DEVELOPING AND IMPLEMENTING INTO PRACTICE AN ENVIRONMENTAL MONITORING STATION WITH AN ARDUINO PRO MINI

A THESIS SUBMITTED TO
THE FACULTY OF ARCHITECTURE AND ENGINEERING
OF
EPOKA UNIVERSITY

BY

NASHO THEMELI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

JUNE, 2024

Approval sheet of the Thesis

This is to certify that we have read this thesis entitled **“Developing and Implementing into Practice an Environmental Monitoring Station with an Arduino Pro Mini”** and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

____

Assoc. Prof. Dr. Arban Uka
Head of Department
Date: June, 28, 2024

Examining Committee Members:

Prof. Dr. Gezim Karapici	(Computer Engineering)	________________
Prof. Dr. Betim Çiço	(Computer Engineering)	________________
Assoc. Prof. Dr. Dimitrios Karras	(Computer Engineering)	________________

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name Surname: Nasho Themeli

Signature: _______________

ABSTRACT

DEVELOPING AND IMPLEMENTING INTO PRACTICE AN ENVIRONMENTAL MONITORING STATION WITH AN ARDUINO PRO MINI

Themeli, Nasho
M.Sc., Department of Computer Engineering
Supervisor: Prof. Dr. Betim Çiço

This study focuses on the development and practical implementation of an environmental monitoring station utilizing the Arduino Pro Mini microcontroller. The station is designed to measure and record various air quality parameters such as PM2.5, CO2, VOCs, ozone, temperature, humidity, light intensity, and UV radiation. The system integrates multiple sensors, including the PMS5003 for particulate matter, MH-Z19 for CO2, MP503 and MQ-131 for VOCs and ozone, and DHT22 for temperature and humidity. The Arduino Pro Mini serves as the central microcontroller, processing data from these sensors and transmitting it to a Nextion touchscreen display for real-time monitoring and analysis. The study reviews 17 related research papers to identify current trends, resolved and unresolved issues, and future directions in the field of microcontroller-based environmental monitoring systems. The results highlight the system's capability to provide accurate and real-time air quality data, demonstrating its potential for broader ecological monitoring and public health applications. Future work aims to enhance sensor integration, system scalability, and IoT connectivity for remote monitoring.

Keywords: *Microcontrollers, Sensors, Arduino Pro Mini, Environmental Monitoring, Air Quality Measurement.*

ABSTRAKT

DIZENJIMI DHE IMPLEMENTIMI NË PRAKTIKË I NJË STACIONI MONITORIMI MJEDISOR ME NJË ARDUINO PRO MINI

Themeli, Nasho

Master Shkencor, Departamenti i Inxhinierisë Kompjuterike

Udhëheqësi: Prof. Dr. Betim Çiço

Ky studim fokusohet në zhvillimin dhe zbatimin praktik të një stacioni monitorimi mjedisor duke përdorur mikrokontrollerin Arduino Pro Mini. Stacioni është dizajnuar për të matur dhe regjistruar parametra të ndryshëm të cilësisë së ajrit, si PM2.5, CO2, VOC, ozon, temperaturë, lagështi, intensitet drite dhe rrezatim UV. Sistemi integron sensorë të shumtë, përfshirë PMS5003 për materien e grimcave, MH-Z19 për CO2, MP503 dhe MQ-131 për VOC dhe ozon, dhe DHT22 për temperaturë dhe lagështi. Arduino Pro Mini shërben si mikrokontrolleri qendror, duke përpunuar të dhënat nga këta sensorë dhe duke i transmetuar ato në një ekran me prekje Nextion për monitorim dhe analizë në kohë reale. Studimi rishikon 17 artikuj kërkimorë të lidhur për të identifikuar tendencat aktuale, problemet e zgjidhura dhe të pazgjidhura, dhe drejtimet e ardhshme në fushën e sistemeve të monitorimit mjedisor bazuar në mikrokontrollerë. Rezultatet theksojnë aftësinë e sistemit për të siguruar të dhëna të sakta dhe në kohë reale për cilësinë e ajrit, duke demonstruar potencialin e tij për aplikime më të gjera në monitorimin mjedisor dhe shëndetin publik. Puna e ardhshme synon të përmirësojë integrimin e sensorëve, shkallëzueshmërinë e sistemit dhe lidhjen IoT për monitorim të largët.

Fjalët kyçe: *Mikrokontrollorë, Sensorë, Arduino Pro Mini, Monitorim Mjedisor, Matja e Cilësisë së Ajrit.*

I dedicate this thesis to my beloved family for their unconditional support and continuous encouragement throughout this academic journey. Special thanks to my parents, who have always inspired me to follow my dreams and work with dedication and passion. I also want to express my deep gratitude to my professors and mentors, especially Prof. Dr. Betim Çiço, for his invaluable guidance and unwavering support. Without your help and support, this work would not have been possible. This achievement is also a reminder of the importance of hard work, perseverance, and self-belief.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to everyone who has supported and guided me throughout the development and completion of this thesis. First and foremost, I thank my advisor, Prof. Dr. Betim Çiço, for his invaluable guidance, constructive feedback, and continuous encouragement. His expertise and insights have been instrumental in shaping this research. I extend my heartfelt thanks to my collaborator, Dorjan Dafku, whose dedication, hard work, and insightful contributions have been pivotal in the successful completion of this project. Your partnership and shared commitment have greatly enriched this research. I am extremely grateful to the faculty and staff of Epoka University for providing the resources and support necessary for my research, with special thanks to my professors and colleagues for their invaluable advice and encouragement throughout my academic journey. I would also like to extend my heartfelt thanks to my family for their unwavering support, patience, and love. Your belief in me has been a constant source of motivation and strength. Finally, I acknowledge the authors of the research papers and resources that have been integral to this study. Your work has been a source of inspiration and knowledge, greatly contributing to my research. Thank you all for your support and contributions. This thesis would not have been possible without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Due to the innumerable inventions and technical advances made possible by microcontrollers and sensors, the electronics industry has seen a profound evolution in recent years. The range of electronic measurements derived in all the many areas of integration has been redefined by these and numerous other components. As a result, the goal of this academic investigation is to highlight and illustrate how microcontrollers, sensors, and other integrating devices interact to form the core of the revolution mentioned above, thereby improving the efficiency, accuracy, and flexibility of the measurements we want to take in this case, environmental air quality.

Because of their ability to carry out specific tasks and produce impressive outputs, integrated circuits, microcontrollers with power processing units, and programmability have undoubtedly served as the foundation of today's modern electronic systems. Successful measurements have also been conducted with these components. On the one hand, our microcontrollers can integrate several sensors and produce real-time data that has already been processed and gathered by coordinating with a designated centralized control unit. Conversely, sensors are unquestionably crucial because they serve as the sensory component or organs of electronic systems, transforming the current physical phenomena within a predetermined range region into electrical impulses.

The variety of sensors available is genuinely astounding as they measure conditions such as humidity, temperature, particulate matter, carbon dioxide, Volatile organic compounds, and more. In this specific instance, however, we focus purely on the air quality sensors and their vital role in measuring, capturing, and recording changes in environmental parameters for a variety of applications that range from automation and industrial usage to daily living, healthcare examinations, and even just functioning as informative units.

The study demonstrates the basic implementations of several air quality sensor technologies, such as PM2.5, CO2, humidity, temperature, and more, in our surroundings through the use of an Arduino Pro Mini microcontroller that we have properly designed. After conducting a thorough analysis of over 17 case studies and academic research papers and observing their practical applicability in related areas, our goal is to present here the ways we can work with each other to address the issues that we encounter daily, including a better understanding of our air pollution while also mentioning recent developments or advancements like Internet of Things protocols that allow for sophisticated remote controlling and monitoring of our systems by third parties.

To continue on our educational journey, our ultimate goal is to give others who are interested a better understanding of the world of electronics. We will do this by explaining the connections among Arduino and additional electronic devices, emphasizing their impact on monitoring different processes using the reading measurements like accuracy, reliability, and scalability, and demonstrating the challenges and implementing risks on the circuit board to help us access our goals. We will additionally try to organize these metrics according to a touch-screen display for 24 hours, as well as by offering an up-close view of our air quality and how each air parameter affects humans.

1.2 Motivation

The constantly evolving energies incorporated within the microprocessors as well as detectors incorporated in these devices stand apart as an essential component to solving the continually evolving and expanding field of electronics complexities. Entering this quickly demanding field from our pressing demands reveals the revolutionary capabilities of these essential elements that drive creativity and flexibility in measuring at a time when the technical nature of this field keeps growing.

Accuracy or precision represents one of the most important indicators in electronics, and since conventional ones don't live up to the standards or the demands necessary, an in-depth examination of the above elements becomes necessary to improve our measurements' accuracy. Microcontrollers, with their potent computational powers, can be used to execute or obtain delicate information from the sensors. However, the near future of manufacturing sectors such as automated processes, medical care, environmental, and more., will greatly benefit from utilizing this feature.

The accuracy of current time information and the drive for achieving its optimization serve as additional key metrics supporting this research. To help speed up the measurement procedure, microcontrollers acting like intelligent organizers examine the real-time execution or operation of multiple sensors. This facilitates the information acquired throughout their respective positions' detecting process and minimizes inaccuracies correlated to noise disruptions with assistance from individuals. To fully realize the possibilities of this cooperation is implied by the need to discover every opportunity or potential for this performance, which essentially speeds operations and reliability in various measuring situations.

Furthermore, as the Internet of Things grows, there will be a growing need for electronic equipment that can readily interact with related systems. This makes the flexibility provided by Arduino and detectors essential to meeting the expectations of advances in technology. The computational center components and microcontrollers enable flexible control, quick data transmission, and remote monitoring. This stimulates further research in the combination of sensors and microcontrollers to maximize the Internet of Things' potential, resulting in a new era of networked electronic devices.

To sum up, we are driven to investigate Arduino microcontrollers and sensors via technology-related measurements not just by current trends but additionally by the urgent need to understand and resolve the intricacies of modern systems through the analysis, combination, testing, and programming of these components to discover perspectives of the mentioned frameworks and to explore through unknown areas of innovation and potential future projects. Consequently, our goal is to create a low-cost, high-precision, handy, as well as properly scale-efficient monitoring device that can measure and record the concentration of our primary gases and other air contaminants, as well as present and connect the data generated everywhere it is needed for analysis and subsequent procedures.

1.3 Objectives

Recognizing a microcontroller's building design that we are using:

Giving a thorough overview of our microprocessor architecture, outlining its essential parts (storage space, input and output links, power processing unit, etc.) and how it interacts with other parts. Highlight the essential function that our microcontroller performs in accomplishing our primary monitoring operation and in controlling real-time processes captured through a planned measuring system using a real-time clock (RTC module).

A detailed analysis of our sensor technologies:

Analyzing the broad range of sensors, we use, such as particle matter, temperature and humidity, and Volatile Organic Compounds (VOC) sensors, among others, to gain a greater awareness of the fundamental ideas behind their functioning and how they communicate inside our monitoring station. Examining developments in the field of sensors and how they impact electronics measuring devices' capability to assess more accurate readings efficiently.

The programming methods implemented in the Arduino board and our Nextion Display:

Exploring inside technological advances appropriate for microcontroller-based systems languages of programming like the Arduino IDE, with the main goal being optimization and attaining optimal performance. Coding our touch display to give us extensive details about the measurements we have taken from all of the sensors connected to our system in synchronization with the real-time clock module.

Techniques for constructing our microcontroller and detectors:

Taking into consideration various approaches for the best potential incorporation of the Arduino Pro Mini and the sensors, keeping in mind crucial hazards including device communication protocols, data synchronization between devices, and calibration. Examining the related issues that exist when wiring and communicating through to gain an overview of the system setup and integration.

Applications and Case Studies:

Even our instance of the air monitoring station, microcontrollers, and sensors are being used in every possible useful everyday life application in a variety of fields, such as medical care, present scientific and manufacturing automation. Focusing on how utilizing scheduled in advance, carefully thought-out microcontrollers and sensors can improve the system's efficiency and measurement precision.

The integration of the Internet of Things (IoT):

Considering various analyses and acquiring a broader understanding of the additional incorporation of detectors into IoT-enabled electronic measuring systems as a current tendency. Examining how microcontrollers function to provide communication, transparency for monitoring from afar, and information processing into multiple devices, with a focus on the possibility of more adaptability and lower scalability.

The performance evaluation:

Determining the critical performance requirements, such as precision accuracy in setup, information processing time, and energy efficiency, which we are going to use for our stationary measuring infrastructure that integrates microcontrollers and detectors and is dependent on our changes or advancements. Examining the various ways sensor arrangements, configurations, and calibration.

Upcoming future trends:

Explain the way new developments in sensors and microcontroller architecture could impact the way electronic measurements are performed soon using our system as an example. Encouraging a bright prospective vision and broadening the opportunities.

1.4 Organization of the Thesis

This thesis is divided into 7 chapters. The organization is done as follows:

In Chapter 1, the problem statement, motivation and objectives of works is presented. Chapter 2, includes the literature review……. Chapter 3, consists of the methodology followed in this study……. In Chapter 4, the experimental results …... In Chapter 5, conclusions and recommendations for further research are stated.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The literature review in our work comprises various papers explored during our case study, focusing on the latest advancements in microcontrollers and sensor technology. These papers are organized in the table below, which includes the paper title, authors, publication year, and a summary of the issues addressed, proposed solutions, recommended future work, and other relevant details. Additionally, we have analyzed these papers, most of which were published in the last four years, and grouped them into four categories based on the similarities in their content.

2.2 Summary of Literature Review (Resolved Problems)

After going through the articles listed above, we divided them into three groups, each with an overview of the challenges they solved as well as their techniques or answers. The classification depends on similarities, such as the microcontroller utilized or planned for their application intents, how technology is integrated into items, and so on.

Group 1: Environmental Events Detection and Monitoring

The publications listed above focus on building monitoring and control systems for recognizing and measuring environmental characteristics such as temperature, humidity, air quality, and pollution levels using particular sensors. They are generally built on microcontrollers such as the Arduino Pro Mini or Mega to deliver efficient smart solutions and advancements for indoor and outdoor environments. In addition, a couple of solutions involve the implementation of wireless sensor networks and the use of IoT interactivity for real-time data collecting and analysis.

Group 2: Sensor Technology and Practical Applications

The studies in group 2 investigate very outstanding emerging innovations and new implications in the field of sensors, starting with analog front ends and circuits for a variety of applications, largely industrial, comparable to our work. They emphasize model advancements and electronics issues in programmable components and detectors and analyze their performance to achieve highly accurate integration, while also determining how to improve performance in a variety of ways.

Group 3: IoT and Remote connectivity

This collection of papers exhibits recent study cases in applications for the Internet of Things and ways to connect between sensors using microcontroller-based platforms such as Arduino, which are the focus of our work. The papers in this section discuss weather surveillance systems, residential alert systems, and cost-effective sensor assessments. They primarily emphasize the role of IoT in improving efficiency, internet access, automation across several domains, data transmission flexibility, and the creation of beneficial outcomes and networks.

2.3 Summary of Literature Review (Future Work)

In this particular section, we have conducted a summary of the unsolved problems proposed by each group of the specific papers. These problems appear as a consequence of their testing or their performance evaluation during the performance of their experimental conducting. While some of these papers prefer to specify their problems, the rest have shown small indices of their challenges. Based on studying this section, we have been able to construct the research gaps and also the enthusiasm to operate our work.

For Sensor Technology, the challenges include enhancing sensor accuracy and reliability, especially in detecting physical environments, improving sensor calibration techniques for optimal performance, and addressing compatibility, scalability issues, and providing noise immunity.

In Communication and Connectivity, the focus is on developing better communication protocols for improved data transmission and increasing the wireless communication range for remote applications.

For Control and Automation, the main issues are optimizing power consumption for prolonged battery life and low power consumption, and improving system responsiveness and real-time response capabilities.

Regarding Safety and Efficiency, the problems include improving algorithms for early detection and prediction in the monitoring process, focusing on privacy concerns associated with sensitive data transmission, and optimizing energy efficiency and reliability.

2.4 Summary of Literature Review (Unsolved Problems)

In this particular section, we have conducted a summary of the unsolved problems proposed by each group of the specific papers. These problems appear as a consequence of their testing or their performance evaluation during their experimental conducting. While some of these papers prefer to specify their problems, the rest have shown small indices of their challenges. Based on studying this section, we have been able to construct the research gaps and also the enthusiasm to operate our work.

In the area of Sensor Technology, the challenges include enhancing sensor accuracy and reliability, especially in detecting physical environments, improving sensor calibration techniques for optimal performance, and addressing compatibility, scalability issues, and providing noise immunity.

Regarding Communication and Connectivity, the focus is on developing better communication protocols for improved data transmission and increasing the wireless communication range for remote applications.

For Control and Automation, the main issues are optimizing power consumption for prolonged battery life and low power consumption, and improving system responsiveness and real-time response capabilities.

In the context of Safety and Efficiency, the problems include improving algorithms for early detection and prediction in the monitoring process, focusing on privacy concerns associated with sensitive data transmission, and optimizing energy efficiency and reliability.

2.5 Research Gaps

Besides the research done in these papers, some parts need to be resolved and critically analyzed. We have listed below the main research gaps for every group of papers:

Group 1: Environmental Phenomena Detecting and Monitoring:

Energy Efficient Air Quality Monitoring System using Arduino: Research Gaps: Integration of real-time data organization to optimize energy consumption dynamically, standardized protocols for data communication and control across different systems, evaluation of long-term performance and maintenance needs in diverse environmental conditions, and increasing trials performed.

Microcontroller-Based Detection and Prevention System: Research Gaps: Improvements in sensor sensitivity and longer range to detect accurately, development of fail-safe protection mechanisms and alert systems wherever needed, longer time reliability, and careful calibration of sensors in varying conditions.

Framework for Real-Time Air Monitoring: Research Gaps: Security challenges in IoT networks, design and implementation of the system for large-scale deployment, and integration of predictive analytics for proactive pollution management.

Group 2: Sensor Technology and Applied Applications:

High-Precision Microcontroller for Industrial Sensing: Research Gaps: Hardware optimization to minimize noise and enhance precision in industrial applications, development of cost-effective solutions for production and deployment, ensuring compatibility and integration with existing industrial control systems and standards.

Wireless Temperature Sensor: Research Gaps: Integration of wireless communication modules without compromising sensor performance, exploration of new application areas, and customization for specific industrial or medical needs.

Group 3: IoT and Remote Connectivity:

Arduino-Based Ambient Air Pollution Sensing System: Research Gaps: Development of reliable low-power techniques for extended sensor operation, standardization of data formats and communication protocols for seamless integration with other systems, addressing data issues in large sensor networks.

In our research work, we will be focused on the first group of papers analyzed, moreover on the microcontroller-based detection and prevention system gaps.

2.6 Hypotheses and Research Questions

Hypothesis 1: Environmental Phenomena Detecting and Monitoring

Hypothesis: Implementing real-time data integration recorded from multiple environmental sensors will significantly optimize energy consumption in systems and improve accuracy.

Research Questions:

1. How does the integration of real-time data impact the energy efficiency of our systems?
2. What improvements in sensor technology can enhance the sensitivity and specificity of our monitoring systems?
3. What are the long-term reliability and maintenance requirements needed for integrated environmental monitoring systems in diverse conditions?

Hypothesis 2: Health and Medical Systems Monitoring

Hypothesis: Maximizing the capabilities of wireless health monitoring systems to include multiple signs will enhance their performance matrices, especially in diverse and resource-limited settings.

Research Questions:

1. How can multi-sensor integration improve the accuracy and reliability of wireless health monitoring systems?
2. How does ensuring data privacy and security impact the effectiveness of wireless health monitoring systems in real-world settings?

Hypothesis 3: Sensor Technology and Applied Applications

Hypothesis: Improving the formats and communication protocols of the data recorded, will enhance the reliability and scalability of IoT-based air pollution sensing systems, facilitating their large-scale deployment and interconnection with other IoT systems.

Research Questions:

1. What are the most effective low-power techniques to extend the operation of IoT sensor nodes in air pollution monitoring systems?
2. How does the standardization of data formats and communication protocols impact the integration and performance of large-scale IoT sensor networks?

2.7 Research Aim

This research aims to investigate and optimize the integration and performance of multi-sensor environments. This includes enhancing energy efficiency, accuracy, and usability of systems, as well as addressing the challenges of data standardization, security, and scalability. Specifically, the research seeks to improve the energy efficiency and functionality of air quality monitoring systems through the integration of real-time environmental data and advanced sensor technologies.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This section outlines the techniques and methods used to process, select, and investigate the resources and papers relevant to our goal of creating and evaluating an air monitoring system that is effective, accurate, and easily accessible. This design process includes the development of blueprints and circuit diagrams. Comprehensive examinations will be conducted under various conditions to test and measure the performance of the devices. Visualizations will be provided to display the data collected from each sensor over 24 hours. Additionally, coding will be done for the Nextion Display and the Arduino Pro Mini to facilitate real-time monitoring and analysis of the measured parameters.

Normative investigations will be presented both quantitatively, using the experimental approach, and qualitatively, to provide a thorough evaluation of the system's performance.

3.2 Quantitative Research

As mentioned above the work of this research will serve to determine our indication of the according field of study. The results of our modified work organized schematically and mathematically will be used to test the effects predicted by our hypotheses while also comparing to the existing systems to show which one provides better performance.

3.3 Qualitative Research

As we mentioned above, in this work qualitative research will be included. It will point out the significance of this research and also give a kind of background about this research area. There will also a research about the designs used before to improve, always by concentrating on the aspects that will be studied.

CHAPTER 4

SYSTEM DESIGN

4.1 Introduction

Our study will be focused on building an environmental monitoring station combining the hardware components and using the software to activate the sensors and present their reading in a touch display. The first thing was the selection of the components, which we will present down below, as well as their detailed explanations such as their datasheets and their working principles. Furthermore, we also will build our circuit diagram to show how the connections are made. Moreover, we will build our circuit 3D model based on the circuit diagram to create a better visualization of how this monitoring station would look if it was implemented in a PCB. We intend to order a custom PCB in the future, but just for the moment, we are using a universal PCB for our prototype.

4.2 Design Specification

As shown below in *Table 1* our monitoring system is constructed based on 8 detectors ranging from the PMS5003, MH-Z19B CO_2, MQ-131 low-concentration Ozone sensor, MP503 VOC, DHT22 temperature and humidity, LDR (Light Depended Resistor), as well as a UV sensor module. All the units will be powered on with an AC/DC adapter operating at an input of 100-240 V and producing an output of 5V, 2A which can power all sensors. The controlling unit for our study will be an Arduino Pro Mini, which operates at 5V and will serve as the sensor interface to collect the data provided by each sensor. To present the readings of our sensors we will be using a Nextion touch display, which is a user-friendly output unit that allows the end users to interact as well as perform data visualization, and real-time monitoring, enable plotting graphs, and give alerts when the thresholds are exceeded.

Table 1. Design Components of The System

	The List of Components	
Nr.	**Component**	**Operation**
1.	PMS5003	Particulate Matter Sensor
2.	MH-Z19	Carbon dioxide sensor
3.	MQ-131	Low-concentration ozone sensor
4.	MP503	VOC sensor
5.	DHT22	Temperature & Humidity sensor
6.	Nextion NX3224T028	2.8″ Display Output unit
7.	Arduino Pro Mini	Microcontroller unit
8.	DS3231	High-precision Real-Time Clock module
9.	Mini USB Connector	Power unit 5V
10.	LDR4589	Light Dependent Resistor
11.	UV 8521	UV Sensing unit

4.3 Design Schematics and Flowchart

Figure 1 depicts our proposed design concept schematics, which are performed using KiCad software. It includes all the devices used in our work with each pin and the way they are connected within the system, whereas *Figure 2* shows the flowchart of the system communication between the input block containing all our sensors connected in serial connection with their respective values of the display output, signal processing, and the display. Additionally, we have included the block diagram and the operational description highlighted as follows.

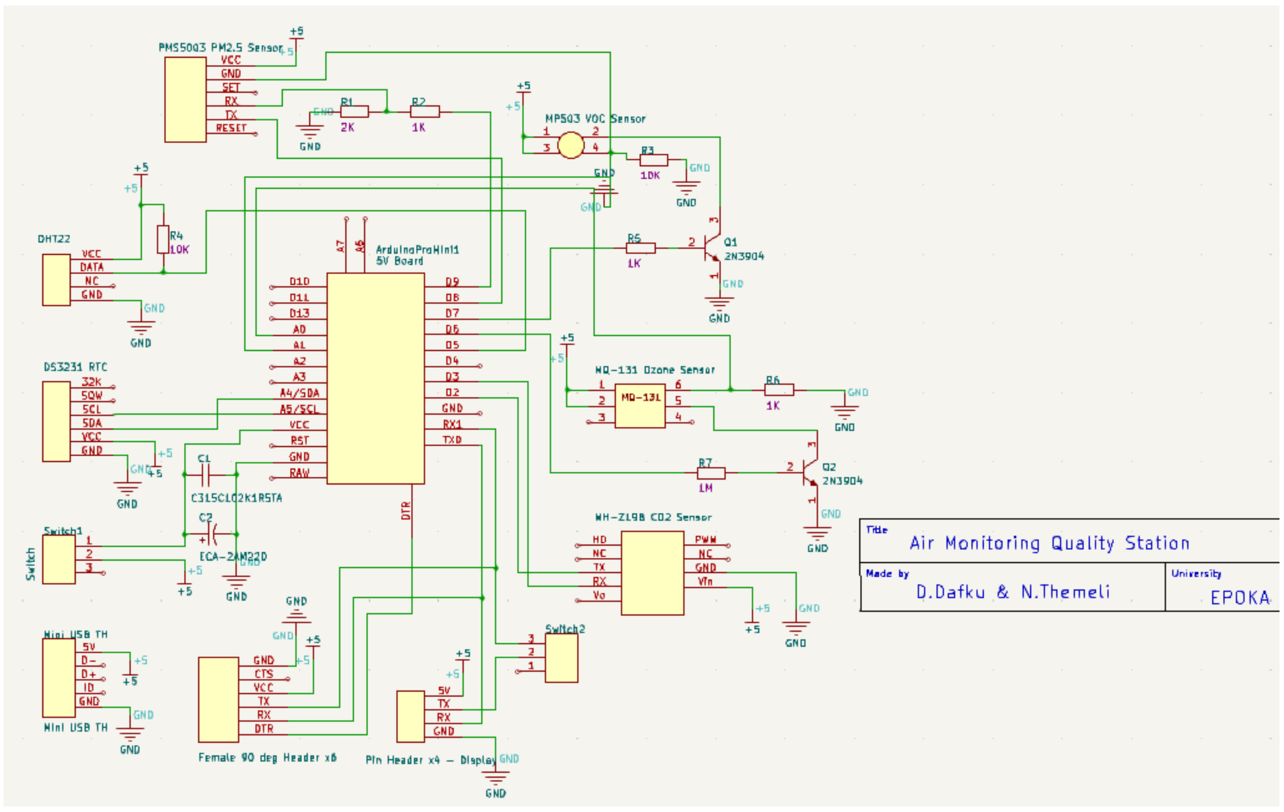

Figure 1. The Schematics design of the system using KiCad

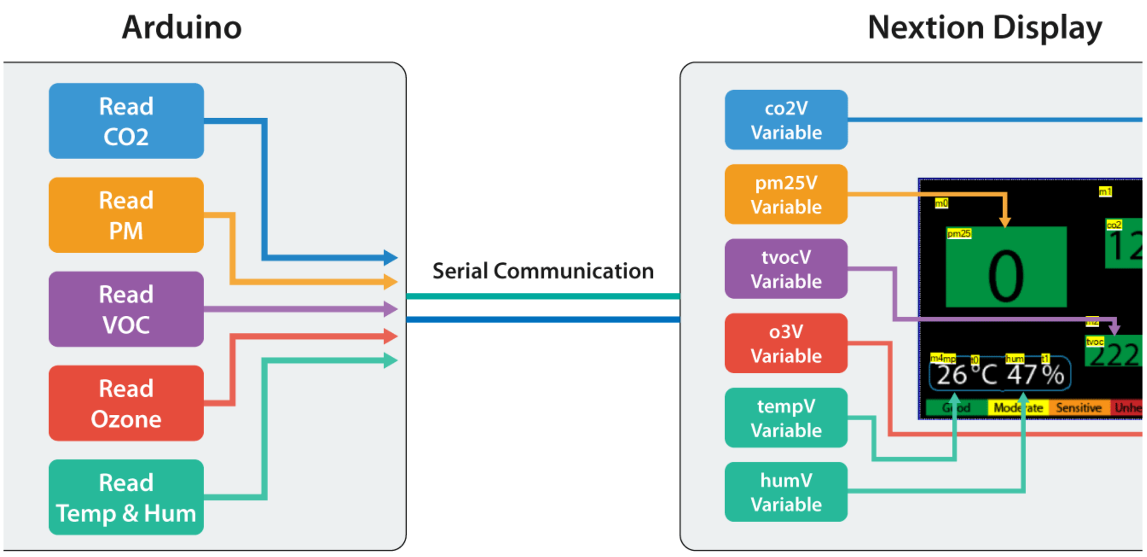

Figure 2. The Circuit Flowchart of the System between Input / Output

4.4 Design Block-diagram

Figure 3 shows the block diagram of the circuit represented in 3D. The model consists of the PM2.5 sensor which is for sensing particulate matter with a diameter of less than 2.5 microns. This sensor requires 5V to power on, but the RX logic level of its components requires 3,3V, so we have to use a voltage divider. Moreover, we have the CO2 sensor, just like the previous sensor, it also uses the serial connection. For the other sensors, like VOC and Ozone we have to use analog input ports of the Arduino Pro Mini, while the DHt22 uses digital connection input ports. Since VOC and Ozone sensors have built-in heaters, we use 2 transistors to activate them.

For traction of the time a high-precision Real-Time Clock is used. The RTC uses I2C communication protocole. The device is powered by an AC/DC adapter which produces a 5V, 2A, enough power to keep up with all the sensors.

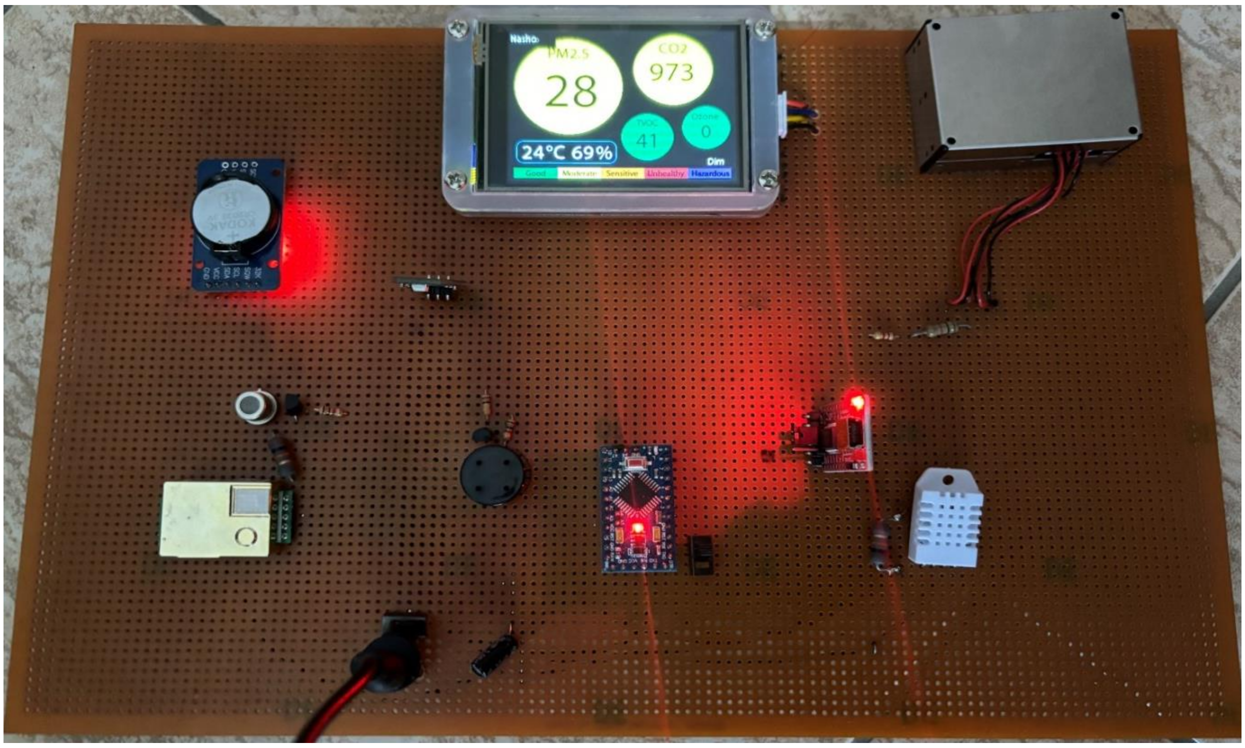

Figure 3. Real-Time Circuit Implementation of the System

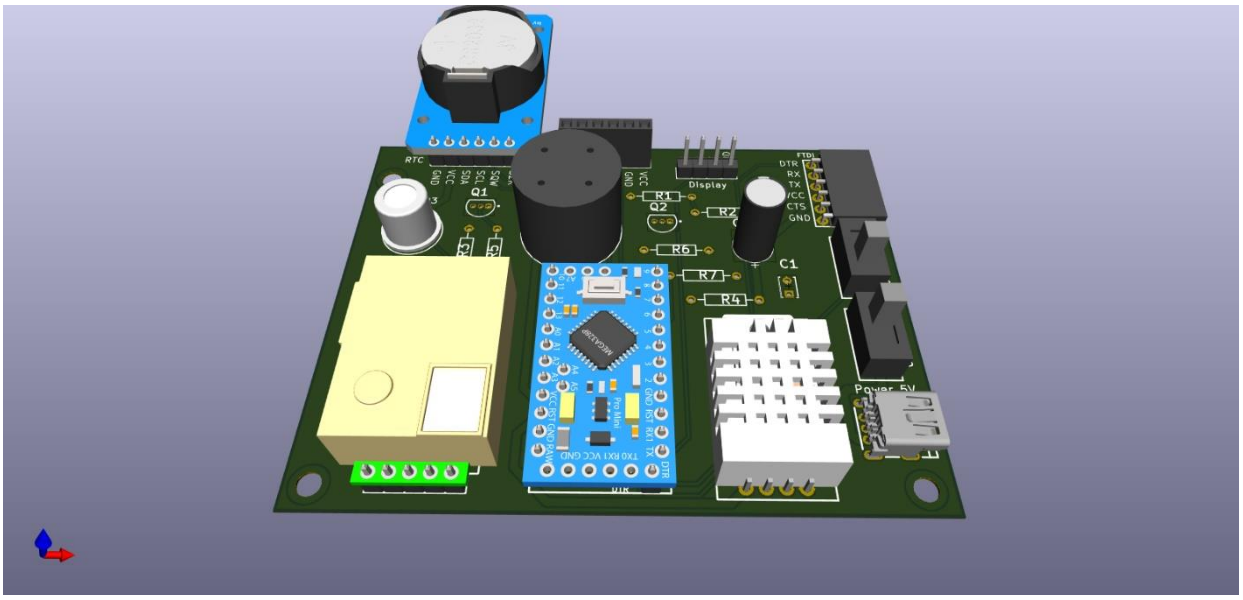

Figure 4. The functional Block Diagram of the system in 3D using KiCad

4.5 The PM2.5 Sensor – PMS5003

The basic principle of this sensor is that it works as a digital particle concentration sensor that calculates the intensity of pollutants in the air and then it produces an output in the form of a digital interface for our system and provides correct data in time. It has four main components and measures matter in the air of around 2.5 microns which by the way are considered truly harmful for humans hence they penetrate deep into our system causing different health problems.

Working principle: The sensor uses a laser or LED light to illuminate the air sample. A photodetector measures the light scattered by particles in the air sample. The sensor has a small fan to draw air into the sensing chamber. A microcontroller processes the data from the photodetector and calculates the PM2.5 concentration. Air Sampling: The sensor draws an air sample into the sensing chamber using a fan. Light Scattering: The light source (laser) emits light that interacts with the particles in the air. When the light hits a particle, it scatters in different directions. Detection: The photodetector captures the scattered light. The amount and angle of scattered light depend on the size and concentration of the particles. Signal Processing: The microcontroller processes the signal from the photodetector. It uses algorithms to interpret the scattered light data and calculate the concentration of PM2.5 particles. Output: The sensor outputs the PM2.5 concentration, usually in micrograms per cubic meter (µg/m³). This data can be displayed on an interface or sent to other devices for further analysis or action.

The block diagram of this sensor is directly taken from its datasheet and we will show below in *Figure 5* while its major configuration features are listed in *Table 2*.

Table 2. The PM2.5 Sensor Parameters

Feature	***Definition***	***Measurement Unit***
Component Name	PMS5003	
Operation Range	0.3-1.0;1.0-2.5;2.5-10	(µ m)
Efficiency in Counting	50% for 0.3µ m 98% for 0.5µ m	
The Range of effictiveness	0 up to 500	µ g/m3
The Maximum Range Covered	1000	µ g/m3
Error as Consistency	10% for 100 up to 500µ g/m3 10µ g/m3 for 0~100µ g/m3	
The Power Supply In DC	5.0 Min & 4.5 Max: 5.5	(V)
Current in Active Mode	100	(mA)
Current Standby mode	200	(µ A)
The Temperature Range of Operation	10 up to 60	°C

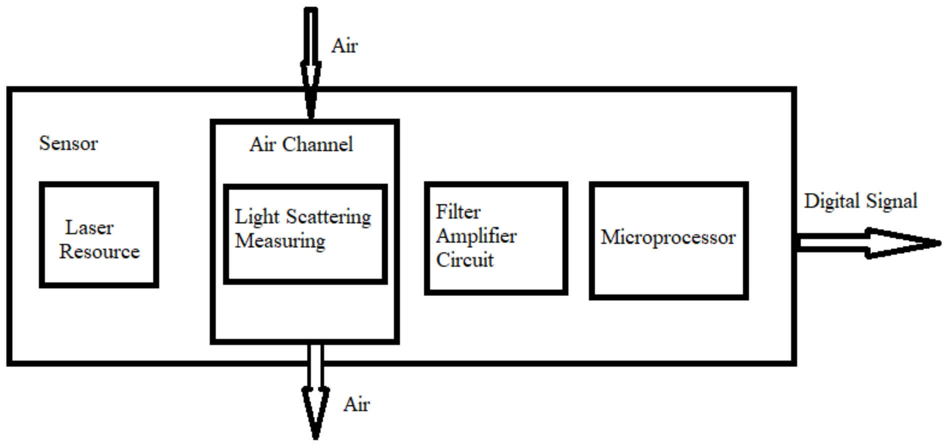

Figure 5. The Flowchart of the PM2.5 Functional Sensor Mechanism

The installation process of the PM2.5 sensor required a DV 5V power supply, as the fan inside the component needed to be driven by that voltage. Given that the data pin level was 3.3V, we used a conversion whenever the MCU was 5V. Concerning the pins, the set and reset pins, including pins 7 and 8, were not connected. Additionally, we ensured that the airflow path of the sensor remained free of any shield or structure.

The PMS5003 sensor communicates using UART with a specified baud rate to transmit data in a structured format consisting of 32 bytes. Each byte in the transmission has a specific purpose: Bytes 0-1 are the start characters (0x42, 0x4D), Bytes 2-3 indicate the frame length (always 28 bytes), Bytes 4-5 represent PM1.0 concentration, Bytes 6-7 represent PM2.5 concentration, Bytes 8-9 represent PM10 concentration, Bytes 10-11 represent PM1.0 concentration, Bytes 12-13 represent PM2.5 concentration, Bytes 14-15 represent PM10 concentration, Bytes 16-29 contain reserved data and other sensor data, and Bytes 30-31 indicate the checksum rate (9600 in this example).

The data frame, containing 32 bytes, includes start characters, frame length, PM concentrations, reserved data, and checksum. Data extraction involves retrieving PM1.0, PM2.5, and PM10 values from specific bytes in the data frame. Error checking utilizes the checksum to validate data integrity. The start characters (0x42, 0x4D) signify the beginning of a data frame, and the frame length always indicates 28 bytes for the PMS5003. PM concentrations are stored as two bytes each, representing concentrations in µg/m^3.

Table 3. PM2.5 Sensor Pin Definition

Pin Coordination	***Definition***	***Measurement Unit***
PIN1	VCC	5V
PIN2	GND	Negative
PIN3	SET	Set pin 3.3V
PIN4	RX	Serial
PIN5	TX	Serial
PIN6	RESET	Module reset signal
PIN7/8	NC	

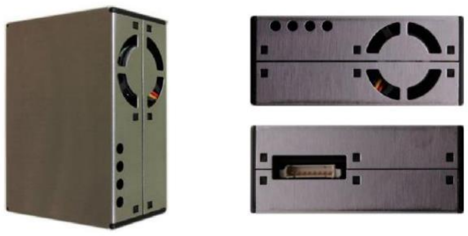
Figure 6. PM2.5 Sensor

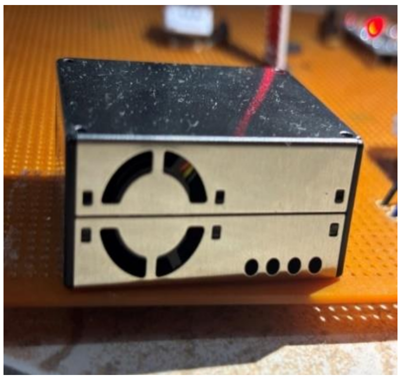
Figure 7. Real-Time Implementation in Board

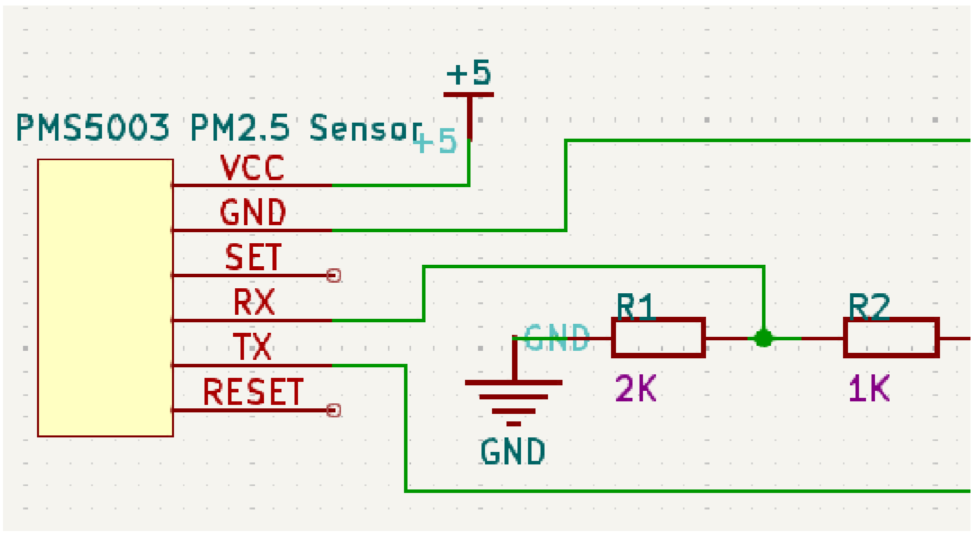

Figure 8. PM2.5 Schematics

4.6 The CO2 Sensor – MH-Z19

The next sensor that we are going to be using is the MH-Z19 which is responsible for recording the CO2 in the air. As humans breathe and exit carbon dioxide while respiration we can agree that the indoor concentration of this gas can easily reach high levels resulting in being dangerous not only for its density but also for its effects on us like tiredness, sleepiness, and more. This sensor uses infrared principles for measuring this gas in air an infrared source of the component directs or shoots light through a tube filled with the air where our device is operating, the CO2 molecules present absorb a specific band of the IR light directed from the source while letting some other wavelengths to pass through. The reasons that we chose this sensor are because of its good selectivity, long lifespan, accuracy, not depend on oxygen, and low power consumption while also its built-in temperature sensor can do the compensation. Below we represent its main features in *Table 4* and how it looks in *Figure 9*.

Table 4. The CO2 Sensor

Component	***Measurement Unit***
The Component Name	MH-Z19
Operational Voltage	From 3.6 up to 5.5 V
Operational Current	18 mA
Covered Range	From 0 up 0.5%
The Operational Working Temperature	From 0 up to 50 °C
Pin Nr.6	Vin
Pin Nr.7	GND
Pin Nr.1	Vout (3.3V, 10mA)

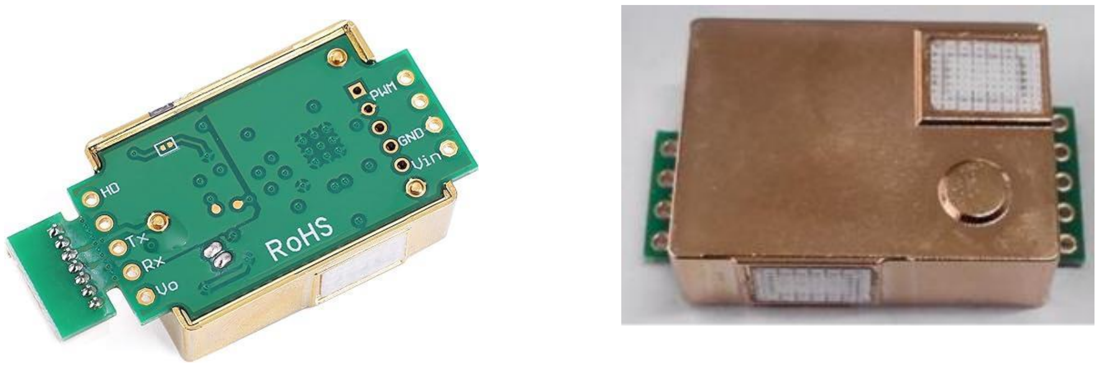

Figure 9 MH-Z19

The installation of the CO2 sensor MH-Z19 involves connecting the MH-Z19 VCC to the Arduino Pro Mini VCC (5V), the MH-Z19 GND to the Arduino Pro Mini GND, the MH-Z19 TX to the Arduino Pro Mini RX (Pin 2), and the MH-Z19 RX to the Arduino Pro Mini TX (Pin 3).

The MH-Z19 sensor transmits data in a structured format consisting of 9 bytes: Byte 0 is the start byte (0xFF), Byte 1 is the command byte (0x86), Bytes 2-3 contain the high and low bytes of the CO2 concentration (in ppm), Bytes 4-6 are reserved bytes (usually zero), and Bytes 7-8 are the checksum bytes. For data interpretation, the concentration of CO2 is calculated by combining bytes 2 and 3: CO2 Concentration =

(Byte 2 << 8) + Byte 3. The sensor sends a 9-byte data packet, which is read by the library to extract the CO2 concentration, with bytes 2 and 3 containing the CO2 concentration in ppm. For example, if the sensor sends 0xFF 0x86 0x01 0xF4 0x00 0x00 0x00 0x00 0x79, bytes 2 and 3 are 0x01 and 0xF4, which combine to form 0x01F4 (500 in decimal), resulting in a CO2 concentration of 500 ppm.

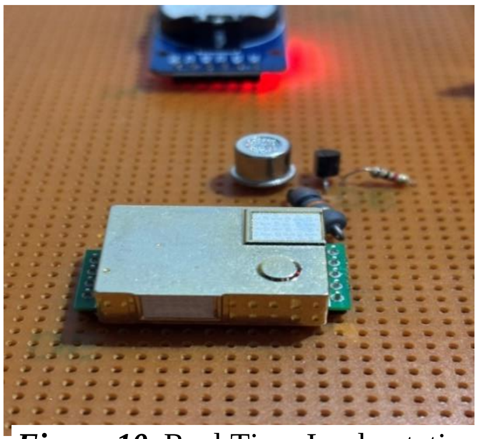
Figure 10. Real Time Implemtation in Board

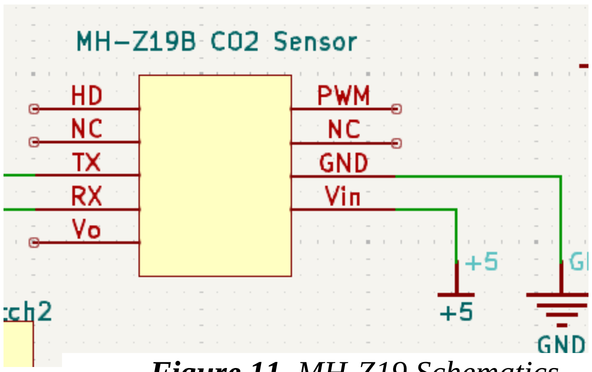

Figure 11. *MH-Z19 Schematics*

4.7 The VOC and Ozone Sensors – MP503 and MQ-131

In this section, we have two more gas sensors that we are using in our work for measuring Ozone *Figure 12* which is an indoor normal household gas generated by different devices like steamers, lamps that use ultraviolet light, or even air purifiers, while on the other hand, we have the other sensor which measures gases like smoke, alcohol, methanol, butane and more on. These sensors are semiconductor-heated metal oxide devices based on detecting changes in their resistance of the aimed gases as mentioned above. The amount or density of the gases present in our area of operation changes the resistance as a specific electrical current passes through the metal substrate constructing these components.

Our MQ131 gas sensor has high conductivity in clean air and high sensitivity to ozone, which means that when ozone gas rises, the sensor's conductivity will get lower while converting this change to the output signal for our system through its circuit. On the other hand, the MP503 sensor *Figure 15* is more about the air quality than the other sensor but the working principle is almost the same.

The MP503 sensor, also known as the MP503A or MP503B, is a type of gas sensor specifically designed to detect volatile organic compounds (VOCs) in the air. The heart of the MP503 sensor is a metal oxide semiconductor material, typically tin dioxide (SnO2). Integrated within the sensor is a heater element used to heat the metal oxide semiconductor to a specific temperature range (usually around 300-400°C). The sensor includes electrodes that measure the electrical resistance or conductivity changes of the metal oxide semiconductor when exposed to VOCs. These electrodes are connected to the sensor's circuitry to capture the sensor's response. The sensor’s circuitry processes the changes in electrical resistance or conductivity of the metal oxide semiconductor and converts these changes into an electrical signal that corresponds to the concentration of VOCs detected. The sensor is typically housed in a protective enclosure with openings or vents to allow air to reach the sensing element. It also includes connection points for power supply and output signals.

The MP503 VOC sensor operates on the principle of metal oxide semiconductors (MOS). It detects gases based on changes in electrical conductivity of a metal oxide semiconductor when exposed to VOCs in the air. The sensor includes a metal oxide semiconductor material, typically tin dioxide (SnO2), which acts as a sensing element. It contains an integrated heater element that heats the sensing element to a specific temperature (usually around 300-400°C). This heating is essential to facilitate the reaction between the target gases and the semiconductor surface. When VOCs or other reducing gases (such as ethanol, methane, benzene, etc.) are present in the air, they chemically react with the heated semiconductor surface, causing a change in its electrical conductivity. The change in conductivity alters the electrical resistance of the sensor. This resistance change is then converted into a measurable electrical signal.

For installation, the MP503 sensor requires specific wiring connections. Power connections include connecting the MP503 VCC to the Arduino 5V pin and the MP503 GND to the Arduino GND pin. For analog output connection, the MP503 OUT should be connected to the Arduino analog input pin (A1), which will read the analog voltage output from the sensor. Some sensors require a separate control signal to activate or adjust the heater element. The MP503 sensor typically operates at 5V logic levels, which is compatible with the Arduino Pro Mini (also 5V). Ensure all connections adhere to this voltage compatibility to prevent damage to components.

The MP503 sensor outputs an analog voltage signal that varies proportionally with the concentration of volatile organic compounds (VOCs) in the air. This analog signal is directly connected to one of the analog input pins on the Arduino. The Arduino uses its built-in Analog-to-Digital Converter (ADC) to convert the analog voltage from the MP503 sensor into a digital value. The ADC in Arduino provides a numerical value (typically between 0 and 1023 for a 10-bit ADC) corresponding to the voltage level received from the sensor. Once the Arduino reads the analog voltage (converted to a digital value), this value can be used directly in your code or converted into meaningful units (such as parts per million, ppm, if calibrated) based on the sensor's characteristics and calibration.

Table 5. The VOC sensor

Component	***Measurement Unit***
Component Name	MP503
Operational Voltage Inside the Loop	Less than 24V DC
The Voltage in Heating	From 5.0V up to 0.1V AC/DC
The Range of Detection	From 10 up to1000ppm
Time needed to Heat Up	48 hours
Pin Nr. 1&2	Electrode Heating
Pin Nr. 3&4	Electrode Measuring

The MQ131 ozone sensor operates on the principle of electrochemical sensing combined with a heating element. It contains an electrochemical cell that reacts with ozone molecules when they come into contact with the sensor’s surface. Like other MQ series sensors, the MQ131 has an internal heater (WO3) that is used to heat the sensing element to a specific temperature range (around 300-400°C). This heating process is crucial as it enhances the sensitivity and selectivity of the sensor by promoting chemical reactions on the sensor’s surface. When ozone molecules interact with the heated sensing element, they undergo a chemical reaction that causes a change in the sensor’s electrical conductivity or other measurable electrical properties. The change in conductivity or electrical properties due to the presence of ozone is then converted into a measurable electrical signal by the sensor’s circuitry.

For installation, connect the MQ131 VCC to the Arduino 5V pin and the MQ131 GND to the Arduino GND pin. The analog output connection involves connecting MQ131 OUT to the Arduino analog input pin (e.g., A0). This pin will read the analog voltage output from the sensor. Some versions of the MQ131 sensor may require a separate control signal to activate or adjust the heater element. Connect this to a digital output pin on the Arduino (e.g., D7) if needed. The MQ131 sensor typically operates at 5V logic levels, which matches the Arduino Pro Mini (also 5V).

The data transmission method for the MQ131 sensor involves providing an analog voltage output that corresponds to the ozone concentration it detects in the environment. This analog output signal varies linearly with changes in ozone concentration. To read data from the MQ131 sensor, connect its analog output pin (typically labeled as OUT or AOUT) to one of the analog input pins on the Arduino (e.g., A0, A1, etc.). Arduino's Analog-to-Digital Converter (ADC) then converts this analog voltage into a digital value representing the ozone concentration level.

The Arduino reads the analog voltage directly from the sensor and interprets it using its ADC capabilities.

Table 6. The Ozone sensor

Component	***Measuring Unit***
Component Name	MQ131
Operational Voltage Inside the Loop	From 5.0V up to 0.1V DC
The Voltage in Heating	From 5.0V up to 0.1V AC or DC
The Range of Detection	From 10 up to 1000ppb
Time needed to Heat Up	48 hours
Pin Nr. 1&2	Electrode Measuring
Pin Nr. 3&4	Electrode Heating

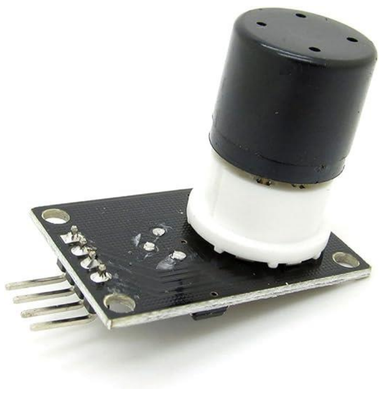

Figure 12. MQ-131

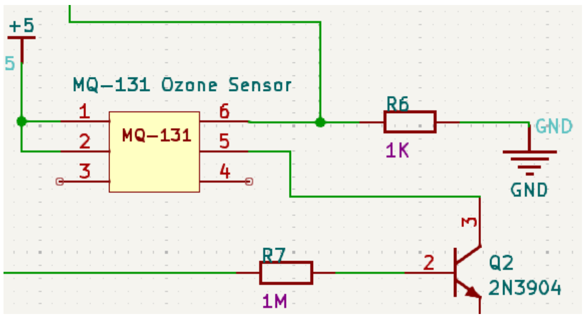

Figure 13. MQ-131 Schematics

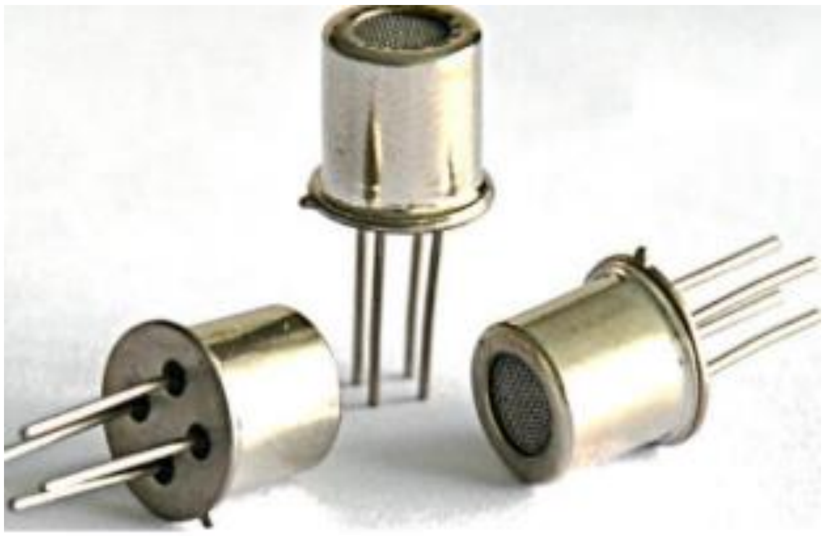

Figure 14. MP503

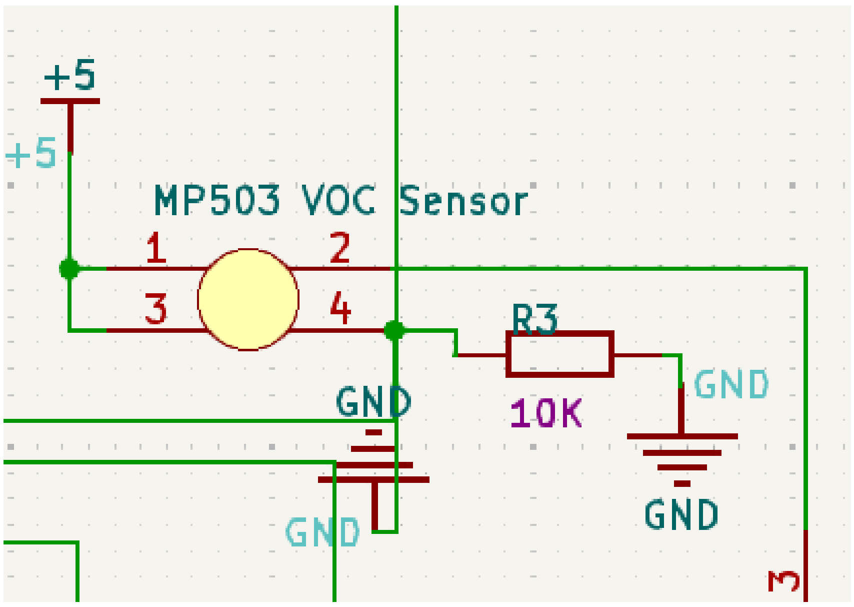

Figure 15. MP503 Schematics

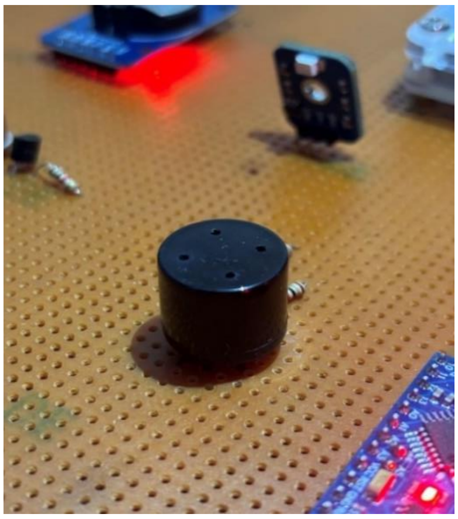

Figure 16. MP503 in Real Time Circuit Implementation

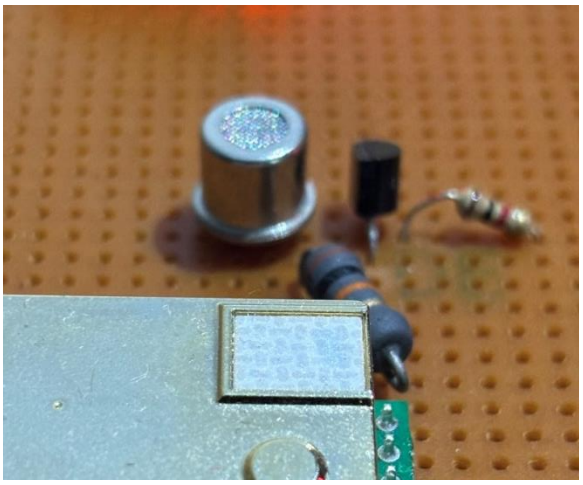

Figure 17. MQ131 in Real Time Circuit Implementation

4.1 DHT22 – Temperature and Humidity Sensor

The next sensor we will be using is the DHT22, which outputs a calibrated digital signal that is subsequently connected to and displayed on our screen. This sensor employs a technique that gathers digital signals, with its sensing elements linked to an 8-bit chip computer. Additionally, we chose to implement this sensor due to its reliability, stability, compact size, and low power consumption. On the other hand, these specifications in *Table 7* make it suitable for almost all kinds of occasions.

Table 7. The Temperature and Humidity Sensor

Component	***Measuring Unit & Definition***
Component Name	DHT22
The supplied Power	From 3.3 up to 6V DC
The Signal of Output	Digital
Level of Sensitivity	Hum. 0.1%RH; Temp. 0.1Celsius
The range of the operation	Hum. 0-100%RH; Temp. 4080Celsius
Pin Nr.1	VDD
Pin Nr.2	DATA
Pin Nr.3	NULL
Pin Nr.4	GND

The DHT22 sensor communicates with microcontrollers like Arduino using a proprietary communication protocol. The microcontroller initiates communication by sending a start signal to the DHT22. The DHT22 responds with a response signal to acknowledge the start signal. The sensor then sends 40 bits of data to the microcontroller: 16 bits for Humidity (in tenths of a percent, e.g., 533 means 53.3% RH), 16 bits for Temperature (in tenths of degrees Celsius, e.g., 215 means 21.5°C), and 8 bits of Checksum to verify data integrity. The microcontroller reads these bits, processes them, and converts them into meaningful temperature and humidity values for further use in applications.

For installation, we connect the VCC pin of the DHT22 sensor to the Arduino 5V pin, the GND pin to the Arduino GND pin, and the OUT pin to the Arduino digital pin (D5). We make sure to use a pull-up resistor (10kΩ) between the VCC and OUT pins of the DHT22 sensor to stabilize the data line.

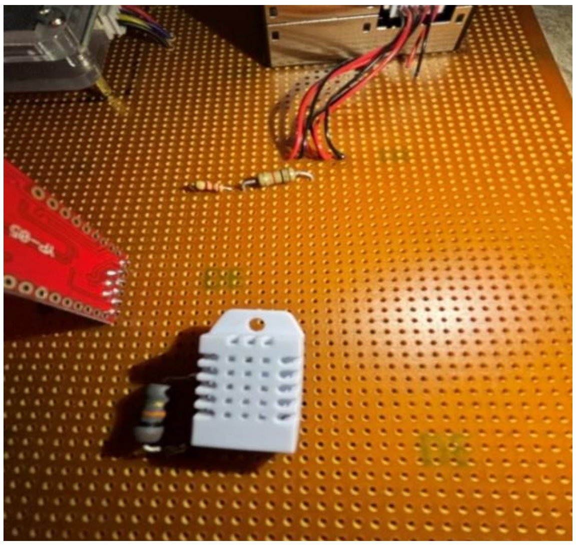
Figure 18. Real Time Implementation in Board

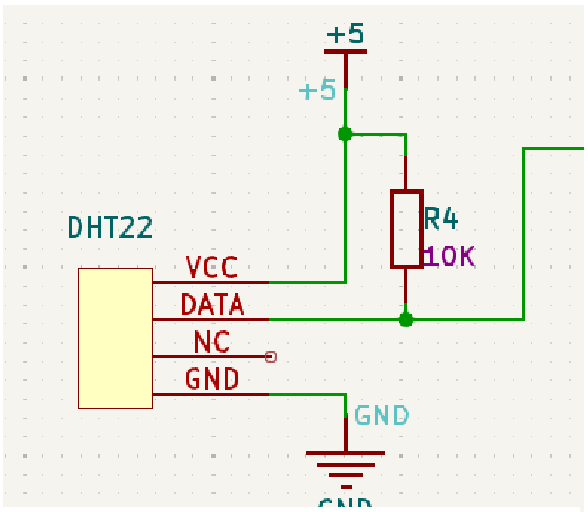

Figure 19. DHT22 Schematics

4.2 DS3231 RTC- Real Time Clock

Next, we will use an RTC component to track the values recorded by each sensor over 24 hours. We chose the DS3231 RTC because it is versatile and can be used in various applications such as telematics, GPS, and power meters. It is capable of counting seconds up to days of the week and year, utilizing a 400 kHz I2C interface.

Table 8. RTC- Real Time Clock

Parameter	***Index***
Product Model	DS3231
Supply voltage	From 2.3V up to 5.5V
Active supply current	From 200 up to 300 μA
Standby supply current	From 110 up to 170 μA
Active Battery Current	From 70 up to 150 μA
Timekeeping Battery Current	From 0.84 up to 3.5 μA

For the DS3231 RTC connections, we connect the VCC pin to the Arduino 5V pin, the GND pin to the Arduino GND pin, the SDA pin to the Arduino A4 pin (or the SDA pin on the Arduino Pro Mini), and the SCL pin to the Arduino A5 pin (or the SCL pin on the Arduino Pro Mini).

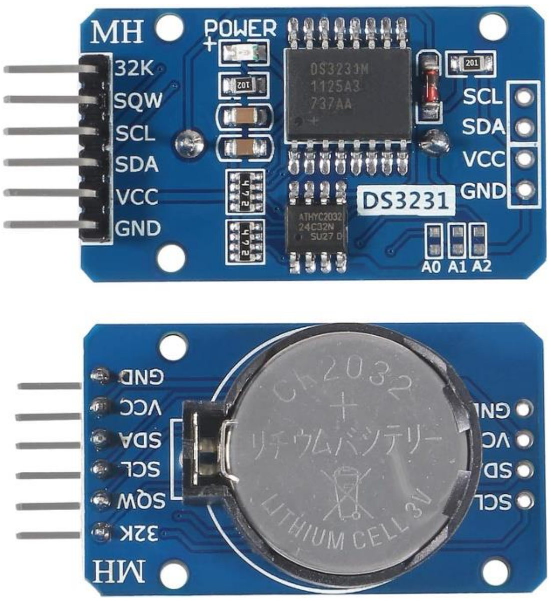

Figure 20. DS3231 RTC

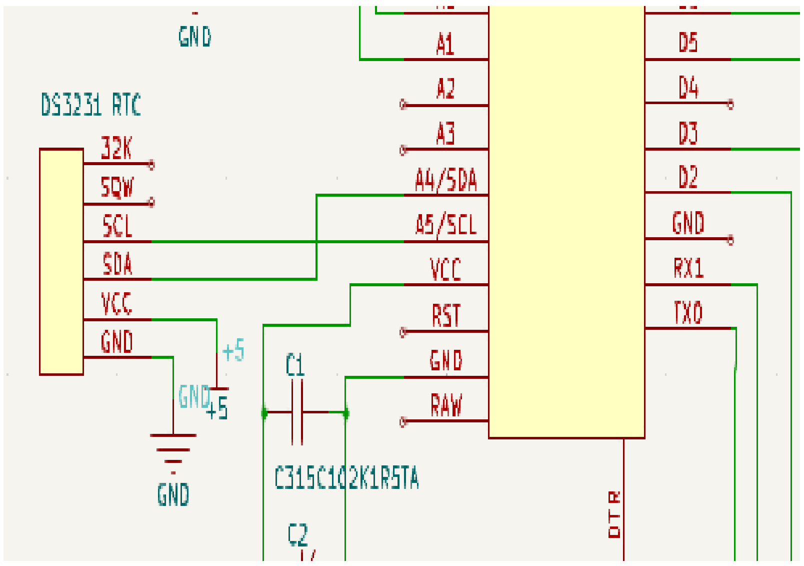

Figure 21. DS3231 RTC Schematics

4.3 NX3224T028 – Nextion Display

To monitor our recorded data, we will use the NX3224T028 Nextion display, which provides human control and a visualization interface. This display is a TFT LCD touchscreen display, measuring 2.8 inches diagonally with a resolution of 320x240 pixels. It features a resistive touchscreen capability and is equipped with a Nextion Intelligent Series Processor. The memory and storage include 4M flash storage and 3584B RAM. Communication with external microcontrollers is facilitated via a UART serial interface. The development environment for this display includes the Nextion Editor Software and a drag-and-drop interface. Power requirements for the display are typically 5V DC, and it is designed for low power consumption, making it suitable for battery-operated applications.

The data transmission for the NX3224T028 display utilizes a communication protocol that involves serial communication. The Arduino Pro Mini communicates with the Nextion display via UART (Serial). Commands are sent as strings or byte arrays representing actions like setting text, updating variables, or handling touch events. Commands to the Nextion display typically end with three bytes (0xFF, 0xFF, 0xFF) to signify the end of the command packet. The Nextion display can also send responses, such as touch event notifications or acknowledgments for commands sent.

Table 9. The Nextion Display

Parameter	***Index***
Product Model	NX3224T028
Operating voltage	From 4.75V up to 7V
Operating current	65 mA
The recommended supplied power：5V, 500mA, DC	

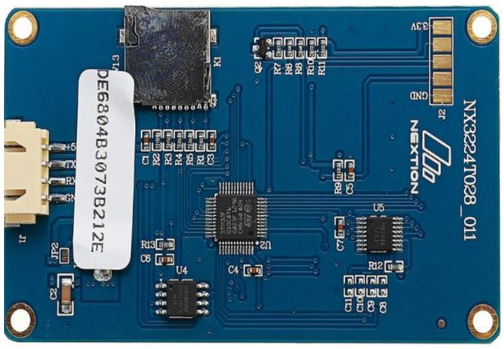

Figure 22.Nextion Display

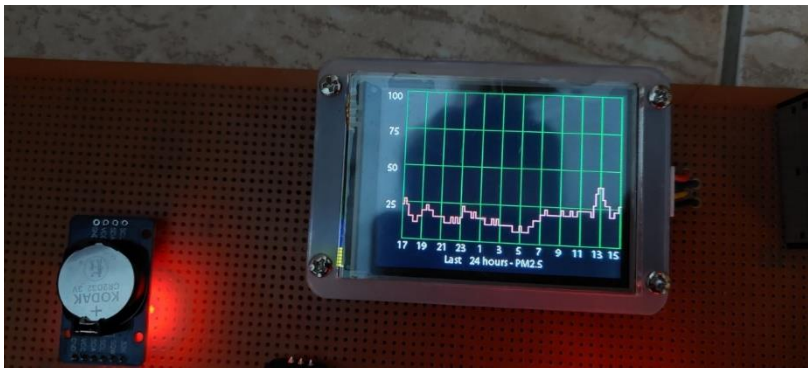

Figure 23. Real Time Graph Display

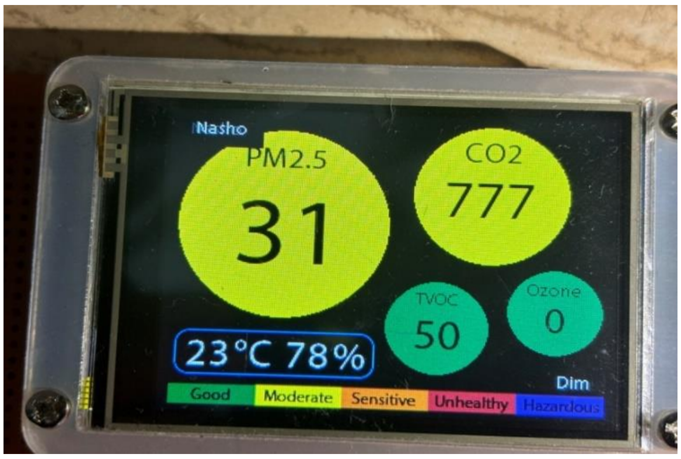

Figure 24. Real Time Implementation in Board

4.4 Arduino Pro Mini – Microcontroller

The Arduino Pro Mini 5V is a compact, low-power microcontroller featuring the ATmega328P with a 16 MHz clock speed, 32 KB flash memory, 2 KB SRAM, and 1 KB EEPROM. It operates at 5V and has 14 digital I/O pins (6 PWM), 6 analog input pins, and supports UART, SPI, and I2C communication. When powered, the voltage regulator ensures a stable 5V supply, and the ATmega328P initializes the stored program. The microcontroller reads analog and digital signals from sensors and switches, processes the input data according to sketch logic for computations and decision-making, and controls devices through digital, PWM signals, or serial communication based on the processed data.

For installation, we connect a power source to the RAW (6-12V) or VCC (5V) pin. We use an FTDI adapter connected to the GND, VCC, RX, and TX pins to upload code. In the Arduino IDE, we select "Arduino Pro or Pro Mini," choose ATmega328P 5V 16MHz, select the COM port, and upload the sketch.

Table 10. Arduino Pro Mini

Component	***Definition***
Name	ATmega328P
Supply power	(5V model)
Operating Voltage	5V
Clock	16 MHz
Memory	32KB

For uploading code, we select “Arduino Pro or Pro Mini” in the Arduino IDE under Tools -> Board. We choose the correct processor (ATmega328P 5V 16MHz), select the correct COM port, and upload the sketch.

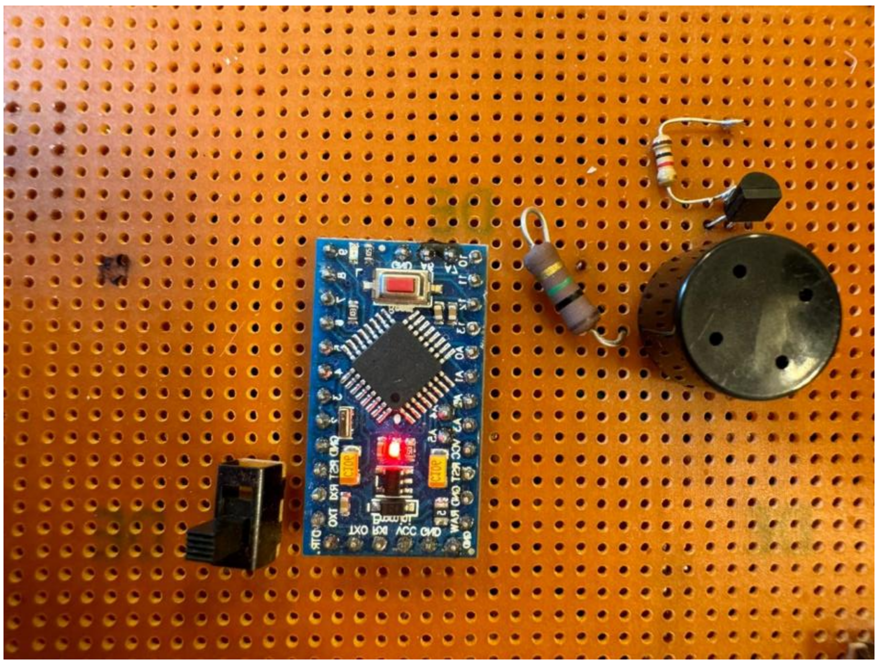

Figure 25. Arduino Real Time Implementation in Board

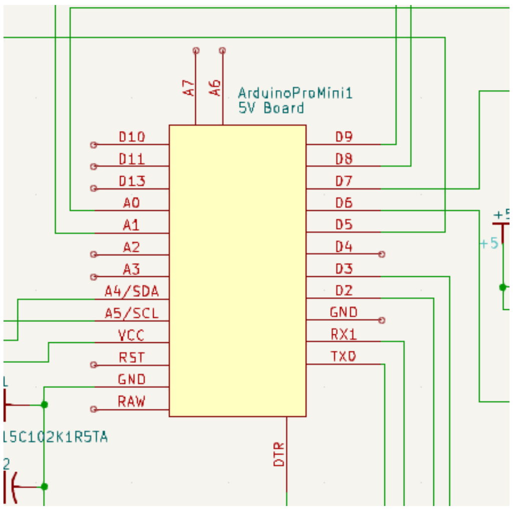

Figure 26. Arduino Schematics

4.5 LDR4589

This specific sensor that we use detects light density or darkness around the area which we are investigating, proving adjustable digital or analog output trigger level, suitable for different systems such as lighting, alarm etc.

Table 11. LDR

Component	***Definition***
Product Model	LDR4589
Operating voltage	From 3.3V to 5V DC
Operating current	15 mA
Output digital	0V to 5V
Pin Nr.1	AO
Pin Nr.2	DO
Pin Nr.3	GND
Pin Nr.4	VCC

For the installation, we connect one end of the LDR to the 5V pin of the Arduino. We connect the other end of the LDR to one end of the fixed resistor. We connect the junction of the LDR and the fixed resistor to an analog input pin of the Arduino (A0). Finally, we connect the free end of the fixed resistor to the GND pin of the Arduino.

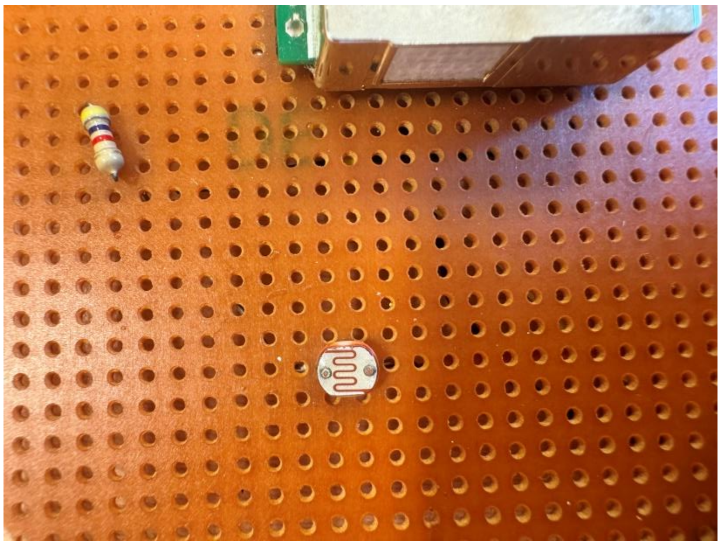

Figure 27. Real Time LDR Impelemtation

4.6 UV detection sensor

The last sensor that we will use is a UV detection sensor which is essential for monitoring ultraviolet radiation and truly crucial in industrial processes, consumer electronics, etc. They provide accurate data for ensuring safety and healthy environments when they are exposed to radiation. Those sensors' operations rely on converting the radiation or light into an electrical signal using a photodiode made from truly sensitive materials to light. The operation results in a current that is proportional to the intensity of the light which is generated by electron-hole pairs of the device system.

For the installation, we connect the VCC to the 5V pin of the Arduino, the GND to the GND pin of the Arduino, and the Analog Output (AOUT) to an analog input pin of the Arduino.

Table 12. UV Sensor

Component	***Definition***
Product Model	UV sensor 8521
Operating voltage	From 3.3 to 5 DC
Current	From 0.06 to 0.1 mA
Output	0V to 1V DC
Wavelength	From 200 up to 370nm
Pin Nr. 1	OUT
Pin Nr. 2	VCC
Pin Nr. 3	GND

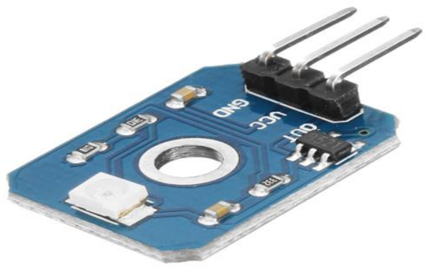

Figure 28. UV Sensor

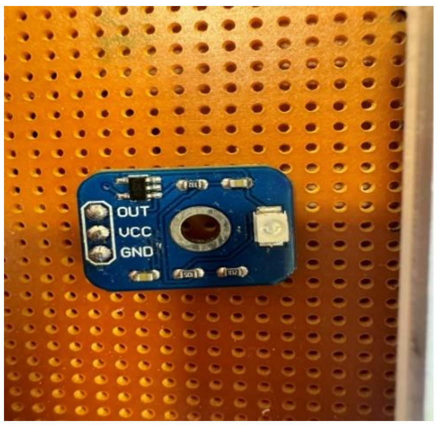

Figure 29. Real Time Implementation

CHAPTER 5

SOFTWARE IMPLEMENTATION

5.1 Introduction

The software implementation of our system is divided into two parts, the first part is implemented in Arduino Ide which will be useful to execute all the necessary commands and functions to connect our microcontroller with all the other parts of the system, while the second part we will use Nextion Editor to implement the code for the display. For the second part, my collaborator and classmate Dorian Dafku handled the software implementation.

5.2 Arduino Ide Code Implementation

The final code of the system will be included in the appendix part.

For the calibration part, we conducted several Arduino code implementations to observe how our sensors reacted to different situations and physical phenomena, and to test their limits. Our system is programmed to test and record data over a 24-hour cycle, which can be displayed using graphs, as shown in the images below. We used the Nextion Editor to assign variables for each of the parameters recorded by the sensors and incorporated different color ranges in the gadgets to represent danger levels and raise awareness.

OZONE SENSOR CALIBRATION:

```
#include <MQ131.h>

void setup() {
  Serial.begin(115200);

  // Init the sensor
  // - Heater control on pin 6
  // - Sensor analog read on pin A0
  // - Model LOW_CONCENTRATION
  // - Load resistance RL of 1MOhms (20000 Ohms)
  MQ131.begin(6,A0, LOW_CONCENTRATION, 1000000);

  Serial.println("Calibration parameters");
  Serial.print("R0 = ");
  Serial.print(MQ131.getR0());
  Serial.println(" Ohms");
  Serial.print("Time to heat = ");
  Serial.print(MQ131.getTimeToRead());
  Serial.println(" s");
}

void loop() {
  Serial.println("Sampling...");
  MQ131.sample();
  Serial.print("Concentration O3 : ");
  Serial.print(MQ131.getO3(PPM));
  Serial.println(" ppm");
  Serial.print("Concentration O3 : ");
  Serial.print(MQ131.getO3(PPB));
  Serial.println(" ppb");
  Serial.print("Concentration O3 : ");
  Serial.print(MQ131.getO3(MG_M3));
  Serial.println(" mg/m3");
  Serial.print("Concentration O3 : ");
  Serial.print(MQ131.getO3(UG_M3));
  Serial.println(" ug/m3");

  delay(60000);
}
```

Table 13. Ozone Calibration

Time (s)	Concentration O3 (ppm)	Concentration O3 (ppb)	Concentration O3 (mg/m³)
80	7.79	7789.62	16.46
160	11.94	11941.79	25.24
240	15.32	15320.62	32.38
320	19.22	19222.25	40.63
400	23.68	23675.72	50.04
480	28.71	28710.62	60.68
560	32.89	32885.86	69.50
640	39.01	39009.21	82.45
720	44.04	44035.27	93.07
800	47.60	47599.49	100.60
880	53.28	53275.82	112.60
960	59.36	59359.90	125.46
1040	68.13	68129.26	143.99
1120	70.44	70442.24	148.88

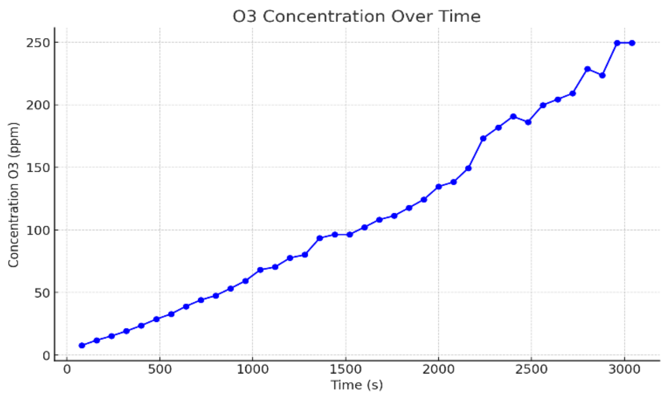

Figure 30. Ozone Concentration Graph Plot

PARTICULATE MATTER SENSOR TEST CODE:

```
#include <SoftwareSerial.h>

#include "PMS.h"


// Define software serial pins

#define RX_PIN 8

#define TX_PIN 9


// Create a SoftwareSerial instance

SoftwareSerial mySerial(RX_PIN, TX_PIN);


// Initialize PMS instance

PMS pms(mySerial);

PMS::DATA data;


void setup()

{

  // Initialize the software serial communication

  mySerial.begin(9600);


  // Initialize the hardware serial communication for debugging

  Serial.begin(9600);

}


void loop()

{
  if (pms.read(data))

  {

    Serial.print("PM 1.0 (ug/m3): ");

    Serial.println(data.PM_AE_UG_1_0);


    Serial.print("PM 2.5 (ug/m3): ");

    Serial.println(data.PM_AE_UG_2_5);


    Serial.print("PM 10.0 (ug/m3): ");

    Serial.println(data.PM_AE_UG_10_0);


    Serial.println();

  }


  // Do other stuff...

}
```

Table 14. PM Sensor Test

Time (s)	PM 1.0 (µg/m³)	PM 2.5 (µg/m³)	PM 10.0 (µg/m³)
1	10	10	10
2	12	13	13
3	14	15	15
4	13	14	14
5	13	14	14
6	13	16	16
7	13	16	16
8	13	17	17
9	16	20	20
10	18	24	24
11	20	29	30
12	21	31	34
13	26	39	44
14	28	41	48
15	30	46	54
16	32	48	57
17	33	50	60
18	34	52	62
19	34	52	63

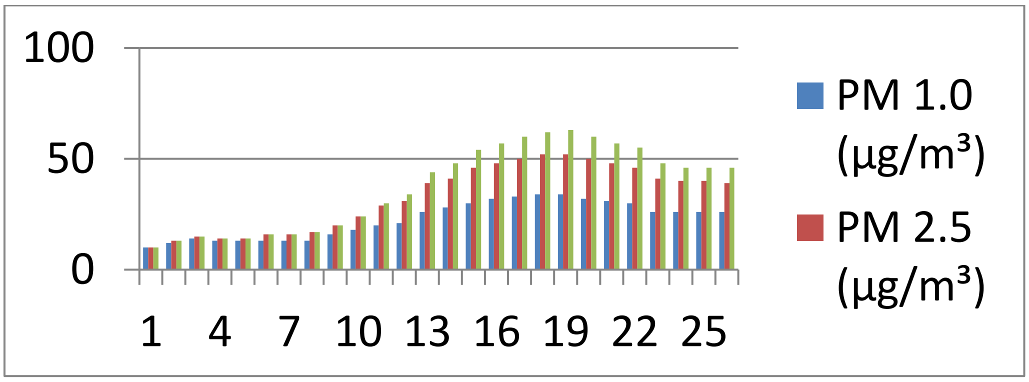

Figure 31. Testing of PM2.5 – Graph Plotting

DHT22 TEST CODE:

```
#include <dht.h>


dht DHT;


#define DHT22_PIN     5


struct
{
    uint32_t total;
    uint32_t ok;
    uint32_t crc_error;
    uint32_t time_out;
    uint32_t connect;
    uint32_t ack_l;
    uint32_t ack_h;
    uint32_t unknown;
} stat = { 0,0,0,0,0,0,0,0};


void setup()
{
    Serial.begin(115200);
    Serial.println("dht22_test.ino");
    Serial.print("LIBRARY VERSION: ");
    Serial.println(DHT_LIB_VERSION);
    Serial.println();
```

```
  Serial.println("Type,\tstatus,\tHumidity (%),\tTemperature (C)\tTime (us)");
}

void loop()
{
  // READ DATA
  Serial.print("DHT22, \t");

  uint32_t start = micros();
  int chk = DHT.read22(DHT22_PIN);
  uint32_t stop = micros();

  stat.total++;
  switch (chk)
  {
  case DHTLIB_OK:
    stat.ok++;
    Serial.print("OK,\t");
    break;
  case DHTLIB_ERROR_CHECKSUM:
    stat.crc_error++;
    Serial.print("Checksum error,\t");
    break;
  case DHTLIB_ERROR_TIMEOUT:
    stat.time_out++;
    Serial.print("Time out error,\t");
    break;
case DHTLIB_ERROR_CONNECT:
    stat.connect++;
    Serial.print("Connect error,\t");
    break;
case DHTLIB_ERROR_ACK_L:
    stat.ack_l++;
    Serial.print("Ack Low error,\t");
    break;
case DHTLIB_ERROR_ACK_H:
    stat.ack_h++;
    Serial.print("Ack High error,\t");
    break;
default:
    stat.unknown++;
    Serial.print("Unknown error,\t");
    break;
}
// DISPLAY DATA
Serial.print(DHT.humidity, 1);
Serial.print(",\t");
Serial.print(DHT.temperature, 1);
Serial.print(",\t");
Serial.print(stop - start);
Serial.println();
```

```
    if (stat.total % 20 == 0)
    {
        Serial.println("\nTOT\tOK\tCRC\tTO\tCON\tACK_L\tACK_H\tUNK");
        Serial.print(stat.total);
        Serial.print("\t");
        Serial.print(stat.ok);
        Serial.print("\t");
        Serial.print(stat.crc_error);
        Serial.print("\t");
        Serial.print(stat.time_out);
        Serial.print("\t");
        Serial.print(stat.connect);
        Serial.print("\t");
        Serial.print(stat.ack_l);
        Serial.print("\t");
        Serial.print(stat.ack_h);
        Serial.print("\t");
        Serial.print(stat.unknown);
        Serial.println("\n");
    }
    delay(2000);
}
```

```
DHT22,  OK,     99.9,   32.4,   5020
DHT22,  OK,     99.9,   32.3,   5020
DHT22,  OK,     99.9,   32.2,   4976
DHT22,  OK,     99.9,   32.1,   4928
DHT22,  OK,     99.9,   32.0,   4928
DHT22,  OK,     99.8,   32.0,   4836
DHT22,  OK,     98.1,   31.9,   5020
DHT22,  OK,     96.7,   31.8,   4976
DHT22,  OK,     94.1,   31.8,   5204
DHT22,  OK,     92.6,   31.7,   5204
DHT22,  OK,     90.5,   31.7,   4976
DHT22,  OK,     89.3,   31.7,   5208
DHT22,  OK,     87.0,   31.6,   4972
DHT22,  OK,     85.5,   31.6,   5068
DHT22,  OK,     83.4,   31.6,   4792
DHT22,  OK,     82.0,   31.5,   5020
DHT22,  OK,     80.0,   31.5,   4976
DHT22,  OK,     78.8,   31.5,   4928
DHT22,  OK,     77.4,   31.4,   4792
DHT22,  OK,     76.8,   31.4,   4836

TOT     OK      CRC     TO      CON     ACK_L   ACK_H   UNK
60      60      0       0       0       0       0       0
```

Figure 32. DHT22 Test

MHZ19 SENSOR CALIBRATION CODE:

```
#include <Arduino.h>

#include "MHZ19.h"

#include <SoftwareSerial.h>

#define RX_PIN 3   // RX pin for SoftwareSerial (to MH-Z19 TX)

#define TX_PIN 2   // TX pin for SoftwareSerial (to MH-Z19 RX)

#define BAUDRATE 9600

MHZ19 myMHZ19;

SoftwareSerial mySerial(RX_PIN, TX_PIN);  // Create SoftwareSerial instance

unsigned long timeElapse = 0;
```

```
void setup() {
   Serial.begin(9600);          // Initialize serial communication for debugging
   mySerial.begin(BAUDRATE);     // Initialize software serial for MH-Z19
   myMHZ19.begin(mySerial);      // Pass software serial to MH-Z19 library

   myMHZ19.autoCalibration(false);  // Disable auto calibration
   bool abcStatus = myMHZ19.getABC();
   Serial.print("ABC Status: ");
   Serial.println(abcStatus ? "ON" : "OFF");  // Print ABC status

   Serial.println("Waiting 20 minutes to stabilize...");
   timeElapse = 20UL * 60UL * 1000UL;  // 20 minutes in milliseconds
   delay(timeElapse);  // Wait for the sensor to stabilize

   Serial.println("Calibrating..");
   myMHZ19.calibrate();  // Calibrate the sensor
}
void loop() {
   if (millis() - timeElapse >= 2000) {  // Check if 2 seconds have passed
      int CO2 = myMHZ19.getCO2();  // Get CO2 concentration

      if (CO2 == -1) {
         Serial.println("!Error: Failed to get CO2 reading.");
      } else {
         Serial.print("CO2 (ppm): ");
         Serial.println(CO2);
```

```
        }
        int8_t Temp = myMHZ19.getTemperature();  // Get temperature

        if (Temp == -1) {
          Serial.println("!Error: Failed to get temperature reading.");
        } else {
          Serial.print("Temperature (C): ");
          Serial.println(Temp);
        }

        timeElapse = millis();  // Reset the timer
    }
}
```

CHAPTER 6

TESTING PERFORMANCE

6.1 Introduction

As we can see in *Figure 37* for that period our sensors have detected that: the purity of the air is quite acceptable at 29 PM2.5, the temperature was 22, humidity at 42%, VOV at 84, and Ozone 42 inactive since it needed 48 hours to turn on, CO2 is extremely normal at 541. In the display it also includes a range of colors with additional descriptions to better understand the environmental status even by looking only at the gadgets.

6.2 Testing and Results for a cycle of 24 hours

Below we will display and observe the recordings done of these environmental monitoring system components during a 24-hour shift at a stationary position.

As we can see from the graph the PM2.5 or the particle sensor monitoring has changed quite a lot during the 24-hour interval starting from 25 and ending at a higher level. During this measurement, we observed that during the night the sensor recorded the lowest values approximately 15 while during the day it reached values up to 40. This substantial variance shows that particle concentration levels vary greatly during the day, presumably due to variations in human activity, transportation, and environmental variables. The lower readings measured at night could be ascribed to less outside activity and traffic, resulting in cleaner air. Higher levels throughout the day, on the other hand, are most likely due to increased particulate-generating activities, such as automobile emissions and industrial operations.

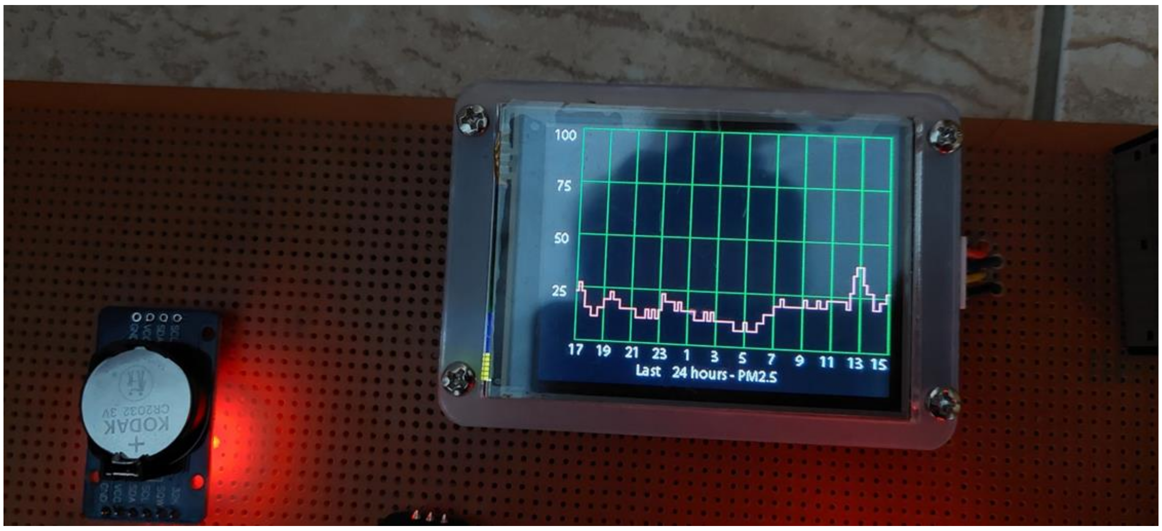

Figure 33. PM2.5 Graph Representation of 24 Hours Measurement

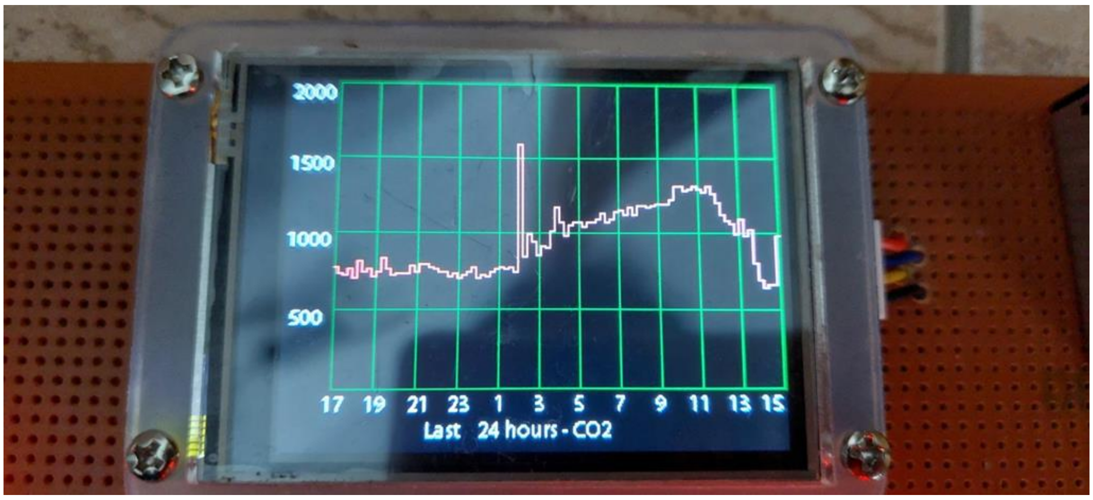

Figure 34. CO2 Graph Representation of 24 Hours Measurement

The visualization graph demonstrates that the CO2 sensor's results vary widely over time. The highest value recorded throughout this experiment, which was carried out under difficult conditions, was above 2000 ppm. In our steady scenario, nevertheless, the maximum value was approximately 1600 ppm, as well as a minimum of 600 ppm. This wide range of data suggests that CO2 concentrations fluctuate greatly, most likely due to environmental and human activity changes.

The result of this measurement is different compared to the other sensors since as we can distinguish the graph distribution is smoother and the range or difference between the values of this measurement conducted is very close to each other.

The minimum value of the VOC sensor is around 35 while the maximum value is around 55. This small range and smooth distribution indicate that VOC levels remain generally steady over time, with few variations.

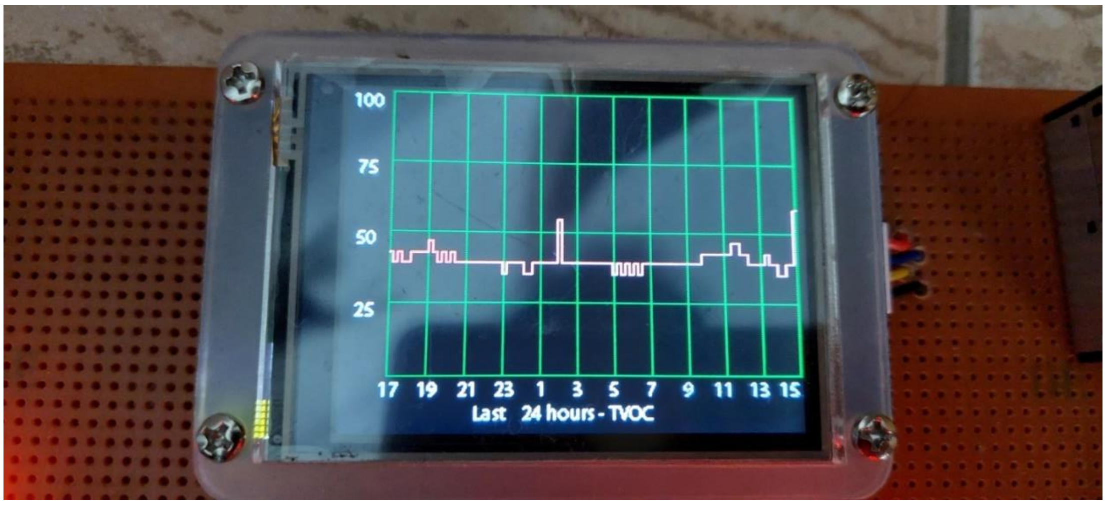

Figure 35. TVOC Graph Representation of 24 Hours Measurement

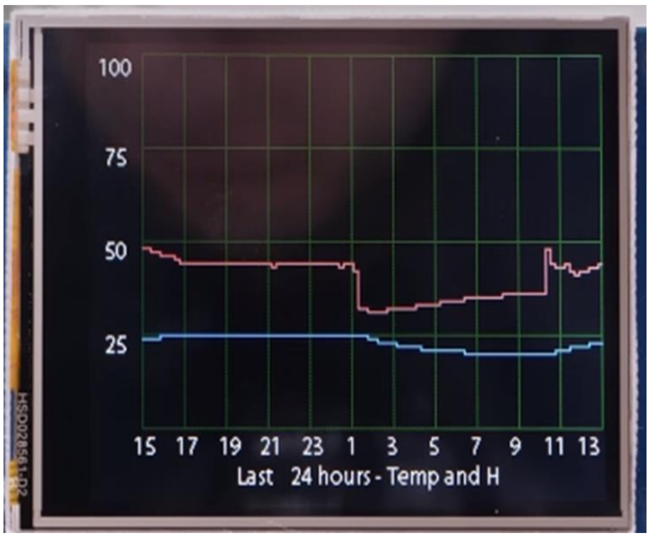

Figure 36. Temperature and Humidity 24 Hours Measurement

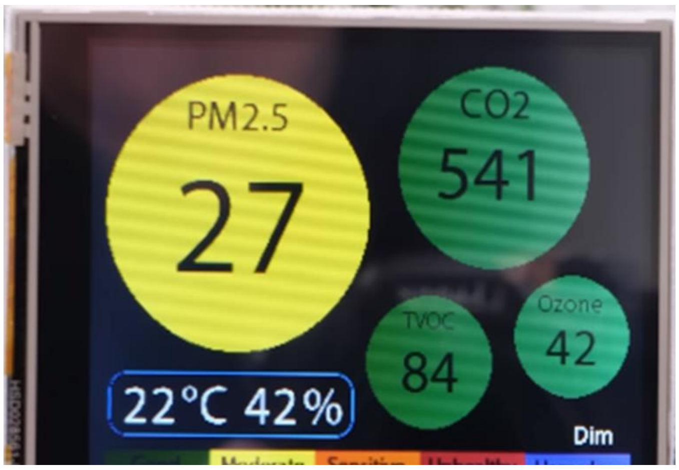

Figure 37. Display Color Range and Parameters Testing

CHAPTER 7

CONCLUSIONS

7.1 Conclusions

During the operation of this project, we were able to sketch detailed schematics and the circuit and we were able to test it in different places.

The devices' programming was performed using the integration of the Nextion Display and Arduino Pro Mini in Arduino Ide and Nextion Editor.

The successful implementation of this environmental monitoring station demonstrates its potential as a reliable tool for air quality assessment, paving the way for broader applications in environmental monitoring and public health initiatives.

Table 15. Power Consumption and Current Needed

Component	***Voltage (V)***	***Current (mA)***	***Power (mW)***
PMS5003	5	100	500
MH-Z19	5	60	300
MQ-131	5	150	750
MP503	5	150	750
DHT22	5	2.5	12.5
DS3231	5	0.3	1.5
Nextion NX3224T028	5	65	325
BMP180	3.3	0.003	0.0099
LDR4589	5	15	75
UV 8521	5	0.1	0.5

Total Current Needed: 543 mA

Total Power Consumption: 2.715 W

Table 16. Total Price

Component	*Price ($)*
PMS5003	13.80
MH-Z19	18.71
MQ-131	12.10
MP503	6.05
DHT22	4.47
DS3231	5.79
Nextion NX3224T028	40.45
Standoff Support Spacers	11.10
LDR4589	3.24
UV 8521	6.15
Arduino Pro Mini	7.20
Programming Header for Arduino Pro Mini	11.10
Mini USB Type B	3.71
Adapter 5V	10
Wires	20
Universal PCB	10
Case	30
Port for Adapter	1
2pcs Three Terminal Switch	1
Resistors, Transistors, Capacitors	2
SD Card	5
SD Card to USB adapter	1
Total	**238.32**

The cost of the device does not include the purchase of a multimeter, soldering iron, solder, flux, wick, desoldering pump, or transportation for each part. It also excludes the cost of the MQ131 sensor with fast delivery, which is quite expensive, the Arduino connector cables, and the sensors that are not installed, such as the barometric sensor, raindrop sensor, and soil moisture sensor.

Table 17. *Frequency and Data Acquisition Rate*

Component	*Data Acquisition Rate (Hz)*
PMS5003	1
MH-Z19	1
MQ-131	0.008 (1 per 2 min)
MP503	0.008 (1 per 2 min)
DHT22	0.5
DS3231	400 kHz (I2C)
Nextion NX3224T028	As needed
BMP180	7.5
LDR4589	Continuous analog output
UV 8521	Continuous analog output

Data Recording Frequency: 1 Hz (once per second)

Table 18. Calibration

Component	*Calibration Status*
PMS5003	Factory calibrated; requires periodic field calibration after assembly
MH-Z19	Factory calibrated; zero and span calibration recommended after installation
MQ-131	Requires calibration in clean air and target gas environment post-assembly
MP503	Requires baseline and target gas calibration post-assembly
DHT22	Factory calibrated; no additional calibration needed after installation
DS3231	Factory calibrated; no additional calibration needed
Nextion NX3224T028	Not applicable
BMP180	Factory calibrated; no additional calibration needed after installation
LDR4589	Requires calibration for ambient light levels after installation
UV 8521	Requires calibration for UV index after installation

7.2 Recommendations for future research

As for future work, we hope to add additional sensors to this system, allowing it to be fuller and more intelligent. Along with to our scientific trials, we will take additional measurements in various locations within our campus environment, so that our data compiled and shown on our monitor will raise people's awareness of the state of their community.

In addition, by linking the system to the Internet of Things systems, users can get immediate data evaluation and distant monitoring of air purity via Internet or smartphone apps from any location. Using artificial intelligence algorithms to forecast air quality trends and identify anomalies can provide more insight into the factors that influence air quality and aid in the quick identification of pollution occurrences.

The monitoring station's usefulness will be increased by installing additional sensors to the collection to monitor additional outside parameters such as levels of noise, nitrogen oxides (NO2), and sulfur dioxide (SO2). The whole thing can be enhanced environmentally friendly and suitable for remote deployments by minimizing power consumption with energy-efficient components, offering sleep modes, and researching the utilization of sources of clean energy such as solar panels.

To ensure the long-term precision and dependability of monitoring data, automatic calibration algorithms will need to be developed, as well as the usage of more accurate sensors. The Nextion Display's user experience might be improved by introducing more interactive elements such as customizable settings, alarms, and historical data visualization.

Including monitoring data in policy frameworks, in collaboration with lawmakers and environment agencies, would help to foster more information-driven choices and regulatory compliance. Further availability will also be secured through inquiry into ways to reduce the expenses associated with the surveillance station while keeping it scalable for bigger deployments, such as using low-cost components and manufacturing economies of scale.

.

REFERENCES

[1] A. Bushnag, "Air Quality and Climate Control Arduino Monitoring System using Fuzzy Logic for Indoor Environments," *IEEE,* pp. 1-6, 2021.

[2] R. K. Jha, "“Air Quality Sensing and Reporting System Using IoT”," *IEEE,* pp. 790-793, 2020.

[3] Y. M. F. R. T. A. M. S. M. E. M K Fadzly, "Smart Air Quality Monitoring System Using Arduino Mega," *IOPScience,* vol. 864, pp. 1-7, 2020.

[4] B. K. Moharana, P. Anand, S. Kumar and P. Kodali, "Development of an IoT-based Real-Time Air Quality Monitoring Device," *IEEE,* pp. 191-194, 2020.

[5] L. Fraiwan and A. M. Rajab, "Smart Indoor Environment Monitoring System," *IEEE,* pp. 1-4, 2020.

[6] T. Manglani, A. Srivastava, A. Kumar and R. Sharma, "IoT Based Air and Noise Pollution Monitoring System," *IEEE,* pp. 604-607, 2021.

[7] M. Lobur, D. Korpyljov, N. Jaworski, M. Iwaniec and U. Marikutsa, "Arduino Based Ambient Air Pollution Sensing System," *IEEE,* pp. 32-35, 2020.

[8] N. B. E. S. A. B. I Ardiansah, "Design of Micro-Climate Data Monitoring System for Tropical Greenhouse based on Arduino UNO and Raspberry Pi," *IOP science,* vol. 757, no. 1, p. 012017, 2020.

[9] A. Chaturvedi and L. Shrivastava, "IOT Based Wireless Sensor Network for Air Pollution Monitoring," *IEEE,* pp. 78-81, 2020.

[10] N. Nowshin and M. S. Hasan, "Microcontroller Based Environmental Pollution Monitoring System though IoT Implementation," *IEEE,* pp. 493-498, 2021.

[11] A. Géczy, L. Kuglics, L. Jakab and G. Harsányi, "Wearable Smart Prototype for Personal Air Quality Monitoring," *IEEE,* pp. 84-88, 2020.

[12] M. N. Bhuiyan, M. M. Billah and F. Bhuiyan, "Design and Implementation of a Feasible Model for the IoT Based Ubiquitous Healthcare Monitoring System for Rural and Urban Areas," *IEEE,* pp. 91984-91997, 2020.

[13] S. L. K. K. Kazi Sultanabanu, "Arduino-Based Weather Monitoring System," *ResarchGate,* pp. 24-29, 2023.

[14] Q. I. Sarhan, "Arduino Based Smart Home Warning System," *IEEE,* pp. 201-206, 2020.

[15] N. M. M. R. Pedro F. Pereira, "Low-cost Arduino-based temperature, relative humidity and CO2 sensors - An assessment of their suitability for indoor built environments," *ResearchGate,* p. 105151, 2022.

[16] C. C. Paglinawan, G. M. Cruz and K. R. M. D. Villar, "Design of an Arduino-Powered Sleep Monitoring System Based on Electrooculography (EOG) with Temperature Control Applications," *IEEE,* pp. 297-302, 2022.

[17] E. L. A. Puput Wanarti Rusimamto, "Implementation of arduino pro mini and ESP32 cam for temperature monitoring on automatic thermogun IoT-based," *ResearchGate,* pp. 1366-1375, 2020.

APPENDIX

```
#include "SoftwareSerial.h"

#include "MHZ19.h"   // https://github.com/WifWaf/MH-Z19

#include "PMS.h"     //https://github.com/fu-hsi/pms

#include "MQ131.h"   // https://github.com/ostaquet/Arduino-MQ131-driver

#include "dht.h"     // https://github.com/RobTillaart/DHTlib

#include "DS3231.h"  // http://www.rinkydinkelectronics.com/library.php?id=73


#define led 13

#define tvocPin 7  // VOC sensor activation

#define dht22 5 // DHT22 temperature and humidity sensor

#define ldrPin A7 // LDR sensor pin

#define uvPin A6 // UV sensor pin


dht DHT; // Creates a DHT object

DS3231  rtc(SDA, SCL); // Initiate the DS3231 Real Time Clock module using the
I2C interface

Time  t; // Init a Time-data structure

MHZ19 myMHZ19;    // CO2 Sensor

SoftwareSerial co2Serial(2, 3);  // (RX, TX) MH-Z19 serial

SoftwareSerial pmsSerial(8, 9); // Particulate Matter sensor

PMS pms(pmsSerial);

PMS::DATA data;


unsigned long dataTimer = 0;
unsigned long dataTimer3 = 0;

unsigned long dataTimer4 = 0;

int readDHT, temp, hum;

int CO2;

int o3;

int tvoc;

int pm25;

int ldrValue; // LDR value

int ldrPercent; // LDR percentage value

int uvValue; // UV sensor value

int hours, minutes;

int previousMinutes = 1;

String timeString;

String receivedData = "Z";


// We store the last 24 hours sensor values  in arrays - store value each 15 minutes so for 24 hours we need 96 bytes.

// We must use bytes and can't increase the storing to let's say 5 mins because the Arduino Pro Mini has a limited dynamic memory

uint8_t tempData[96] = {};

uint8_t humData[96] = {};

uint8_t tvocData[96] = {};

uint8_t co2Data[96] = {};

uint8_t pm25Data[96] = {};

uint8_t o3Data[96] = {};

uint8_t ldrData[96] = {}; // Array to store LDR values

uint8_t uvData[96] = {}; // Array to store UV sensor values
```

```
int8_t last24Hours[12] = {};
int yAxisValues[4] = {};
int maxV = 0;
int8_t r = 99;

void setup() {
  Serial.begin(9600);
  // Device to serial monitor feedback
  pinMode(6, OUTPUT);
  pinMode(tvocPin, OUTPUT);

  // Warming up sensors
  digitalWrite(6, HIGH);       // Ozone sensor
  digitalWrite(tvocPin, HIGH);  // TVOC sensor
  delay(20 * 1000); // delay 20 seconds
  digitalWrite(6, LOW);
  digitalWrite(tvocPin, LOW);

  // Initialize all sensors
  rtc.begin();
  co2Serial.begin(9600);
  pmsSerial.begin(9600);
  myMHZ19.begin(co2Serial);
    myMHZ19.autoCalibration(false);    // Turn auto calibration ON (OFF
autoCalibration(false))
```

```
  MQ131.begin(6, A0, LOW_CONCENTRATION, 1000000); //
  MQ131.setTimeToRead(20); // Set how many seconds we will read from the Ozone sensor. It blocks flow
  MQ131.setR0(9000); // We get this value using the calibrate() function from the Library calibration example
}

void loop() {
  // Read temperature and humidity from DHT22 sensor
  readDHT = DHT.read22(dht22); // Reads the data from the sensor
  temp = DHT.temperature; // Gets the values of the temperature
  hum = DHT.humidity; // Gets the values of the humidity

  // Read TVOC for 5 seconds
  digitalWrite(tvocPin, HIGH);
  delay(5000); // Blocking the program - It would be best if the sensor heater is active all the time, we would get the most accurate values that way. The thing is that the sensors heat up quite a lot and mess with the temperature values. If better air circulation is provided to the case, that's the way to go.
  tvoc = analogRead(A1); // Please note that we are only reading raw data from this sensor, not ppm or ppb values. Just analog values from 0 to 1024. Higher values mean there is a presence of VOC
  digitalWrite(tvocPin, LOW);

  // Read LDR value
  ldrValue = analogRead(ldrPin);
  ldrPercent = map(ldrValue, 0, 1023, 0, 100); // Map LDR value to 0-100
  // Read UV sensor value
  uvValue = analogRead(uvPin);

  // Check for incoming data from the display - check whether we have clicked a
particular sensor for reading the last 24 hours
  checkForIncomingData();

  // Read MHZ19 - CO2 sensor for 3 seconds - if we don't use a blocking method with
the while loop we won't get values from the sensor.
  co2Serial.listen();
  dataTimer = millis();
  while (millis() - dataTimer <= 3000) {
    CO2 = myMHZ19.getCO2(); // Request CO2 (as ppm)
  }

  // we check for incoming data after each operation, because the operation is blocking
the program
  checkForIncomingData();

  // Read Particulate Matter sensor for 2 seconds
  pmsSerial.listen();
  dataTimer3 = millis();
  while (millis() - dataTimer3 <= 1000) {
    pms.readUntil(data);
    pm25 = data.PM_AE_UG_2_5;
  }
  checkForIncomingData();
  // Read MQ131 Ozone sensor
  MQ131.sample();
  o3 = MQ131.getO3(PPB);

  checkForIncomingData();

  // Get the time from the DS3231 Real Time Clock module - For setting the time use
the library example
  t = rtc.getTime();
  hours = t.hour;
  minutes = t.min;
  // Store current sensors data
  storeData();

  // Send the data to the Nextion display
  dataTimer4 = millis();
  while (millis() - dataTimer4 <= 200) {
    Serial.print("co2V.val=");
    Serial.print(CO2);
    // each command ends with these three unique write commands in order for the data
to be sent to the Nextion display
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);

    Serial.print("pm25V.val=");
    Serial.print(pm25);

    Serial.write(0xff);

    Serial.write(0xff);

    Serial.write(0xff);


    Serial.print("o3V.val=");

    Serial.print(o3);

    Serial.write(0xff);

    Serial.write(0xff);

    Serial.write(0xff);


    Serial.print("tempV.val=");

    Serial.print(temp);

    Serial.write(0xff);

    Serial.write(0xff);

    Serial.write(0xff);


    Serial.print("humV.val=");

    Serial.print(hum);

    Serial.write(0xff);

    Serial.write(0xff);

    Serial.write(0xff);


    Serial.print("tvocV.val=");

    Serial.print(tvoc);

    Serial.write(0xff);
```

```
    Serial.write(0xff);
    Serial.write(0xff);

    Serial.print("ldrV.val=");
    Serial.print(ldrPercent); // Send the mapped percentage value
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);

    Serial.print("uvV.val=");
    Serial.print(uvValue);
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);
  }
}

void checkForIncomingData() {
  // Check if data is coming from the Nextion
  if (Serial.available() > 0) {
    receivedData = Serial.readString();
    delay(30);
    if (receivedData == "0") {
      r = 0;
    }
    if (receivedData == "1") {
      r = 1;
    }
    if (receivedData == "2") {
      r = 2;
    }
    if (receivedData == "3") {
      r = 3;
    }
    if (receivedData == "4") {
      r = 4;
    }
  }
  // if we have received any data, send data to the Nextion display to change to page 1,
or the waveform
  if (r == 0 || r == 1 || r == 2 || r == 3 || r == 4) {
    delay(200);
    dataTimer3 = millis();
    while (millis() - dataTimer3 <= 200) {
        Serial.print("pageSwitch.val="); // Activate page 1, or the waveform on the
Nextion display
      Serial.print(1);
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
    delay(100);
    getLast24Hours(); // get the last 24 hours and print them as X-axis values on the
waveform
    getYAxisValues(); // get the Y-axis values according to the sensor, its range, and its
max value. Print the Y-axis values as well as scale the Y-axis of the waveform
accordingly
    sendDataToWaveform(); // send the stored data of the last 24 hours to the waveform
    r = 99; // reset the "r" to 99 (an arbitrary number, different from the ones we assign
when we receive data depending on which sensor we have pressed)
  }
}

void storeData() {
  // Storing current sensor values into arrays
  if ((minutes - previousMinutes) >= 15) {  // store the value each 15 minutes
     memmove(tempData, &tempData[1], sizeof(tempData)); // Slide data down one
position
    tempData[sizeof(tempData) - 1] = temp; // store newest value to the last position
    memmove(humData, &humData[1], sizeof(humData));
    humData[sizeof(humData) - 1] = hum;
    memmove(tvocData, &tvocData[1], sizeof(tvocData));
    // we use bytes for storing the data, as we said the Arduino Pro mini doesn't have
enough memory, so we must convert the values from 0 to 1000 to 0 to 255 which is
one byte
    tvocData[sizeof(tvocData) - 1] = map(tvoc, 0, 1000, 0, 255);
    memmove(co2Data, &co2Data[1], sizeof(co2Data));
    co2Data[sizeof(co2Data) - 1] = map(CO2, 0, 3000, 0, 255);
    memmove(pm25Data, &pm25Data[1], sizeof(pm25Data));
    pm25Data[sizeof(pm25Data) - 1] = map(pm25, 0, 1000, 0, 255);
    memmove(o3Data, &o3Data[1], sizeof(o3Data));
    o3Data[sizeof(o3Data) - 1] = map(o3, 0, 1000, 0, 255);
    memmove(ldrData, &ldrData[1], sizeof(ldrData));
    ldrData[sizeof(ldrData) - 1] = map(ldrValue, 0, 1024, 0, 255); // store LDR value
    memmove(uvData, &uvData[1], sizeof(uvData));
    uvData[sizeof(uvData) - 1] = map(uvValue, 0, 1024, 0, 255); // store UV value
    previousMinutes = minutes;
  }
  // So these if statements check whether 15 mins have passed since the last time we
stored a value - you can change this to any minutes you want, but you need to do that
on both if statements, for example "10" in the first if statement, and "-50" in the second
if statement
  else if ((minutes - previousMinutes) == -45) { // when minutes start from 0, next hour
     memmove(tempData, &tempData[1], sizeof(tempData)); // Slide data down one
position
    tempData[sizeof(tempData) - 1] = temp; // store newest value to the last position
    memmove(humData, &humData[1], sizeof(humData));
    humData[sizeof(humData) - 1] = hum;
    memmove(tvocData, &tvocData[1], sizeof(tvocData));
    tvocData[sizeof(tvocData) - 1] = map(tvoc, 0, 1000, 0, 255);
    memmove(co2Data, &co2Data[1], sizeof(co2Data));
    co2Data[sizeof(co2Data) - 1] = map(CO2, 0, 3000, 0, 255);
    memmove(pm25Data, &pm25Data[1], sizeof(pm25Data));
    pm25Data[sizeof(pm25Data) - 1] = map(pm25, 0, 1000, 0, 255);
    memmove(o3Data, &o3Data[1], sizeof(o3Data));
    o3Data[sizeof(o3Data) - 1] = map(o3, 0, 1000, 0, 255);
    memmove(ldrData, &ldrData[1], sizeof(ldrData));
    ldrData[sizeof(ldrData) - 1] = map(ldrValue, 0, 1024, 0, 255); // store LDR value
    memmove(uvData, &uvData[1], sizeof(uvData));
    uvData[sizeof(uvData) - 1] = map(uvValue, 0, 1024, 0, 255); // store UV value
    previousMinutes = minutes;
  }
}

void getLast24Hours() {
  for (int i = 11; i >= 0; i--) {
    last24Hours[11] = hours; // get the current hour - according to this hour get 12 more
hours, each 2 hours. For example, current hour = 13, so 11, 9, 7...
    last24Hours[i - 1] = last24Hours[i] - 2;
    if (last24Hours[i - 1] < 0) {
      for (int k = -0; k > -11; k--) {
        if (last24Hours[i - 1] == k) {
          last24Hours[i - 1] = 24 + k;
        }
      }
    }
  }
  // send the hours values to the Nextion display
  for (int i = 0; i < 12; i++) {
    String last24 = ("n") + String(i) + String(".val=") + String(last24Hours[i]); // e.g.
for i=0 > "n0.val="
    Serial.print(last24);
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);
    delay(20);
  }
  // Another write just to make sure it sends all data
  for (int i = 0; i < 12; i++) {
    String last24 = ("n") + String(i) + String(".val=") + String(last24Hours[i]); // e.g.
for i=0 > "n0.val="
    Serial.print(last24);
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);
    delay(20);
  }
}

// With the following custom function we set the Y axis value for each sensor
individually, as each sensor has different maximum value for the Y axis
void getYAxisValues() {
  maxV = 0;
  // PM2.5 Y-axis values
  if (r == 0) {
    // Get the max sensor value from the last 24 hours
    for (int i = 0; i < sizeof(pm25Data); i++) {
      if (maxV < map(pm25Data[i], 0, 255, 0, 1000)) {
        maxV = map(pm25Data[i], 0, 255, 0, 1000);
      }
    }
    // Setting the Y-axis values and scaling the waveform
    if (maxV <= 100) {
      yAxisValues[0] = 25;
      yAxisValues[1] = 50;
      yAxisValues[2] = 75;
      yAxisValues[3] = 100;
      Serial.print("s0.dis="); // this command ".dis" is used for scaling the Y-axis
       Serial.print(78 * 10);  // scale the waveform Y-axis - pm2.5 values are from 0 to 1000 which are represented from 0 to 78% in the Y axis - 78% because the waveform is 200px which is 78% of 255 which is the default 100% value of the waveform - 78*10 because we show values from 0 to 100, which are 10 times smaller
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
    // if the value is higher than 100, get its max value, and according to it scale the y-axis - For example, if the max value is 235, set the max value of the Y-axis to 300 - 235/100=2+1=3*100=300
    else if (maxV > 100) {
      int l = ((maxV / 100) + 1) * 100;  // get the hundreds value so we can properly scale the Y axis of the waveform
      yAxisValues[0] = l / 4;
      yAxisValues[1] = l / 2;
      yAxisValues[2] = l * 3 / 4;
      yAxisValues[3] = l;
      float ll = 78.0 / (l / 1000.0); // scale value for the Y-axis in % - We multiply by 78 instead of 100 because our waveform is 200px in height, which is 78% of 255 (255 is max value the waveform can accept, 1 byte)
      Serial.print("s0.dis=");
      Serial.print(round(ll));
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
  }
  // TVOC Y-axis values
  if (r == 2) {
    // Get the max sensor value from the last 24 hours
    for (int i = 0; i < sizeof(tvocData); i++) {
      if (maxV < map(tvocData[i], 0, 255, 0, 1000)) {
        maxV = map(tvocData[i], 0, 255, 0, 1000);
      }
    }
    // Setting the Y-axis values and scaling the waveform
    if (maxV <= 100) {
      yAxisValues[0] = 25;
      yAxisValues[1] = 50;
      yAxisValues[2] = 75;
      yAxisValues[3] = 100;
      Serial.print("s0.dis=");
      Serial.print(78 * 10);
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
    else if (maxV > 100) {
      int l = ((maxV / 100) + 1) * 100;  // get the hundreds value so we can properly scale
the Y axis of the waveform
      yAxisValues[0] = l / 4;
      yAxisValues[1] = l / 2;
      yAxisValues[2] = l * 3 / 4;
      yAxisValues[3] = l;
      float ll = 78.0 / (l / 1000.0); // scale value for the Y-axis in % - We multiply by 78
instead of 100 because our waveform is 200px in height, which is 78% of 255 (255 is
max value the waveform can accept, 1 byte)
      Serial.print("s0.dis=");
      Serial.print(round(ll));
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
  }
  // Ozone Y-axis values
  if (r == 3) {
    // Get the max sensor value from the last 24 hours
    for (int i = 0; i < sizeof(o3Data); i++) {
      if (maxV < map(o3Data[i], 0, 255, 0, 1000)) {
        maxV = map(o3Data[i], 0, 255, 0, 1000);
      }
    }
    // Setting the Y-axis values and scaling the waveform
    if (maxV <= 100) {
      yAxisValues[0] = 25;
      yAxisValues[1] = 50;
      yAxisValues[2] = 75;
      yAxisValues[3] = 100;
      Serial.print("s0.dis=");
      Serial.print(78 * 10);
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
    else if (maxV > 100) {
      int l = ((maxV / 100) + 1) * 100;  // get the hundreds value so we can properly scale
the Y axis of the waveform
      yAxisValues[0] = l / 4;
      yAxisValues[1] = l / 2;
      yAxisValues[2] = l * 3 / 4;
      yAxisValues[3] = l;
      float ll = 78.0 / (l / 1000.0); // scale value for the Y-axis in % - We multiply by 78
instead of 100 because our waveform is 200px in height, which is 78% of 255 (255 is
max value the waveform can accept, 1 byte)
      Serial.print("s0.dis=");
      Serial.print(round(ll));
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
  }
  // CO2 Y-axis values are fixed from 0 to 3000 so we don't need to look for the max
value in the array
  if (r == 1) {
    // Get the max sensor value from the last 24 hours
    for (int i = 0; i < sizeof(co2Data); i++) {
      if (maxV < map(co2Data[i], 0, 255, 0, 3000)) {
        maxV = map(co2Data[i], 0, 255, 0, 3000);
      }
    }
    if (maxV <= 2000) {
      // Setting the Y-axis values and scaling the waveform
      yAxisValues[0] = 500;
      yAxisValues[1] = 1000;
      yAxisValues[2] = 1500;
      yAxisValues[3] = 2000;
      Serial.print("s0.dis=");
      Serial.print(117);  // scale the waveform from 0 - 3000 to 0 - 2000 range
      Serial.write(0xff);
      Serial.write(0xff);
      Serial.write(0xff);
    }
    if (maxV > 2000) {
      // Setting the Y-axis values and scaling the waveform
      yAxisValues[0] = 750;
      yAxisValues[1] = 1500;
      yAxisValues[2] = 2250;

      yAxisValues[3] = 3000;

      Serial.print("s0.dis=");

      Serial.print(78);

      Serial.write(0xff);

      Serial.write(0xff);

      Serial.write(0xff);

    }

  }

  // Temperature and Humidity Y-axis values - fixed from 0 to 100

  if (r == 4) {

    // Setting the Y-axis values and scaling the waveform

    yAxisValues[0] = 25;

    yAxisValues[1] = 50;

    yAxisValues[2] = 75;

    yAxisValues[3] = 100;

    Serial.print("s0.dis=");

    Serial.print(200); // from 0 to 100 - 255/100 * 78 = ~200

    Serial.write(0xff);

    Serial.write(0xff);

    Serial.write(0xff);

  }

  delay(50);

  // Send the  Y-axis values to the Nextion display

  for (int i = 0; i < 4; i++) {
    String yValues = ("y") + String(i) + String(".val=") + String(yAxisValues[i]); // e.g.
for i=0 > "y0.val="
    Serial.print(yValues);
    Serial.write(0xff);
    Serial.write(0xff);
    Serial.write(0xff);
    delay(10);
  }
}

void sendDataToWaveform() {
  int k = 0;
  while (k != 2) {
    String str = String("addt 1,0,") + String(288); // with this command we tell the
Nextion display that we will send an array of data to the waveform
    Serial.print(str);
    delay(100);
    Serial.write(0xFF);
    Serial.write(0xFF);
    Serial.write(0xFF);
    delay(100);
    // Now depending on the selected sensor we want the values stored in the arrays
    // PM2.5
    if (r == 0) {
      for (int t = 0; t < sizeof(pm25Data); t++) {
        int z = 0;
        while (z != 3) {
```

```
      Serial.write(pm25Data[t]);
      z++;
    }
  }
}
// CO2
if (r == 1) {
  for (int t = 0; t < sizeof(co2Data); t++) {
    int z = 0;
    while (z != 3) {
      Serial.write(co2Data[t]);
      z++;
    }
  }
}
// TVOC
if (r == 2) {
  for (int t = 0; t < sizeof(tvocData); t++) {
    int z = 0;
    while (z != 3) {
      Serial.write(tvocData[t]);
      z++;
    }
  }
}
// Ozone
```

```
if (r == 3) {
  // Temperature values on channel 0
  for (int t = 0; t < sizeof(o3Data); t++) {
    int z = 0;
    while (z != 3) {
      Serial.write(o3Data[t]);
      z++;
    }
  }
}
// Temp and hum
if (r == 4) {
  for (int t = 0; t < sizeof(humData); t++) {
    int z = 0;
    while (z != 3) {
      Serial.write(humData[t]);
      z++;
    }
  }
  delay(100);
  Serial.write(0xFF);
  Serial.write(0xFF);
  Serial.write(0xFF);
  delay(100);
  // Humidity values on channel 1
  String str = String("addt 1,1,") + String(288);
```

```
      Serial.print(str);
      delay(100);
      Serial.write(0xFF);
      Serial.write(0xFF);
      Serial.write(0xFF);
      delay(100);
      for (int t = 0; t < sizeof(tempData); t++) {
        int z = 0;
        while (z != 3) {
          Serial.write(tempData[t]);
          z++;
        }
      }
      delay(100);
      Serial.write(0xFF);
      Serial.write(0xFF);
      Serial.write(0xFF);
    }
    k++;
  }
}
```

Nr.	Paper title	Authors	Year	Source	What is it about?	Solved Problems	Unsolved Problems	Future work
1.	[1]“Air Quality and Climate Control Arduino Monitoring System using Fuzzy Logic for Indoor Environments”	Anas Bushnag	2021	IEEE	The suggested approach makes use of fuzzy logic controllers and the Arduino platform to provide a completely automated system that detects and regulates temperature, humidity, and indoor air quality.	The approach suggested works excellently when it comes to regulating while maintaining updates on the indoor air quality within a room. The fuzzy logic controller, which modifies the ventilation system's speed and interval of execution, is responsible for this decrease.	This study does not present any unsolved problems.	The proposed framework can be enhanced in a variety of ways in the future. Raising the number of sensors can help enhance the reliability of environmental condition readings. Raising the number of member functions within fuzzy logic to get a higher accuracy ventilation speed could be an additional method of system improvement.

2.	[2]“Air Quality Sensing and Reporting System Using IoT”	Rohan Kumar Jha	2020	IEEE	This study presents an innovative real-time air quality monitoring system that combines Internet of Things (IoT) infrastructure for assistance.	The Arduino Uno receives analog signals from a variety of gas sensors, including the MQ135, dust detector, MQ7, and others, using its analog input ports. The Arduino Uno's ADC transforms these data into digital format. The obtained data is initially transformed into parts per million (ppm) of the gases, and the index of air quality is then computed utilizing this ppm of gases.	This study does not present any unsolved problems.	This system requires barely any energy for operation, and batteries can power it with ease. To improve the sensors' sensitivity and expand their detecting capabilities for broad use, more optimization work could be done.

3.	[3]“Smart Air Quality Monitoring System Using Arduino Mega”	M K Fadzly, Yiling M F Rosli T. Amarul	2020	IOP Science	The objective is to use an Arduino Mega to build and construct a system to track air quality that will improve reliability to a level that just slight discomfort in the eyes can cause an individual to lose focus and cause major incidents. Whenever a cloudy day is detected by the temperature and dust sensor, a buzzer will warn the user to limit outdoor activities.	To guarantee that the information was precise, the project underwent testing throughout the entire day. Furthermore, it could help in preventing errors in projects. The buzzer component will sound to notify the user once the degree of sensitivity is set to high.	This study does not present any unsolved problems.	No future work is proposed.

4.	[4]“Development of an IoT-based Real-Time Air Quality Monitoring Device”	Bikash Kumar Moharana, Prateek Anand, Sarvesh Kumar and Prakash Kodali	2020	IEEE	The NodeMCU ESP32, the MQ-135 gas sensor, and the DHT-11 humidity and temperature sensor component make up the recommended air quality surveillance equipment. Their suggested system has an advantage over its competitors concerning price, compact size, and efficient power utilization. The NodeMCU, which serves as the setup's base station, receives the data that the sensors capture.	The NodeMCU system ESP32, which serves as the setup's base station, is utilized by the suggested system. It functions being a microcontroller chip as well as a combination of WI-FI and Bluetooth chips that can replace the GSM module and the microcontroller with identical hardware. It also boasts a sturdy build and extremely low power consumption.	The experiment must be carried out in several settings, including outside where calibration can be done.	No future work is proposed.

5.	[5]“Smart Indoor Environment Monitoring System”	Luay Fraiwan Abdulrahman Mahmoud Rajab	2020	IEEE	The present study offers an alternative to the standard methods of collecting air samples and testing in laboratories to maintain indoor air quality.	A central microprocessor in the suggested system gathers information from multiple sensors, including humidity and temperature, CO2, CO, VCO, and LPG. The microcontroller stores the information it has collected in an internet database. With a dedicated application designed for this purpose, every encoded information can be viewed and examined. The application continuously monitors the data and notifies the user if any measured levels surpass the user-established thresholds.	The current remedies to the risks associated with indoor environmen ts are fragmented and insufficient. Moreover, they don't offer a way to continuousl y monitor.	This study provides an intelligent indoor environmental surveillance system. The data from the recording can be utilized to create a mathematical framework that forecasts potential mold growth, allowing for the implementation of preventative actions before mold starts to form.

6.	[6]“IoT Based Air and Noise Pollution Monitoring System”	Pooja Shraddha Priyanka A. D. Sonawane	2021	IEEE	This study suggests an Internet of Things (IoT)-based air quality and sound pollution detection system that enables real-time monitoring and verification of air quality and sound pollution in specific areas.	The device continuously sends information to the microcontroller while using an air sensor to sense the existence of dangerous gases and other compounds in the atmosphere.	This study does not present any unsolved problems.	They plan to update the system in the future to send an SMS or app alert to the user when the condition of the air and noise level exceeds what is considered acceptable. Anywhere in the world, we have access to monitoring devices for sound and air pollution.

7.	[7]“Arduino Based Ambient Air Pollution Sensing System”	Mykhailo Lobur Marek Iwaniec Uliana Marikutsa	2020	IEEE	The suggested ambient air quality sensing system transports data to higher-level programs for evaluation and forecasting and offers real-time, flexible, cost-effective monitoring of the five air parameters such as CO, PM 2.5, carbon dioxide (CO2) temperature, and humidity that are most critical to human health in metropolitan areas.	The system employs the platform known as Arduino as the controlling unit and combines temperature, humidity, carbon monoxide, PM 1.0, PM 2.5, PM 10, and carbon dioxide (CO2) sensors into a single small unit. GPS positions and timestamps are linked together with the data collected.	The design would make it easy to monitor further critical air pollution characteristics by simply integrating more sensors that provide the appropriate interface.	No future work is proposed.

8.	[8]“Design of Micro-Climate Data Monitoring System for Tropical Greenhouse based on Arduino UNO and Raspberry Pi”	Ardiansah , N Bafdal , E Suryadi , A Bono	2020	IEEE	The purpose of this research initiative is to better understand how to use a wireless system of monitoring that can capture microclimate data and upload it into the cloud for end-users through a web application.	The suggested approach needs to reduce human interference, be economical, and be dependable. The Raspberry Pi serves as a processor for information and the Arduino UNO serves as a monitoring unit. Microclimate analysis data on the Raspberry Pi is wirelessly transmitted to a database in the cloud within a specific time frame, where it gets processed by a web application and displayed when a web browser request it.	With a single sensor, the created gadget can be used in small to medium-sized greenhouse s. The price of purchasing equipment increases for applications on a large scale, while the cost for surveillance stays the same.	By simply modifying specific code lines in UNO, RPi, and CA, it can be achieved to add additional types of sensors required for the growth of plants, such as carbon dioxide, wind, sun index, and soil moisture sensors.

9.	[9]“IoT Based Wireless Sensor Network for Air Pollution Monitoring”	Ajay Chaturvedi Laxmi Shrivastava	2020	IEEE	Among the primary fields where plenty of effort is being made is the use of wireless sensor networks. The work encompasses a wide range of tasks, including air pollution, gas and water monitoring, fire detection, and spread.	The detectors are placed to track the surroundings after the entire targeted region has been split up into smaller local zones. Zigbee can be used to send data that has been gathered in the nearby vicinity. Every node that uses several nodes can send information to a single computer. Through the Internet, information from computers can be sent to a shared control center.	This study does not present any unsolved problems.	No future work is proposed.

10.	[10]“Microcontroller-Based Environmental Pollution Monitoring System though IoT Implementation”	Nadia Nowshin Md. Shajedul Hasan	2021	IEEE	Attempting to mitigate the adverse effects of the aforementioned pollution and to facilitate the implementation of anti-pollution measures, this study suggested and carried out a fully automated microcontroller-based air along with noise pollution monitoring.	The built-in sensors combined with the microcontroller in the proposed system enable it to detect greater decibel noise levels and hazardous gas components. Through the data it collects, the sensors communicate with the microcontroller. The microcontroller then uses an IoT (Internet of Things) analytical application framework to analyze and transmit the data via an Internet server.	This study does not present any unsolved problems.	No future work is proposed.

11.	[11]“Wearable Smart Prototype for Personal Air Quality Monitoring”	Attila Géczy, Lajos Kuglics, László Jakab, Gábor Harsányi	2020	IEEE	With its compact size, ease of use, and smartphone integration, this work intends to demonstrate a sophisticated concept of affordable wearing air quality monitoring technology appropriate for various use scenarios.	It makes applications for Arduino programming and Mobile software design. The suggested setup consists of an Arduino board, a carbon capture and storage (CCS811) sensor for measuring volatile organic compounds (VOCs), a ZPH01 particle matter (PM) detector, as well an HC-05 Bluetooth device.	It can be challenging to organize or evaluate outcomes based on quantitative data for a non-professional user.	No future work is proposed.

12.	[12]“Design and Implementation of a Feasible Model for the IoT Based Ubiquitous Healthcare Monitoring System for Rural and Urban Areas”	Mohammad Nuruzzaman Bhuiyan; Md Masum Billah; Farzana Bhuiyan	2022	IEEE	The article describes an IoT-based health-tracking system that can assess, track, and report on individual medical conditions online as well as offline from anywhere.	The suggested IoT-based approach can convey critical health data to healthcare facilities and caretakers in real-time. The suggested system uses Arduino UNO, Nodemcu, and Global System for Mobile Communication (GSM) modules to monitor human body temperature, heart rate, saturation of oxygen, ambient temperature, and pollution levels in a smart home environment.	Several constraints and pertinent circumstances hamper continual development; yet, such investigations provide significant possibilities to address the highlighted difficulties.	The authentication process of networks is an important part of assuring the foundation of remote healthcare surveillance systems. After proper manufacture, the system has great potential for urban as well as rural areas, especially in nations that are emerging.

13.	[13]“Arduino-Based Weather Monitoring System”	Kazi Sultanabanu, Sayyad Liyakat, Kutubuddin Kazi	2023	Researc h Gate	In the following article, they develop an Arduino-based environmental monitor platform that can offer us accurate current time weather information for our location, such as the condition of the air, humidity, temperature, pressure in the atmosphere, and the intensity of light.	They use the DHT11 sensor, which can quickly determine temperature and humidity. Furthermore, they utilize the Arduino Uno to process and present the data on the screen.	This study does not present any unsolved problems.	As technology advances, they can expect innovations in this field, which will improve the efficiency and reliability of weather monitoring.

14.	[14]“Arduino Based Smart Home Warning System”	Qusay Idrees Sarhan	2020	IEEE	The article describes the conceptualization and execution of an Arduino-based automated home alert system.	In this framework, the microcontroller known as Arduino Uno is being utilized along with multiple appropriate detectors (the DHT22, the MQ2, as well as the camera), mechanisms (a buzzer and relays that come with connected water valve, air ventilator, as well as a spotlight lamp), and GSM as a mobile communications medium to allow users to communicate with the system that has been suggested. .	This study does not present any unsolved problems.	For future work, supplying coordinates from GPS will be extremely beneficial when sending SMSs and emails to a police or fire station. This technology will allow law enforcement or firefighters to simply locate the house that is having trouble.

15.	[15]“Low-cost Arduino-based temperature, relative humidity and CO2 sensors - An assessment of their suitability for indoor built environments”	Pedro F. Pereira Nuno M.M. Ramos	2022	ResearchGate	This paper describes cost-effective Arduino-compatible detectors that have been validated in the indoor environments of Southern European countries.	Five various sensors for the following characteristics were examined: temperature, moisture content (RH), and carbon dioxide. The T and RH experiments combine two types of exposure over 24 months, with temperatures ranging from 10 °C to 35 °C and humidity levels ranging from 50% to 95%.	This study does not present any unsolved problems.	No future work is proposed.

16.	[16]“Design of an Arduino-Powered Sleep Monitoring System Based on Electrooculography (EOG) with Temperature Control Applications”	Genesis M. Cruz Kyle Reifel M. Del Villar Charmaine C. Paglinawan	2022	IEEE	This study proposes an electrooculogram-based system for monitoring sleep. When contrasted with PSG or EEG recording sessions, EOG is easy to install and can be employed independently.	An EOG sleep tracking device was created utilizing the Arduino Uno Module, and it obtained a total precision of 82.31% in recognizing different sleep stages. Along with sleep evaluation, adaptive temperature management according to the suggested system's tracking of users' phases of sleep was used to control the active sleep environment.	This study does not present any unsolved problems.	It recommended raising both the total amount of those involved and the number of hours of sleeping tracked per user.
17.	[17]“Implementation of Arduino Pro Mini and ESP32 cam for temperature monitoring on automatic thermogenic IoT-based "	Puput Wanarti Rusimamto, Endryansyah, Lilik Anifah	2020	Resarch Gate	The goal of this study is to track temperature using Arduino Pro Mini and ESP32 with IoT technology linked to a web interface.	The procedure of photographing and monitoring body temperature is automated. The offered web interface allows users to view sensor data as well as photo information provided by Arduino and ESP32.	Limitations with the proximity detector, which is oversensitive, make it simple to detect nearby items.	Problems can be avoided by constructing the mechanical construction so that it limits the sensitivity of the proximity sensor.