span style="color: #fd5353;"><strong>CO2
       LEVELS ARE TOXIC!.</strong></span>' }
  else {
    co2imply.innerHTML = '<span style="color: #d353fd;"><strong>
       OXYGEN DEPRIVATION MAY OCCUR!.</strong></span>' }
```

```
  })
  .catch(error => console.error(error));
</script>
<br>

<br><br>
<style>
hr {color: white; border-style: solid;}
</style>
<hr>
<br><br>
</p>

<p style="font-family: arial; font-size: 120%; text-align: center;">
<br>
<strong>About:</strong> An indoor plant monitoring application that
    tracks plant health, state solutions, and measure effects on close
    environment. Used AM2320, BH1750, and SGP30 sensors, ESP8266 Wi-Fi
    Modules, STM32F411RE as master, and Thingspeak as server. Chose <i>
    Epipremnum aureum</i> (Golden Pothos) as it has the most positive
    effects on indoor environment.
<br><br>
(c) John Danielle Castor (jtcastor1@up.edu.ph), Airick Miguel Reyes-
    Gonzales (agonzales2@up.edu.ph), Zylm M. Sabater (zmsabater@up.edu.
    ph); June 25, 2023<br>
<br><br>
</p>


</body>
</html>
```

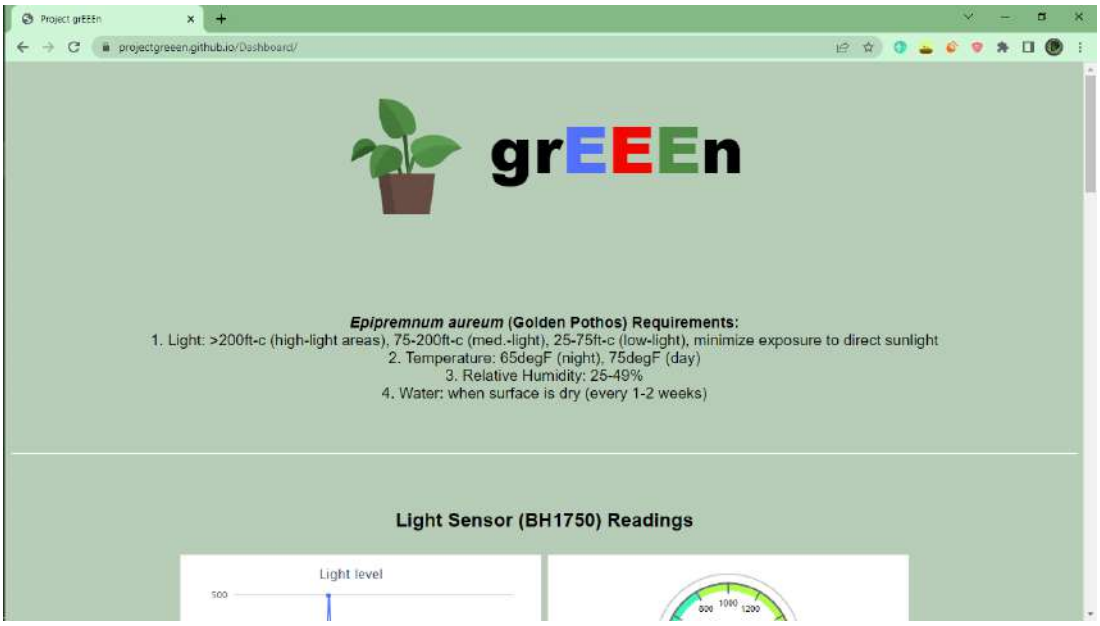
Appendix D

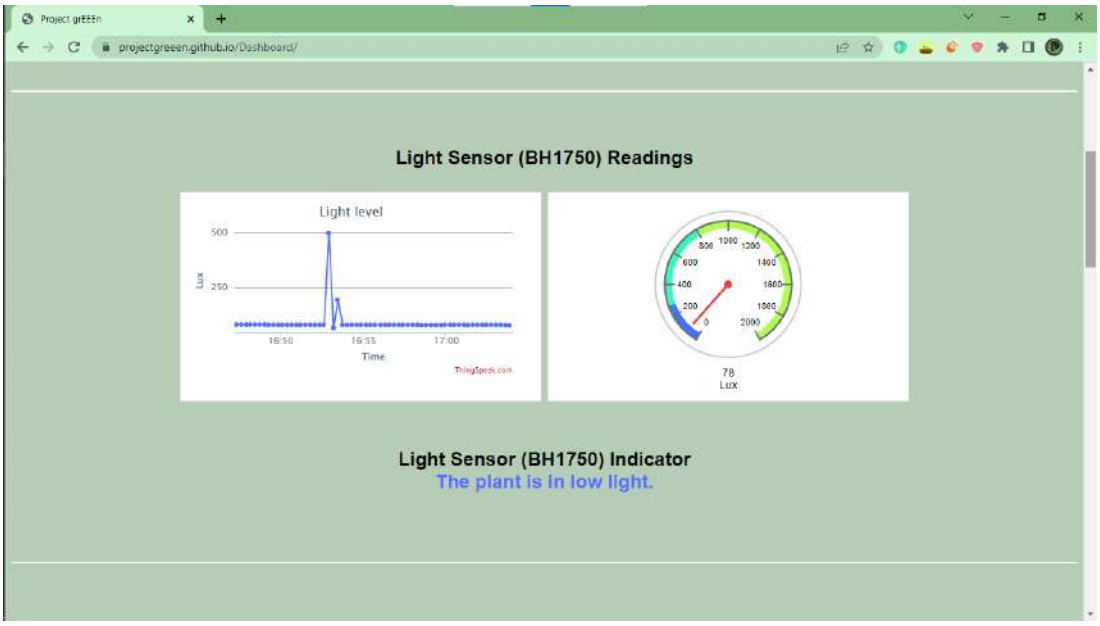
Deployment Setup

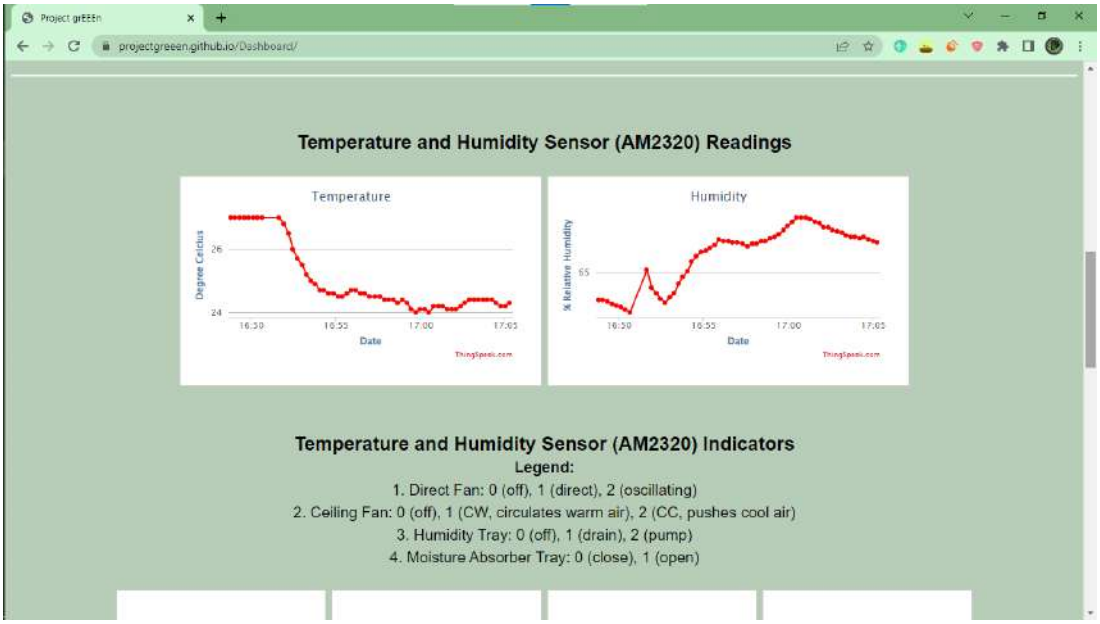


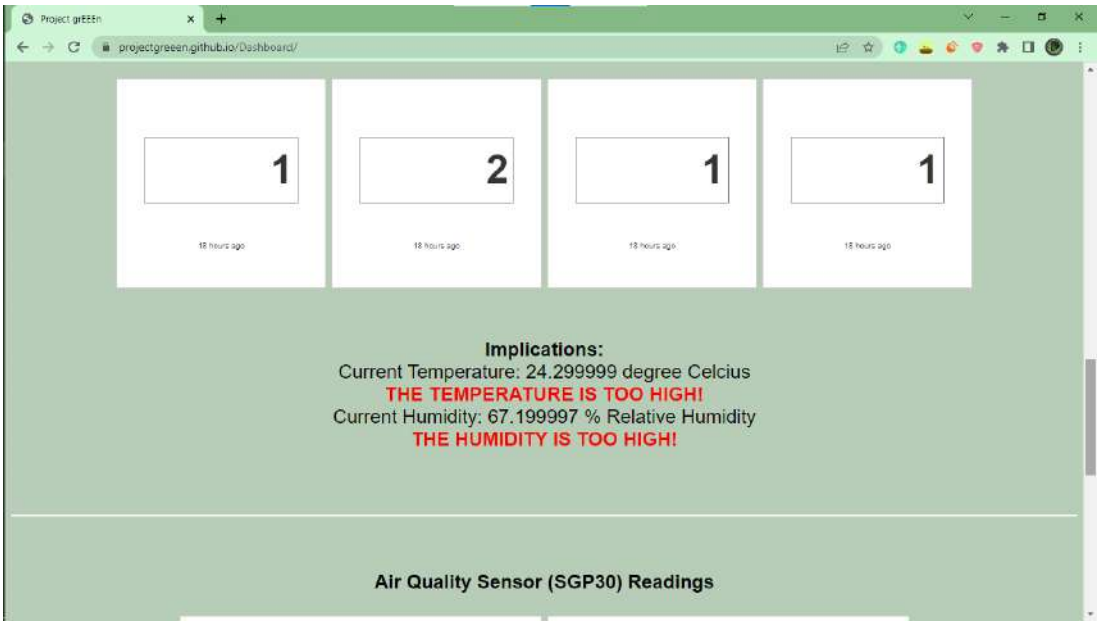
Appendix E

Web Interface

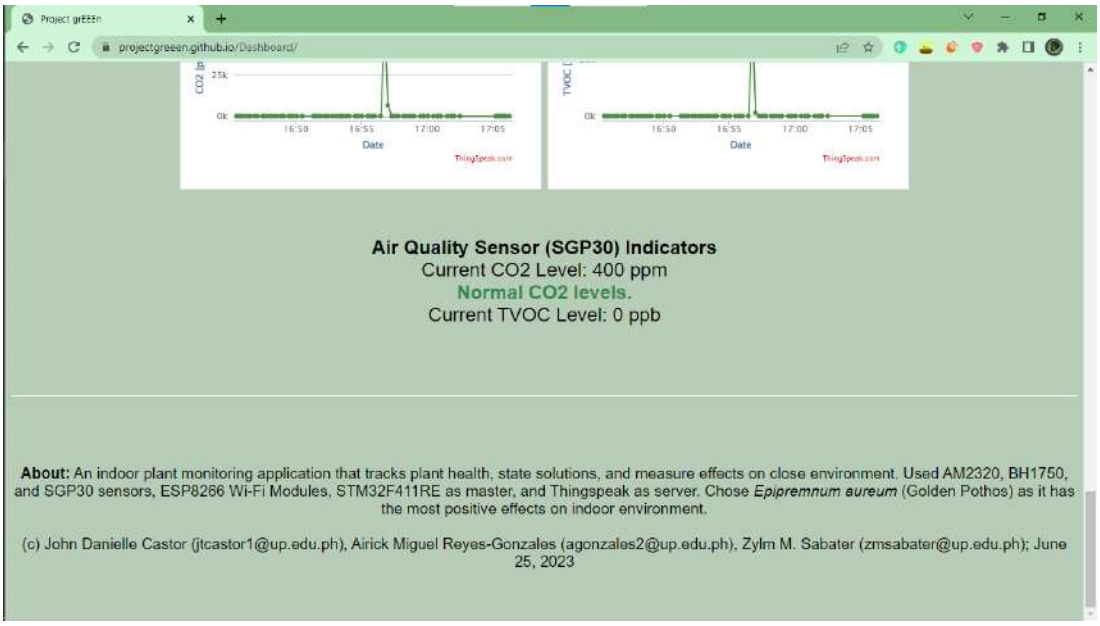



projectgreeen.github.io/Dashboard/
Light Sensor (BH1750) Readings
Light level
Lux
Time
78
Lux
Light Sensor (BH1750) Indicator
The plant is in low light.


projectgreeen.github.io/Dashboard/
Temperature and Humidity Sensor (AM2320) Readings
Temperature
Degree Celcius
Date
Humidity
% Relative Humidity
Date
Temperature and Humidity Sensor (AM2320) Indicators
Legend:
1. Direct Fan: 0 (off), 1 (direct), 2 (oscillating)
2. Ceiling Fan: 0 (off), 1 (CW, circulates warm air), 2 (CC, pushes cool air)
3. Humidity Tray: 0 (off), 1 (drain), 2 (pump)
4. Moisture Absorber Tray: 0 (close), 1 (open)


Project greeen
projectgreeen.github.io/Dashboard/
1
2
1
1
18 hours ago
18 hours ago
18 hours ago
18 hours ago
Implications:
Current Temperature: 24.299999 degree Celcius
THE TEMPERATURE IS TOO HIGH!
Current Humidity: 67.199997 % Relative Humidity
THE HUMIDITY IS TOO HIGH!
Air Quality Sensor (SGP30) Readings


Project greeen
projectgreeen.github.io/Dashboard/
Air Quality Sensor (SGP30) Readings
CO2 Reading
CO2 [ppm]
50k
25k
0k
16:50
16:55
17:00
17:05
Date
ThingSpeak.com
TVOC Reading
TVOC [ppb]
20k
0k
16:50
16:55
17:00
17:05
Date
ThingSpeak.com
Air Quality Sensor (SGP30) Indicators
Current CO2 Level: 400 ppm
Normal CO2 levels.
Current TVOC Level: 0 ppb


Project grEEEn
projectgreeen.github.io/Dashboard/
CO2 (ppm)
25k
0k
16:50
16:55
17:00
17:05
Date
ThingSpeak.com
TVOC (ppb)
0k
16:50
16:55
17:00
17:05
Date
ThingSpeak.com
Air Quality Sensor (SGP30) Indicators
Current CO2 Level: 400 ppm
Normal CO2 levels.
Current TVOC Level: 0 ppb
About: An indoor plant monitoring application that tracks plant health, state solutions, and measure effects on close environment. Used AM2320, BH1750, and SGP30 sensors, ESP8266 Wi-Fi Modules, STM32F411RE as master, and Thingspeak as server. Chose Epipremnum aureum (Golden Pothos) as it has the most positive effects on indoor environment.
(c) John Danielle Castor (jtcastor1@up.edu.ph), Airick Miguel Reyes-Gonzales (agonzales2@up.edu.ph), Zylm M. Sabater (zmsabater@up.edu.ph); June 25, 2023