ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

AUTOMATED SENSOR-BASED FILTRATION SYSTEM

Team No. 68

Omar Koueider (oyk2@illinois.edu)

Prithvi Saravanan (prithvi3@illinois.edu)

Karthik Talluri (talluri4@illinois.edu)

TA: Selva Subramaniam

Professor: Arne Fliflet

May 3, 2023

Abstract

The final report provides insight into how we defined a massive environmental problem, our electronics-based solution to this problem, and the implementation of this system as our project for ECE 445. In this paper, we will provide various descriptions of elements of our project, specifically system features, design, cost, and benefits.

Contents

1. Introduction 1
  1.1 Problem 1
  1.2 Solution 1
  1.3 Visual Aid 1
  1.4 High-Level Requirements 2
2 Design 3
  2.1 Block Diagram 3
  2.2 Data Acquisition Subsystem 3
    2.2.1 Overview 3
    2.2.2 Requirements 4
    2.2.3 Design Decisions 4
    2.2.4 Data Acquisition Subsystem RV Table 7
    2.2.5 Data Acquisition Testing 7
  2.3 Microcontroller Subsystem 8
    2.3.1 Overview 8
    2.3.2 Requirements 9
    2.3.3 Design Decisions 9
    2.3.4 Microcontroller Subsystem RV Table 10
  2.4 Dynamic Filtration Subsystem 11
    2.4.1 Overview 11
    2.4.2 Requirements 11
    2.4.3 Design Decisions 11
    2.4.4 Dynamic Filtration Subsystem RV Table 12
    2.4.5 Dynamic Filtration Subsystem Testing 12
  2.5 Tolerance Analysis 13
  2.6 PCB Design 14
3. Conclusion 16
  3.1 Results and Accomplishments 16
  3.2 Uncertainties 17
  3.3 Future Considerations 17
Appendix A Ethics and Safety 18
  A.1 Ethics 18
  A.2 Safety 18
    A.2.1 PCB Design 18
    A.2.2 Usage of Dust 18
    A.2.3 Usage of Blowers 18
References 22

1. Introduction

We will provide context, information, and a detailed analysis of the problem the automated sensor-based filtration system will address and how we contextually built a solution to solve this problem.

1.1 Problem

As our environment continues to change for the worse with the presence of global warming and increased human consumption of resources like fossil fuels, the safety associated with breathing clean air is being threatened. In metropolitan areas worldwide, there is an increase in smog and toxic output, leading to increased respiratory problems. In areas such as Delhi and Beijing, it is impossible to venture outside on certain mornings due to air quality and dangerous toxins [1]. No building filtration systems adapt according to compounds outside the building, like volatile organic compounds (VOCs). As a result, implementing a new, different filtration system becomes necessary. This ever-present problem will continue as the world's population increases, and breathing clean air indoors is a fundamental right everyone should have.

1.2 Solution

The solution to this vast and unending problem is a dynamic filtration system that adjusts according to the concentration of a specific outdoor particle. We have chosen to monitor CO_2 and PM2.5 particles commonly found in dust. The goal was to keep the indoor particle concentrations despite any change in the composition of the outside air. This was accomplished using a sensor subsystem, an ESP32 microcontroller, and an air blower for filtration testing. The electrochemical sensor system constantly monitors these factors and provides a reading that will activate the dynamic filtration subsystem to filter out particles more accurately. To keep the indoor concentration numbers constant, we constantly compared data from the outdoor particulate sensor system with one based inside the enclosure. Two separate electron chemical sensor systems monitor outdoor and indoor particles, and a microcontroller takes the data from these sensors and determines what particles to filter out. The adaptation functionality of changing the directional flow of an external air source is implemented according to the results from the data acquisition subsystem and the responses of the microcontroller subsystem.

1.3 Visual Aid

We have provided a view of the entire system in Figure 2. Two enclosures, one with dirt particles and one without them, represent outdoors and indoors. Each of these has a pair of sensors that allow a user to visualize the data and changes in air filtration accordingly. The client can then see the difference in air quality in the dirty enclosure and the clean enclosure based on the dirt filtered from the dynamic filtration mechanism. Found below is a CAD design of the expected mechanical setup (Figure 1) as well as the actual setup we were able to demo (Figure 2).

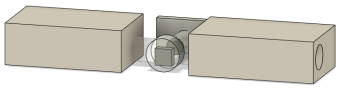


Figure 1: CAD design of mechanical setup, generated using AutoFusion 360

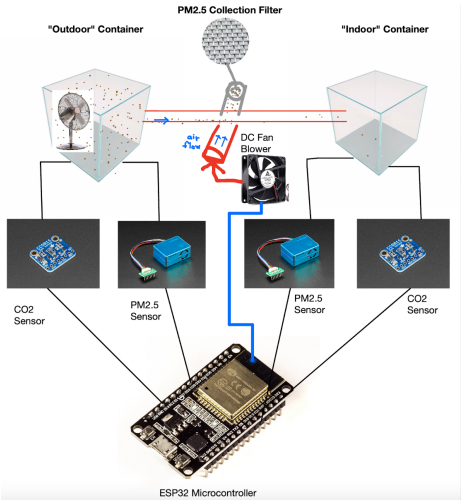


Figure 2: Automated Sensor-Based Filtration System Overview

Above is our design for the integration of the mechanical and electrical components. This diagram represents the high-level requirements and how a user can visualize the benefits of having this type of system. By viewing the two layers of filtration, we are able to show how the dirty air will not reach the clean container very easily and would increase safety and efficiency for use.

1.4 High-Level Requirements

The following needed to be met during our demonstration of the system:

- The concentration of PM2.5 in the “indoor” enclosure should be lower than that of the “outdoor” by approximately 75-80%.
- The concentration of CO_2 determines whether airflow will speed up or slow down based on circulation.
- To maintain power efficiency, the dynamic filtration mechanism starts running when the PM2.5 or CO_2 particles reach a certain level.

2 Design

2.1 Block Diagram

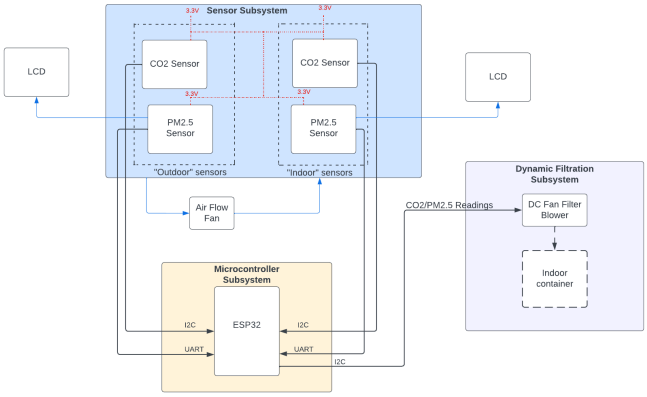


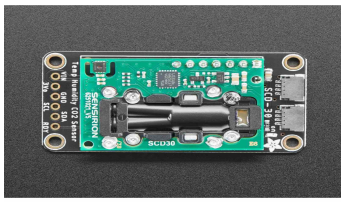
Figure 3: Block Diagram Subsystems

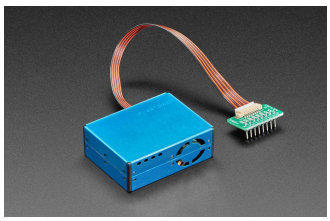
Our design consists of three subsystems, with the sensor and filtration subsystems housed within the enclosure of two containers joined by an intermediary tube. The ESP32 microcontroller is located outside, hooked up to all the sensors inside the enclosure.

2.2 Data Acquisition Subsystem

2.2.1 Overview

The goal of the data acquisition subsystem is to feed real-time data accurately to the overall system to provide essential information for the correct implementation of the entire project. It consists of an array of electrochemical sensors that receive data from the enclosures' concentrations of PM2.5 particles and carbon compounds and transmit that data to the microcontroller subsystem. Specifically, we utilized the PMSA003I PM2.5 sensor and the SCD30 VOC and CO_2 sensors, where each enclosure section contains one. These are essential for our high-level requirements since they allow us to detect dust particles as well as CO_2 concentrations, therefore enabling our dynamic filtration system to filter air more efficiently. We also used the sensors to compare dust/CO_2 levels in the contaminated environment with those in the "clean" environment to provide us with information concerning the success of our filtration techniques. All four sensors (2 of each) pass on data for analysis to the microcontroller subsystem as they are all hooked up to the same breadboard and communicate via I^2C protocol. Finally, it was vital for us to actually display the data from the sensors so that the user can confirm the success of the filtration technique without having to slow down data collection to check the output on the computer. Therefore, our results were displayed on two LCD displays, one for the clean environment and one for the dirty.





Figures 4 and 5: SCD30 VOC and CO_2 Sensor (left). PMSA003I PM2.5 Sensor (right).

2.2.2 Requirements

The requirements for this subsystem are as follows:

- There is a constant supply of 3.3 V to power both the CO_2 and PM2.5 sensors.
- The PM2.5 and CO_2 sensors provide accurate data and are not reading incorrectly.
- There is a communication protocol between the PM2.5 and CO_2 sensors using the microcontroller.
- If sensors stop reading data, the user must be informed. If the display fails, then the Arduino IDE can print a data stream onto the terminal window.

These requirements and their corresponding verifications are detailed in Table 1.

2.2.3 Design Decisions

I²C Communication:

To allow for communication between the microcontroller and the sensors, we needed to utilize I²C protocol and write the necessary code. I²C protocol is a two-wire communication protocol with two ports: SDA and SCL. The SCL port is used to feed a clock signal, which allows for synchronous transfer and retrieval of data between the slave device and the master [2]. The SDA port is used for the transfer of actual data [2]. It is first utilized by the master to transfer a device address and a registered address to write to, typically for initial configuration. Then, the slave device utilizes SDA to send the master device sensor data. This information can then be reformatted and printed to give us a numerical output coming from the sensor. To achieve all of the above, there is an I²C structure that we must abide by so that we are providing the sensor with the correct clock signal and the correct data at the right time. Any slight delay or error in our code could result in junk data being read from the wrong register due to improper configuration or reading. A figure of the I²C protocol has been provided below:

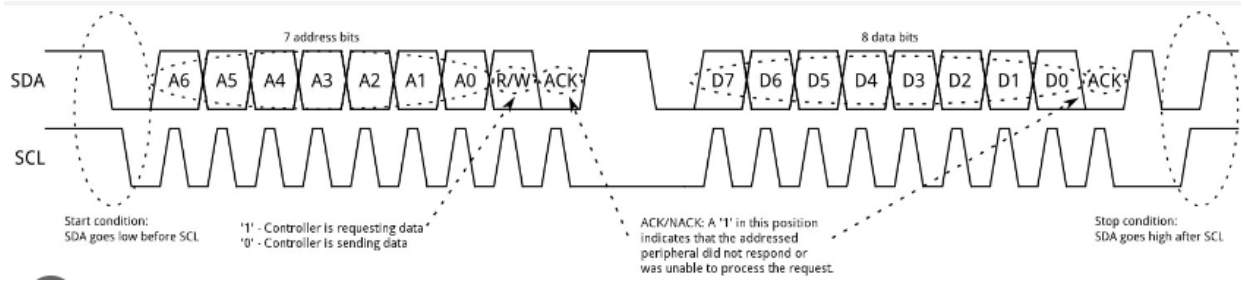
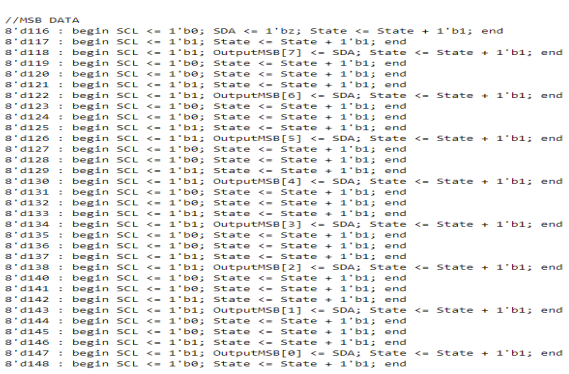
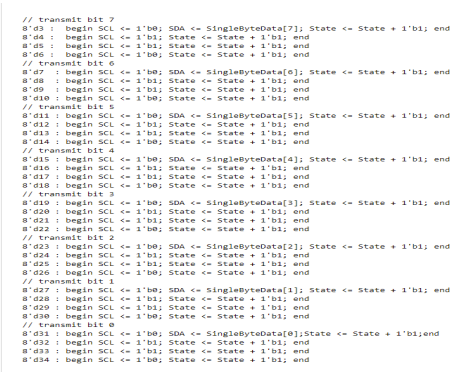


Figure 6: I²C Protocol Structure

As we can see, there are important sequences that we must incorporate to make communication work, such as the start condition, the end condition, and the acknowledgment [3]. We accomplished all these sequences using a state machine where each state performed a certain act on SCL and SDA. We developed Verilog code to achieve this and have included snippets below:



```
//MSB DATA
8'd116 : begin SCL <= 1'b0; SDA <= 1'bz; State <= State + 1'b1; end
8'd117 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd118 : begin SCL <= 1'b1; OutputMSB[7] <= SDA; State <= State + 1'b1; end
8'd119 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd120 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd121 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd122 : begin SCL <= 1'b1; OutputMSB[6] <= SDA; State <= State + 1'b1; end
8'd123 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd124 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd125 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd126 : begin SCL <= 1'b1; OutputMSB[5] <= SDA; State <= State + 1'b1; end
8'd127 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd128 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd129 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd130 : begin SCL <= 1'b1; OutputMSB[4] <= SDA; State <= State + 1'b1; end
8'd131 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd132 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd133 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd134 : begin SCL <= 1'b1; OutputMSB[3] <= SDA; State <= State + 1'b1; end
8'd135 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd136 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd137 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd138 : begin SCL <= 1'b1; OutputMSB[2] <= SDA; State <= State + 1'b1; end
8'd140 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd141 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd142 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd143 : begin SCL <= 1'b1; OutputMSB[1] <= SDA; State <= State + 1'b1; end
8'd144 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd145 : begin SCL <= 1'b0; State <= State + 1'b1; end
8'd146 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd147 : begin SCL <= 1'b1; OutputMSB[0] <= SDA; State <= State + 1'b1; end
8'd148 : begin SCL <= 1'b0; State <= State + 1'b1; end
```



```
// transmit bit 7
8'd3 :  begin SCL <= 1'b0; SDA <= SingleByteData[7]; State <= State + 1'b1; end
8'd4 :  begin SCL <= 1'b1; State <= State + 1'b1; end
8'd5 :  begin SCL <= 1'b1; State <= State + 1'b1; end
8'd6 :  begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 6
8'd7  : begin SCL <= 1'b0; SDA <= SingleByteData[6]; State <= State + 1'b1; end
8'd8  : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd9  : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd10 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 5
8'd11 : begin SCL <= 1'b0; SDA <= SingleByteData[5]; State <= State + 1'b1; end
8'd12 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd13 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd14 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 4
8'd15 : begin SCL <= 1'b0; SDA <= SingleByteData[4]; State <= State + 1'b1; end
8'd16 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd17 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd18 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 3
8'd19 : begin SCL <= 1'b0; SDA <= SingleByteData[3]; State <= State + 1'b1; end
8'd20 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd21 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd22 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 2
8'd23 : begin SCL <= 1'b0; SDA <= SingleByteData[2]; State <= State + 1'b1; end
8'd24 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd25 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd26 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 1
8'd27 : begin SCL <= 1'b0; SDA <= SingleByteData[1]; State <= State + 1'b1; end
8'd28 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd29 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd30 : begin SCL <= 1'b0; State <= State + 1'b1; end
// transmit bit 0
8'd31 : begin SCL <= 1'b0; SDA <= SingleByteData[0];State <= State + 1'b1;end
8'd32 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd33 : begin SCL <= 1'b1; State <= State + 1'b1; end
8'd34 : begin SCL <= 1'b0; State <= State + 1'b1; end
```

Figures 7 and 8: Snippets of code that represents writing data onto the slave device using the SDA port and how we retrieve data from SDA. SingleByteData here is the register address we are passing onto the slave device.

Physical Setup:

Some of the design decisions we had to make included how we were going to hook up the sensors to our breadboard when there was a big distance between the breadboard and our boxes. When we attempted to wire everything together, we found the sensors could not reach the breadboard. Therefore, we used cable extenders for our PM2.5 sensors. We were not able to do the same for the CO_2 sensors as the sensor is on top of the pins that are meant to connect to the board and eventually to the GPIO pin of the microcontroller. This led us to purchase mini breadboards for us to attach the CO_2 sensors, which we would then connect to the main breadboard using ordinary wires.

In terms of the placement of the sensors, we attached the CO_2 sensors to the side of the boxes so that they were hidden and looked neat. These are then connected to mini breadboards, which are wired to the main breadboard. We taped the PM2.5 sensors on the bottom of the boxes and in front of the large fan/connecting tube since they were not that sensitive and we wanted to make sure the increase in readings was instantaneous and significant for our demo. Below are pictures of how both sensors were attached to our system.

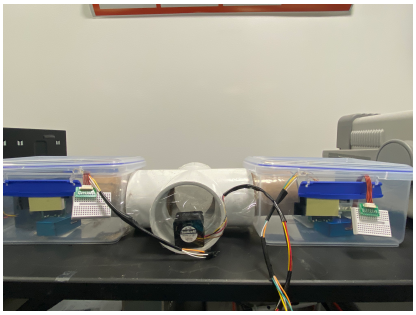


Figure 9: Mechanical Design of the Automated Sensor-Based Filtration System

LCD Displays:

We included LCD displays as part of our data acquisition system so that it is easier for the user to assess the effectiveness of our filtration system. These LCDs are attached to the microcontroller through GPIO pins. They were then configured according to the following code:

```
void front_lcd_callback()
{
 char buf_line1[16];
 char buf_line2[16];

 Serial.println("Front LCD");
 front_lcd.clear();
 front_lcd.setCursor(0, 0);
 sprintf(buf_line1, "C:%4d.%02d F:%4d", front_co2_whole, front_co2_decimal, duty_cycle);
 front_lcd.print(buf_line1);
 front_lcd.setCursor(0,1);
 sprintf(buf_line2, "PM:%4d %4d", frontpm25_data.pm25_standard, frontpm25_data.pm10_standard);
 front_lcd.print(buf_line2);
 }
```

Figure 10: Front LCD display code

```
void back_lcd_callback()
{
 char buf_line1[16];
 char buf_line2[16];

 Serial.println("Back LCD");

 back_lcd.clear();
 back_lcd.setCursor(0, 0);
 sprintf(buf_line1, "C:%4d.%02d", back_co2_whole, back_co2_decimal);
 back_lcd.print(buf_line1);
 back_lcd.setCursor(0, 1);
 sprintf(buf_line2, "PM:%4d %4d", backpm25_data.pm25_standard, backpm25_data.pm10_standard);
 back_lcd.print(buf_line2);
}
```

Figure 11: Back LCD display code

Data from the sensors is stored in variables and we distinguished between the clean and dirty environments by naming them front and back. We simply used the commands sprintf, then print to print onto the back LCD and the front LCD. Below is a picture of the output on the LCDs.

Figure 12: LCDs in use during demonstration

2.2.4 Data Acquisition Subsystem RV Table

Requirements	Verification
• The necessary 3.3V to power both CO_2 and PM2.5 sensors are constantly received regardless of sensor status.	• Power supply must not be touched and should be capable of powering up sensors for long periods of time.
• PM2.5 and CO_2 sensors give accurate data and are not reading junk	• I^2C protocol will be tested on all sensors first through an Arduino file to ensure the data being read is accurate consistently and constantly changes. • Different modes and accuracies will be written onto the sensor via I^2C to test the sensor and pick the most efficient environment. • Sensors will be placed in an area of high dust and CO_2 concentrations and significant changes in readings must be achieved.
• Communication between PM2.5 and CO_2 sensors with microcontroller	• Ensure microcontroller and sensors are connected to the PCB board or breadboard. Further verification will be handled in the microcontroller subsystem section.
• If sensors stop reading data, the user must be informed. If the display fails, the Arduino IDE can print a sensor data stream on the terminal window.	• Interface will be implemented to display the data from the sensors. This will allow us to spot if data is inaccurate or if no data is being read at all due to glitches.

2.2.5 Data Acquisition Testing

PM2.5 Sensor:

To test the PM2.5 sensor, we hooked it up directly to the GPIO pins of the microcontroller. This allowed us to analyze the results on the Arduino IDE terminal. We added a check to ensure the data being read was not junk, which would be called checksum failure. Unfortunately, data was being accurately detected and displayed, but almost a minute later, we would find a checksum failure and read extraordinarily high values. This would sometimes be displayed as junk letters when connected to the LCD. After debugging our code, we found that the addition of delays when outputting the results was causing the checksum failure. This is because it was affecting the timing sequence necessary for UART communication. We had initially inserted delays since data was being collected too quickly for us to be able to see on the terminal. After fixing this small issue, we found no further problems with our PM2.5 sensors.

CO_2 Sensor:

The CO_2 sensor was much easier to test than the PM2.5 since we found no errors or junk data being read. However, we faced many problems once we had to connect both CO2 sensors. We found the sensors' I^2C communication troublesome because our microcontroller only allowed one of the sensors to be hooked up to the I^2C port (address 0x61). Additionally, the code we were using to allow for I^2C communication stopped working when two SCD30 sensors were introduced, it only worked with one. For our code, the modification was quite simple as we duplicated it and allowed for more SDA and SCL ports. We needed to be more creative to make our microcontroller support both sensors simultaneously. We devised the idea to modify two GPIO pins to turn them into an SDA and an SCL. We accomplished this through the Arduino IDE using an external library.

Displays:

We faced a similar issue with the displays as we did with the CO_2 sensors in the sense that each LCD worked alone but combining them resulted in issues. The LCDs come with a default I^2C address of 0x27. However, we needed to use two of them, and they could not share the same address. Fortunately, the LCDs come with address solder pads, which we shorted to change one of the LCD's addresses from 0x27 to 0x28.

2.3 Microcontroller Subsystem

2.3.1 Overview

The microcontroller subsystem consists of a single microcontroller that will communicate with both the data acquisition subsystem and the dynamic filtration subsystem to provide accurate instructions to the directional airflow about when to activate. Using an external power source with a measured voltage, this subsystem will be the core impetus for the functionality of this entire project. In the final implementation of the microcontroller subsystem, we implemented the ESP32-WROOM-D, as that was the most accessible of the microcontrollers available with the functionalities that we required.

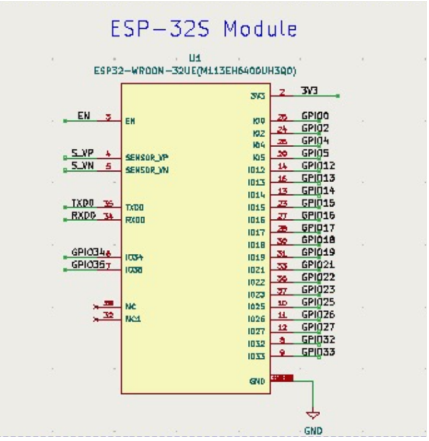


Figure 13: Schematic for the ESP-32 Microcontroller

2.3.2 Requirements

The requirements for this subsystem are as follows:

- The necessary 3.3V to power the ESP32 has constantly received whether the system is on or off.
- The ESP32 microcontroller will receive data from the CO_2 and PM2.5 sensors and communicate with the sensor controlling the amount of air received from the external blower.
- The system should adjust based on the numerical readings processed and returned by the ESP32.

2.3.3 Design Decisions

This is the ESP-32 microcontroller, the crux of our microcontroller subsystem. The microcontroller allowed us to link the real-time sensor data from the data acquisition subsystem to the dynamic filtration mechanism. By assigning various footprint elements to the respective sensors' data transmission, we wrote Arduino code that allowed for seamless communication between the various elements of the overall system. Each sensor was provided a specific GPIO pin for the respective UART, I^2C, TX, and RX connections.

```
#define GPIO_RELAY_PIN        27 // ESP32 GIOP27 - connects to the IN pin of relay
#define GPIO_PWM_PIN          15 // GPIO to send PWM for Fan
#define GPIO_FRONT_PM25RX     16 // Serial port RX GPIO To front sensor PM2.5 TX
#define GPIO_FRONT_PM25TX     17 // Serial port TX GPIO To front sensor PM2.5 RX
#define GPIO_BACK_PM25RX      18 // Serial port RX GPIO To back sensor PM2.5 TX
#define GPIO_BACK_PM25TX      19 // Serial port TX GPIO To back sensor PM2.5 RX

#define GPIO_BACK_CO2SCL      14
#define GPIO_BACK_CO2SDA      12

#define FRONT_LCD_I2C_ADDR    0x27
#define BACK_LCD_I2C_ADDR     0x26
#define FRONT_SCD30_I2C_ADDR  0x61

#define BACK_SCD30_I2C_ADDR   0x62
#if 0
```

Figure 14: GPIO Pins Assigned to Sensors and Fan through Microcontroller

There are also various thresholds for dangerous PM2.5 and CO_2 concentrations defined for increased or decreased airflow. PM2.5 concentrations are considered unsafe above 35 $\mu g/m^3$ and extremely dangerous indoors when above 250 $\mu g/m^3$ [4]. According to the Department of Health, concentrations of CO_2 above 1400 ppm are considered unhealthy [5]. Therefore, we created a scaling system that allows for increased airflow as additional stimuli are introduced to the environment that causes either of these sensors to detect an increase in concentration.

```
#define PM25_MODERATE       250
#define PM25_UNHEALTHY      500
#define PM25_VERY_UNHEALTHY 750
#define PM25_HAZARDOUS      1000

#define PM10_MODERATE       500
#define PM10_UNHEALTHY      800
#define PM10_VERY_UNHEALTHY 1200
#define PM10_HAZARDOUS      1600

#define CO2_MODERATE        1400
#define CO2_UNHEALTHY       1700
#define CO2_VERY_UNHEALTHY  1900
#define CO2_HAZARDOUS       2200
#endif
```

Figure 15: Thresholds for each degree of air in the code

Each threshold is then used to determine the fan speed needed to ensure air circulation throughout the system.

2.3.4 Microcontroller Subsystem RV Table

Requirements	Verification
• The necessary 3.3V to power the ESP32 is constantly received whether the system is on or off	• The ESP32 should be continuously checking for particulate readings to ensure that the system is performing as instructed • We will measure the voltage of the subsystem by linking the device to a multimeter, which will produce the output. • The voltage source should be a constant provider to the ESP32 without any fail
• The ESP32 microcontroller will receive data from the CO_2 and PM2.5 sensors and communicate with the sensor that is controlling the amount of air received from the external blower	• Perform the experiment by starting the initial air flow that transfers the dust/CO_2 mixture from one container to the other. • The CO_2 and PM2.5 readings should be displayed when requested • Check whether the readings reflect the purpose of the project
• The system should adjust based on the numerical readings processed and returned by the ESP32.	• Create a program where the ESP32 receives and responds to the particulate readings • Provide contingencies on when the ESP32 should send signals to the system to turn on or off with thresholds of 1400 ppm of CO_2 or 35 μm/m^3.

2.4 Dynamic Filtration Subsystem

2.4.1 Overview

The Dynamic Filtration subsystem uses a programmable DC fan to filter out the PM2.5 particles. The ESP32 output signal containing CO_2 and PM2.5 data controls the fan's RPM. Based on the readings of the two concentration levels, the fan will use the concept of inertial impaction to filter out the PM2.5 dust into a separate pocket in the tube. The particles will be collected using a lint filter, and the filtered air with carbon dioxide will continue to flow along the pipe.

2.4.2 Requirements

The requirements for this subsystem are as follows:

- The DC fan must have enough RPM to redirect dust particles from the horizontal airflow to the collection pocket of the tube.
- The RPM/speed must adjust according to the PM2.5 and CO_2 concentrations detected by the sensors in the initial container, which are then transmitted to the Data Acquisition subsystem. The ESP32 must send the readings to the circuit controlling the DC fan speed.
- Fan should significantly reduce particulate matter and dangerous compounds between the initial and final enclosures.

2.4.3 Design Decisions

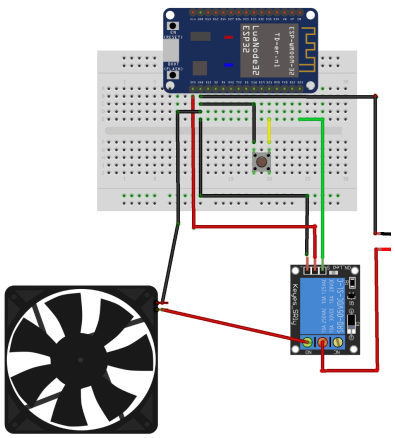


Figure 16: DC Fan Controller Circuit

As shown in the figure above, the circuit that controls the speed of the blower (DC fan) consists of a relay, the ESP32 microcontroller, a pushbutton, and the fan itself. The push button is mainly used to test if the speed control works and is correctly connected to the microcontroller. Once this is verified, the pushbutton will be removed from the breadboard. The fan requires a 5V DC voltage and is controlled using a PWM(pulse-width modulation) signal [6]. The relay is connected between an ESP32 pin and the fan. It acts as a switch that turns on/off depending on the signal sent from the ESP32. In our case, it must turn on once a particular PM2.5/CO_2 concentration is reached. This is done by flashing the C++ program to the microcontroller.

2.4.4 Dynamic Filtration Subsystem RV Table

Requirements	Verification
● The DC fan must have enough RPM to redirect dust particles from the horizontal airflow to the collection pocket of the tube.	● Experiment by starting the initial air flow that transfers the dust/CO_2 mixture from one container to another. ● During the transfer process, switch the filtering DC fan in the tube on, and test different RPM speeds. Observe which one successfully redirects the dust particles. ● Calculate the RPM by connecting the fan output to the oscilloscope. Observe any repetitive waveforms/spikes on the display to determine the period, then use it to calculate the frequency of rotation. ● Note down the minimum successful RPM and check if the dust concentration in the second container is reduced.
● The RPM/speed must adjust according to the PM2.5 and CO_2 concentrations detected by the sensors in the initial container, which are then transmitted to the Data Acquisition subsystem. The ESP32 must send the readings to the circuit controlling the DC fan speed.	● Create the fan speed controller circuit separately on a breadboard, using the ESP32, relay, and pushbutton. ● Program the ESP32 to switch the relay on once the pollutants reach a certain level. Then it must alternate between on/off as real-time sensor data is collected. ● Supply 6V to the microcontroller and relay. If the fan turns when the button is pressed, then the ESP32 has proper control over the fan.
● Fan should significantly reduce particulate matter and dangerous compounds between the initial and final enclosures.	● Take the output of both sensor subsystems in the two containers sent through the I^2C/UART. Hook the readings to 2 separate LCDs, one for each container. ● Compare the concentrations between the initial and final container and verify if the final is lower.

2.4.5 Dynamic Filtration Subsystem Testing

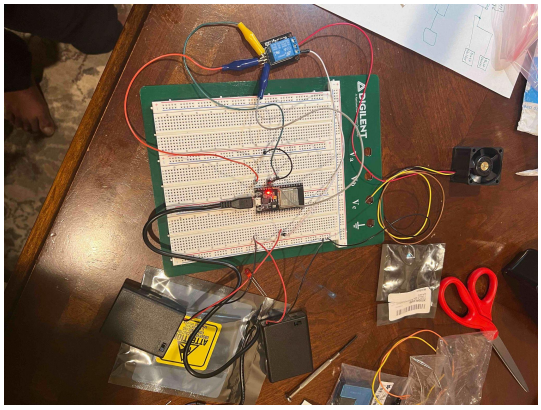


Figure 17: Dynamic Filtration Subsystem Testing with Relay, Microcontroller, and Fan

The figure above correlates to how we initially set up and tested the fan. The 12V battery can be seen on the bottom left side of the picture, which is hooked up to the relay at the top. Wires then exit the relay to go into the ESP32 and the fan. We did this to test the fan and if the relay adequately protected the ESP32 from the 12V high-voltage supply for the fan. We used a probe to ensure the relay was working and that only 5V was transferred to the microcontroller.

2.5 Tolerance Analysis

The most critical part of this project is the DC fan that functions as the programmable blower filtering the PM2.5 particles. The rated voltage is 12V, with the operation range being 4.5-13.8 VDC. The speed/airflow of the fan is proportional to the supplied voltage; if the fan VDC decreases, so does the RPM [5].

ITEM	DESCRIPTION
RATED VOLTAGE	12 VDC
OPERATION VOLTAGE	4.5 ~ 13.8 VDC
INPUT CURRENT	0.11 (MAX. 0.17) A
INPUT POWER	1.32 (MAX. 2.04) W
SPEED	8000 R.P.M. (REF.)
MAX. AIR FLOW (AT ZERO STATIC PRESSURE)	0.375 (MIN. 0.338) M^3/MIN. 13.24 (MIN. 11.94) CFM
MAX.AIR PRESSURE (AT ZERO AIR FLOW)	9.39 (MIN. 7.61) mmH_2O 0.370 (MIN. 0.300) $inchH_2O$
ACOUSTICAL NOISE (AVG.)	41.0 (MAX. 45.0) dB-A
INSULATION TYPE	UL: CLASS A

Figure 18: DC Fan Datasheet

According to the datasheet, the nominal speed at 12V is 8000 RPM. The tolerance level associated with this value is typically +-10%. The average peak current draw is 0.17 A, which can also be displayed as a periodic waveform of a specific frequency rather than a single value [5].

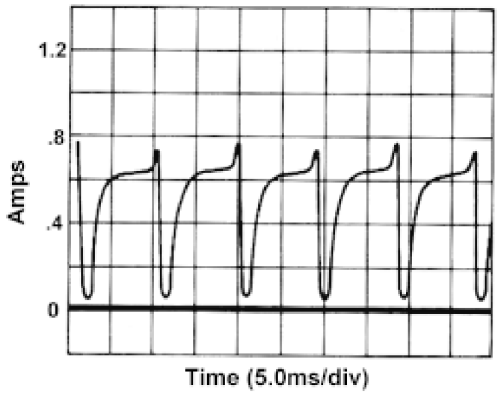


Figure 19: DC Fan Current Ripple

We can measure the running current using the true root mean square formula:

$$TRMS = (DC^2 + AC^2)^{1/2}$$

For measuring the current RPM of the fan, we need to monitor the DC ripple current. Using an oscilloscope, we can display the current waveform and determine the current graph's period. This will have some slight uncertainty, since the measurement will not be exact.

Then we determine RPM with: $f = \frac{1}{T}$

The applied voltage can never exceed the operation range 4.5-13.8 V, which means the RPM should be 8000 maximum.

2.6 PCB Design

Although we did not use the PCB, we put extensive effort into designing and attempting to implement the PCB. Our designs involved multiple elements that in the end did not work as intended.

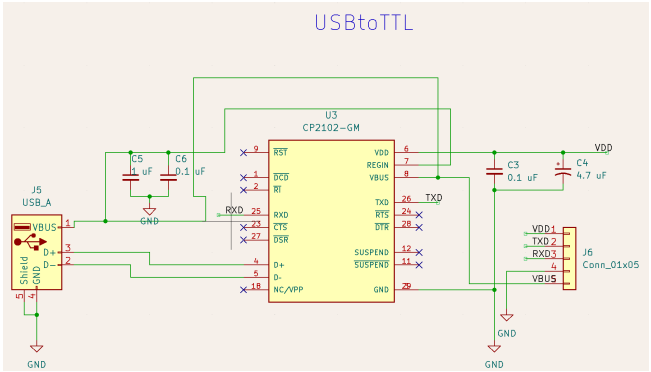
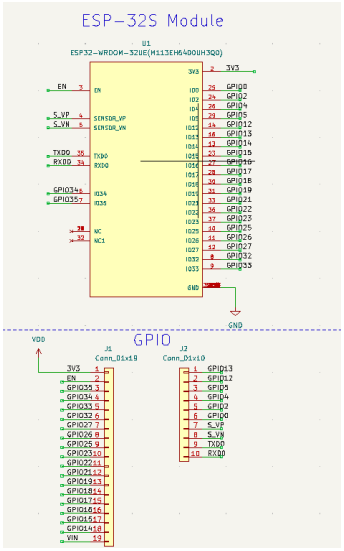
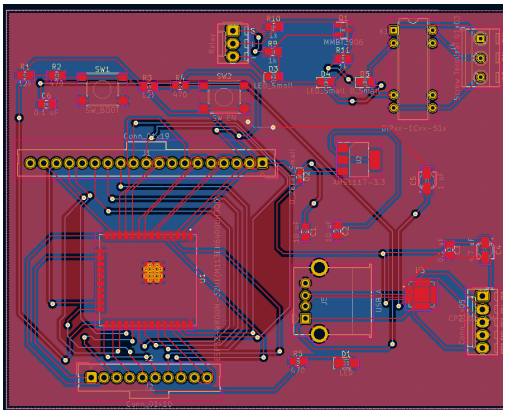


Figure 20: USB-to-TTL schematic in PCB Design

In the USB-to-TTL schematic, we can see that there are three components to the design, the USB plug that accepts the USB programmer, the USB chip, and the headers that connect to the rest of the circuit. When the user plugs in the USB programmer from the computer to the PCB, it should send the VBUS signal and the D+/D- signals to the USB chip, which then processes the TXD and RXD signals that

communicate with the sensors. The VBUS will then connect to the header pins that will communicate with the rest of the circuit.





Figures 21 and 22: ESP32 Microcontroller and PCB Design, respectively

The ESP32 microcontroller, specifically the ESP32-Wroom-UE, was the microcontroller that we decided to use because of the dedicated spots for UART, TX, and RX signals [7]. We also wanted to incorporate a pulse width modulation (PWM) reader, which is possible through this specific model. As a result, there was a linkage between the relay, acting as a regulator, and the microcontroller, allowing for the sensor data to provide input to the fan speed.

Figure 22 demonstrates our attempted PCB design to build the entire system. We had various components to our PCB design, but the key elements include the USBtoTTL scheme and the ESP32 module. These components are linked to the sensors that relay real-time data and the dynamically programmed fan.

3. Conclusion

3.1 Results and Accomplishments

Overall, we proved that our filtration system worked as PM2.5 levels did not increase significantly when dust was blown into the clean environment. We also detected dust sticking to our lint remover. Data was constantly collected and displayed on the LCDs with no failure or stoppages. Our microcontroller successfully utilized the I^2C protocol to communicate with the sensors and pass on that information to the filtration fan. Our filtration fan varied in speed according to data from both CO_2 and PM2.5 sensors and functioned correctly.

On demo day, we were challenged with many obstacles as everything seemed to go wrong. The relay we had implemented to stop the ESP32 from being exposed to high voltages stopped working. This resulted in our ESP32 microcontroller heating up and malfunctioning. When we attempted to reboot, the computer was not recognizing the ESP32, and we found no voltage at any of the pins when we probed it. Thankfully, we were given an ESP32 chip by another group who wasn't using it anymore and quickly remapped all the wires and GPIO pins with the new model. We consider this to be a major accomplishment.

To discuss our results in more detail, we generated a graph of how our filtration system responded to the data from the sensors. As shown below in Figure 23, the first parameter we tested was an increase in CO_2 readings (red plot). When this occurred, there was a massive jump in the fan's speed (blue plot). We then returned CO_2 to a normal state and increased PM2.5 as well as PM10 concentrations at the same time. This, yet again, increased fan speed. Our results prove that our dynamic filtration system changed according to different stimuli and could also consider multiple stimuli simultaneously.

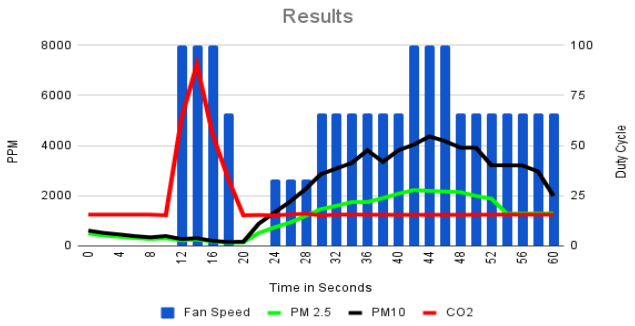
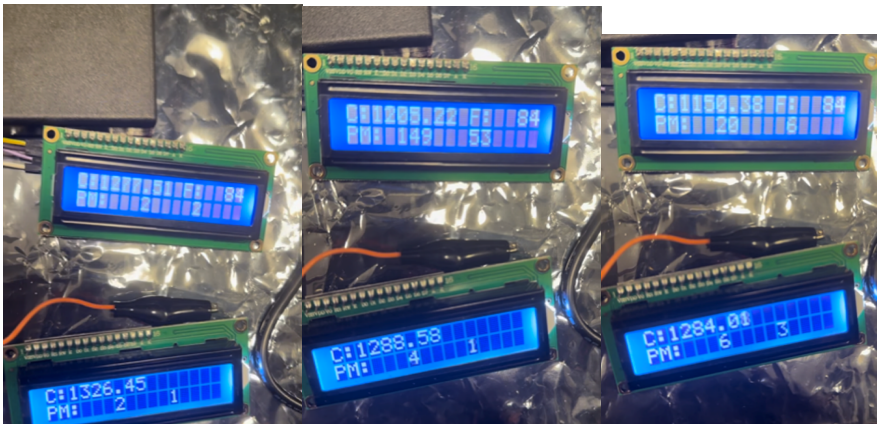


Figure 23: Results from testing the system

The second key experiment for analyzing the success of our system was observing what occurred to the clean environment when a massive amount of dust was introduced to the dirty environment. This experiment was conducted using a video of the LCD displays, which was about five minutes long. Below are screenshots of the first few seconds of the video, 10 seconds after the dust was introduced, and 5 minutes later. At the beginning of the experiment, as denoted by Figure 24, the PM2.5 of both the clean (bottom LCD) and dirty (top LCD) environments are 2 $\mu g/m^3$. When dust is blown into the system, we can see the PM2.5 of the dirty environment increase to 149 $\mu g/m^3$, while the clean

environment is still at a low 4 μg/m³. After we allow the dusty air to travel through the mailing tube and into the clean box, we can see the dirty environment's PM2.5 decreases due to the decrease in the dust (20 μg/m³). Still, the clean environment stays at a very low 6 μg/m³, considered very safe. The increase from 2 μg/m³ to only 6 μg/m³ despite the magnitude of dust introduced shows the success of our project.



Figures 24, 25, 26: LCDs as the demonstration continues, values climb and decline real-time

3.2 Uncertainties

- Scalability: While the proposed solution may work for a single building, it may not be scalable to larger areas. This is because monitoring outdoor particle concentrations in a broader area may require many sensors, which can be costly and difficult to manage.
- User acceptance: The dynamic filtration system proposed in the solution may be difficult for users to understand and operate, leading to user error or misuse. Additionally, users may not be willing to adapt to changes in the system, such as changes in airflow direction, which may limit its effectiveness.

3.3 Future Considerations

There are many different avenues that can be explored in building this project because our system is very adaptable. The first step is implementing dynamic filtration capabilities with the CO_2 sensor inside the indoor enclosure. Given the increased stale air in the environment, we would like to simulate the environment and how the filtration system would respond to that. The second step would be to include PM1.0 particles in the functionality, as the current sensors can detect these particles but give rather inaccurate results. By including a PM1.0 sensor, we could detect even the most minuscule and dangerous particles. We also would like to create an air current in the indoor enclosure to simulate additional air circulation to view the effects of increased CO_2 even with forced convection. This would allow us to look at the stale air and understand how to maximize filtration and airflow through the system.

Appendix A Ethics and Safety

A.1 Ethics

These experiments have been done following the IEEE Code of Ethics. This experiment involves many elements, especially experimental numbers, that may impact the system's success.

1. We seek to provide the most truthful information based on the results of our work. [8]

   We had multiple reviews with teaching assistants and amongst ourselves to ensure the system was performing as we intended. We understood that the data from the fans may be unpredictable and may not show what we are looking for concerning accuracy, so we ensured reporting real-time results from the sensors, and the fan was a possibility.

2. We treated all members with respect and fairness.

   To ensure constant and seamless communication between the team, we created a group chat dedicated to completing our project. By sharing code files and details of the hardware implementation, we consistently portrayed a well-oiled machine that worked properly, even with the pitfalls we faced across the way.

A.2 Safety

A.2.1 PCB Design

There were general risks to PCB assembly and the use of electronic components. Solder was used to stick our sensors and microcontroller onto one PCB, therefore, we had to beware of burn hazards as well as chemical hazards. We aimed to proceed cautiously with the use of gloves as well as safety goggles.

A.2.2 Usage of Dust

One main safety hazard related to our project would have been excessive inhalation of dust. Our goal was to demo our project by creating an environment filled with dust particles to see if our clean environment was capable of filtering all of it out. When creating this dust-infested environment, we had to wear masks and control the dispersion of dust particles as the buildup of it in our body could have been very dangerous, the consequences of which included lung infection and even more serious complications for those with asthma.

A.2.3 Usage of Blowers

There were various IEEE safety regulations associated with the usage of blowers and their application in this project. Blowers were high-demanding, high-quality machines that required precise measurements and accurate usage to ensure the best possible performance. However, common issues that happened to these blowers were symmetry irregularities, rotor malfunctions, and speed. These could occur given specific instances of a manufacturing issue, a power surge, or even a power failure. Symmetrical variations in the blower may have caused either incorrect

directional airflow or even an internal issue in the functionality of the device. [9] Voltage dips may have caused the blower to function improperly, either at low capacity or turn off, depending on the power input. Power surges could have caused the capacitors within the blower to overheat and malfunction, making the entire device worthless. We purchased a blower from a well-known manufacturer and ran basic tests on it to test its performance and symmetrical properties.

Appendix B Costs and Schedule

Costs of all Parts

Part	Manufacturer	Quantity	Retail Cost ($)	Bulk Purchase Cost ($)	Actual Cost ($)
3'' Mailing Tubes	Walmart	3	20.97	20.97	20.97
Utility Duct Tape	Walmart	1	4.99	4.99	4.99
3'' Dia PVC Pipe	Amazon	1	34.93	34.93	34.93
LCDs	Amazon	2	19.98	19.98	19.98
Tubberware Boxes	Amazon	2	19.98	19.98	19.98
Relay Module	Amazon	1	5.50	5.50	5.50
DC Fan	Amazon	1	9.99	9.99	9.99
PMSA003I PM2.5 Sensors	Adafruit Industries	2	39.95	39.95	39.95
SCD30 CO2 Sensors	Sensirion	2	128.80	128.80	128.80
Big Fan	Mouser	1	28.88	28.88	28.88
ESP32 Microcontroller	Amazon	1	10.99	10.99	10.99
Total					**324.96**

Schedule

Week	Task	Group Member
2/20 - 2/27	● Order remaining parts ● Complete design document ● Start PCB design ● Prepare for design review on 02/27 ● Finish team contract (due 02/24)	● All
2/27 - 3/06	● Complete PCB design and pass audit (03/07 due date) ● Meet with machine shop to manufacture enclosure ● Test sensors with Verilog code via I2C protocol ● Test the blower control circuit ● Test microcontroller	● All ● Omar ● Karthik ● Prithvi
3/06 - 3/13	● Assemble blower ● Test efficiency of dust filter by utilizing blower and PM2.5 sensors	● Karthik ● Omar

	• Complete teamwork evaluation (due 03/08) • Make sure nothing else is needed of machine shop (revisions due 03/10)	• All
3/13 - 3/20	• Spring break	
3/20 - 3/27	• Integrate sensors with PCB board • Make sure microcontroller is capable of receiving data from sensors	• Prithvi and Omar
3/27 - 4/03	• Complete individual progress reports (due 03/29) • Make sure microcontroller is able to communicate with blower	• All • Karthik and Prithvi
4/03 - 4/10	• Program microcontroller to change blower speed depending on readings from sensors (PM2.5 and CO2) • Start assembling enclosure with PCB, blower, and filter	• All
4/10 - 4/17	• Complete team contract fulfillment (due 04/14) • Continue maximizing efficiency for data collection, blower control, dust filtration, and air circulation • Start on presentation powerpoint	• All
4/17 - 4/24	• Mock demo • Make changes based on feedback from mock demo • Complete presentation preparations	• All
4/24 - 05/01	• Final demo and mock presentation	• All
05/01 - 05/08	• Final presentation • Finish final paper (due 05/03)	• All

References

[1] G. Qadir, “New Delhi's air pollution leaves residents gasping for breath,” *NBCNews.com*, 14-Nov-2022. [Online]. Available: https://www.nbcnews.com/news/world/india-new-delhi-air-pollution-rcna56535. [Accessed: 03-May-2023].

[2] S. Mishra, N. K. Singh, and V. Rosseau, “Display Interfaces,” *System on Chip Interfaces for Low Power Design*, 27-Nov-2015. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/B9780128016305000049. [Accessed: 03-May-2023].

[3] J. Valdez and J. Becker, “Understanding the I2C Bus,” Jun-2015. [Online]. Available: https://www.ti.com/lit/an/slva704/slva704.pdf. [Accessed: 03-May-2023].

[4] Devadmin, “PM2.5 explained,” *Indoor Air Hygiene Institute*, 22-Apr-2021. [Online]. Available: https://www.indoorairhygiene.org/pm2-5-explained/#:~:text=Most%20studies%20indicate%20PM2.,breathing%20issues%20such%20as%20asthma. [Accessed: 03-May-2023].

[5] “Carbon dioxide,” *Wisconsin Department of Health Services*, 29-Mar-2023. [Online]. Available: https://www.dhs.wisconsin.gov/chemical/carbondioxide.htm. [Accessed: 03-May-2023].

[6] “DC Fan 40 28 mm - mouser electronics,” *Mouser Electronics*. [Online]. Available: https://www.mouser.com/datasheet/2/471/San_Ace_40GA28_E-1285263.pdf. [Accessed: 03-May-2023].

[7] “ESP32WROOM32E ESP32WROOM32UE - Espressif,” *Espressif Systems*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf. [Accessed: 03-May-2023].

[8] “IEEE code of Ethics,” *IEEE*, Jun-2020. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 03-May-2023].

[9] “Standards,” *IEEE Public Safety Technology Initiative*. [Online]. Available: https://publicsafety.ieee.org/standards. [Accessed: 03-May-2023].