**Abstract**

This thesis presents the development and implementation of an IoT-based environmental monitoring system using Arduino Nano 33 IoT for measuring atmospheric data, including temperature, humidity, and pressure. The collected sensor data is securely transmitted via MQTT protocol over a WiFi network to a Node-RED dashboard hosted on a Raspberry Pi. Real-time visualization and local storage of atmospheric data in .csv format on an SD card are achieved, allowing remote access to the data through an IP address without the need for an external monitor.Security and reliability are fundamental aspects of this system. The MQTT protocol is utilized with encryption mechanisms to ensure the secure transmission of data, maintaining its confidentiality and integrity during transfer. Additionally, system reliability is emphasized to ensure consistent and accurate data collection and transmission. In addressing energy considerations, the thesis includes an analysis of power consumption patterns associated with the IoT devices used in the monitoring system. This analysis aims to optimize energy efficiency and sustainability in IoT deployments. The study showcases the practical implementation of IoT technologies for environmental monitoring, demonstrating a comprehensive approach to data collection, transmission, visualization, and storage. By integrating security, reliability, and energy efficiency considerations, this research contributes to advancing IoT applications in environmental science and underscores the potential for scalable and sustainable IoT solutions in various monitoring contexts. This thesis contributes valuable insights into the design and deployment of IoT-based environmental monitoring systems, highlighting the importance of secure and reliable data transmission, user-friendly data visualization, and energy-conscious IoT device management. The findings presented offer practical implications for researchers and practitioners interested in leveraging IoT technologies for environmental data collection and analysis.

List of Abbreviations :

**Acknowledgements**

I extend my deepest gratitude to my esteemed professors for their unwavering support, guidance, and encouragement during my master's program. Their expertise and commitment have played an instrumental role in shaping both my research endeavors and academic development. Special appreciation is due to Professor Dr.-Ing. Werner Bogner and my supervisor, Michael Benisch, for their invaluable feedback and constructive criticism, which have significantly enhanced the quality of my work. Their mentorship and motivation have been pivotal to my achievements.I am profoundly thankful to my mother for her boundless love, unwavering support, and for instilling in me the values of perseverance and diligence. Without her sacrifices and encouragement, reaching this milestone would not have been possible.Lastly, heartfelt thanks go to my siblings for their constant presence, support, and belief in me throughout this journey. Their unwavering confidence has served as a constant source of strength and motivation. I am deeply appreciative of the contributions each of you has made to my personal and academic growth.

**Table of Contents**

**List of Figures and Tables**

**Introduction**

The Internet of Things (IoT) refers to a network of interconnected devices, machines, objects, and even living beings that possess unique identifiers (UIDs). All have the capability to exchange data over a network without the need for direct human or computer interaction. These devices can range from computing devices to mechanical systems, and enable seamless communication and data transfer, leading to increased automation and efficiency in various domains. By leveraging IoT technology, the world becomes more interconnected, allowing for enhanced monitoring, control, and communication between diverse entities, ultimately shaping a more interconnected and intelligent ecosystem. In the Internet of Things (IoT), a “thing” can refer to various entities, such as a person wearing a smartwatch, livestock fitted with RFID transponders, or vehicles such as cars, trucks or motorbikes with tracking devices. Essentially, any object, whether natural or man-made, can be assigned an Internet Protocol (IP) address and possess the capability to transmit data over a network. This interconnectedness enables seamless communication and data exchange, enhancing monitoring, control, and functionality across a wide range of applications. The IoT empowers objects and beings with the ability to contribute to and benefit from the vast network of connected devices, creating a more intelligent and interconnected world.(<https://internationalsecurityjournal.com/why-iot-is-important/>)

**History of IoT:**

In 1999, the co-founder of the Auto-ID Center at MIT, Kevin Ashton introduced the concept of the Internet of Things (IoT) during a presentation to Procter & Gamble (P&G).His intention was to highlight the potential of radio frequency identification (RFID) technology to P&G’s senior management. Ashton cleverly named his presentation “Internet of Things” to align with the trending concept of the internet during that time. Additionally, MIT professor Neil Gerstenfeld’s book, titled “When Things Start to Think,” published in the same year, provided a clear vision of the direction in which the IoT was heading, even though it didn’t explicitly use the term.These early contributions laid the foundation for the development and recognition of the IoT as a transformative concept.While Kevin Ashton is credited with popularising the term “Internet of Things” in his presentation in 1999, the concept of connected devices has been present since the 1970s. Back then, it was referred to as the “embedded internet” or “pervasive computing.” These early ideas laid the foundation for the interconnected world we see today, where devices are seamlessly integrated into our daily lives, sharing data and enabling new possibilities. Ashton’s contribution brought attention to this evolving concept and helped shape the narrative around the Internet of Things. Today, the Internet of Things (IoT) has emerged through the convergence of wireless technologies, microelectromechanical systems (MEMS), microservices, and the internet. This convergence has played a crucial role in breaking down the barriers between operational technology (OT) and information technology (IT). As result, unstructured data generated by machines can now be analyzed to extract valuable insights and drive continuous improvements.The integration of these technologies has paved the way for a more interconnected and data-driven ecosystem. In the early 1980s, a notable example of an internet appliance was a Coke machine located at Carnegie Mellon University.This machine was connected to the web, allowing programmers to remotely check its status.

They could easily determine if the machine was stocked with cold drinks, saving them a trip if they were not available. This early instance showcased the potential of connecting everyday objects to the internet and accessing information remotely, foreshadowing the broader concept of the Internet of Things. IoT emerged from the concept of machine-to-machine (M2M) communication, which involves the interconnection of devices through a network without human intervention. M2M technology enables devices to connect to the cloud, enabling remote management and data collection. It forms the foundation of IoT by facilitating the seamless exchange of information and enabling devices to operate intelligently and autonomously represents the advancement of M2M technology, forming a vast network of intelligent devices that connect individuals, systems, and various applications to gather and exchange data.M2M serves as the underlying connectivity framework that enables the seamless integration and communication between these devices, forming the foundation of IoT.(<https://internationalsecurityjournal.com/why-iot-is-important/>)

**Background**

Information on IoT and its applications in environmental monitoring: The integration of the Internet of Things (IoT) has revolutionized the way we interact with and monitor our environment. IoT facilitates the monitoring of crucial environmental phenomena through the deployment of sensor-equipped devices. These devices collect data, which is then wirelessly transmitted to storage servers, typically cloud-based, where it is analyzed and presented in a meaningful format. This data can be accessed via various user interfaces, such as mobile and web applications, enabling informed decision-making based on real-time environmental insights.Traditional notions of end users and servers in the Internet landscape are evolving with the emergence of IoT. In this new paradigm, devices and objects themselves become hosts, giving rise to the term "Internet of Things." These devices, equipped with sensors, capture and transmit data on various environmental parameters such as temperature, pressure, humidity, sound, emissions, and even vital signs. Environmental monitoring systems built on IoT principles encompass data collection through sensor networks and subsequent analysis for both short-term interventions, like remote heating or cooling control, and long-term data interpretation and trend analysis.The proliferation of IoT has led to numerous applications across different sectors, including smart power stations, smart cities, wearable devices, and home automation. These applications heavily rely on the collection of sensor data, its processing by microcontrollers, and its transmission via wired or wireless means to storage platforms like the cloud. Recent research efforts have focused on developing cost-effective IoT solutions using platforms such as Raspberry Pi and Arduino. These endeavors have resulted in practical implementations ranging from home automation to environmental monitoring systems.One significant aspect of IoT-enabled environmental monitoring is the utilization of wireless sensor networks. However, challenges such as energy consumption, network congestion, data loss, and reliability issues need to be addressed. Different wireless technologies like Bluetooth, Zigbee, Wi-Fi, GSM, MQTT and LoRa offer varying trade-offs in terms of energy efficiency, data transfer rates, and range, catering to diverse application requirements.In agriculture, IoT-based solutions have shown promise in water conservation and crop management. Soil moisture sensors, combined with camera and sensor networks, enable farmers to monitor and control irrigation systems remotely, optimizing water usage and improving crop yield. The integration of IoT technologies in agriculture holds potential for addressing challenges related to water scarcity and sustainable farming practices.( <file:///C:/Users/Sabir%20Shah/Downloads/IJAREEIEPAPERPUBLISHED.pdf>)

Cunin, McGrath, and MacNamee showcase a comprehensive approach to environmental monitoring using IoT technology, emphasizing the importance of sensor modules, wireless communication, data management, and user interface design. By deploying sensor modules across the University of Limerick campus, the team demonstrates the feasibility of real-time monitoring of various environmental parameters such as pollution, humidity, temperature, and luminosity. This initiative not only serves the immediate goal of environmental surveillance but also serves as a valuable educational platform for students to delve into IoT application development, requiring interdisciplinary skills in software development, data analysis, and hardware systems. The authors' emphasis on energy efficiency, durability, and scalability underscores the project's potential for broader applications beyond the university campus, laying the groundwork for future environmental monitoring initiatives on a larger scale(file:///C:/Users/Sabir%20Shah/Downloads/ICSensT.2018.8603658.pdf)

Problem statement: In the realm of atmospheric data collection and analysis, there persists a demand for a comprehensive system that seamlessly integrates measurement, wireless transmission, visualization, and storage while ensuring reliability, security, and energy efficiency. Although IoT technologies offer promising solutions, existing implementations often face challenges in terms of integration, coding complexity, visualization accessibility, and real-time data storage.Traditional approaches to IoT system development typically involve extensive programming, potentially leading to complexity and errors. However, leveraging Node-RED—an intuitive visual programming tool—provides a solution to this challenge. With its graphical interface and a wide array of pre-built functions, Node-RED simplifies the development process, allowing users to create intricate IoT workflows through drag-and-drop operations. This eliminates the need for manual coding, significantly reducing development time and potential errors while enhancing system flexibility and scalability. Moreover, Node-RED offers a diverse set of functions tailored for IoT applications, including data parsing, protocol conversion, and integration with various services and platforms. This versatility enables developers to tailor their solutions to specific requirements without extensive customization or external libraries, further streamlining the development process and enhancing system robustness.Additionally, Node-RED facilitates effortless data visualization through customization dashboards, which can be accessed via a web browser using the system's IP address. This accessibility ensures that stakeholders can easily monitor and interpret atmospheric data in real-time, fostering informed decision-making and enabling timely responses to environmental changes.The ability to store data in real-time enhances the system's utility and analytical capabilities. By continuously capturing and archiving atmospheric data to a secure location, stakeholders can conduct comprehensive analyses, identify trends, and derive actionable insights to support various applications, including environmental monitoring, research, and decision support.However, despite the advancements facilitated by Node-RED and IoT technologies, challenges persist in ensuring the secure and reliable transmission of data, optimizing energy consumption to prolong system lifespan, and addressing potential scalability issues to accommodate future expansion and integration with emerging technologies.Therefore, this thesis aims to address these challenges by designing and implementing an integrated IoT system for atmospheric data collection, transmission, visualization, and storage. Through the utilization of Arduino Nano 33 IoT and Raspberry Pi, coupled with the power of Node-RED, the research endeavors to develop a robust, efficient, and user-friendly solution that meets the diverse needs of stakeholders while contributing to the advancement of IoT-based environmental monitoring and analysis.

**Objectives of your project**

The overarching goal of this project is to design, develop, and implement an integrated IoT system for atmospheric data collection, transmission, visualization, and storage. Leveraging Arduino Nano 33 IoT, Raspberry Pi, Node-RED, and MQTT communication, the project aims to address key challenges in existing solutions while enhancing reliability, security, and energy efficiency. The specific objectives include:

**System Integration:**

* Integrate Arduino Nano 33 IoT and Raspberry Pi into a cohesive system for seamless data flow.
* Establish MQTT communication protocol between devices to enable reliable and secure data transmission.

**Node-RED Implementation:**

* Utilize Node-RED as the primary development platform for creating IoT workflows.
* Explore the vast library of pre-built functions in Node-RED to streamline development and minimize coding requirements.

**Ease of Programming:**

* Demonstrate the ease of programming with Node-RED through graphical interface and drag-and-drop functionality.
* Eliminate the need for extensive manual coding, reducing development time and potential errors.

**Data Visualization:**

* Develop customizable dashboards using Node-RED for intuitive visualization of atmospheric data (temperature, pressure, and humidity).
* Enable stakeholders to access real-time data visualization via a web browser using the system's IP address.

**Real-Time Data Storage:**

* Implement mechanisms for real-time data storage, ensuring continuous capture and archiving of temperature, pressure, and humidity data.
* Store data locally in a .csv file on the Raspberry Pi to facilitate easy access and analysis.

**Operational Autonomy:**

* Design the system to operate without the need for an external monitor or display, ensuring autonomous functionality.
* Enable remote access to system functionality and data visualization through the web-based interface.

**Energy Considerations:**

* Investigate energy-efficient strategies to optimize power consumption and prolong system lifespan.
* Implement measures to minimize energy usage during data collection, transmission, and storage processes.

**Security and Reliability:**

* Ensure secure and reliable transmission of data over wireless networks, utilizing encryption protocols and authentication mechanisms.
* Implement robust error handling and data integrity checks to mitigate potential risks and ensure data reliability.

**Scalability and Future Integration:**

* Design the system with scalability in mind to accommodate future expansion and integration with emerging technologies.
* Explore possibilities for integrating additional sensors or functionalities to enhance the system's capabilities over time.

By achieving these objectives, the project aims to deliver a robust, efficient, and user-friendly IoT solution for atmospheric data monitoring and analysis, specifically focusing on temperature, pressure, and humidity measurements. This contribution will advance environmental monitoring, research, and decision support efforts.

Overview of the methodology and technologies used (Arduino Nano 33 IoT, MQTT, Node-RED, Raspberry Pi)

**Literature Review (10-12 pages)**

**Review of relevant literature on:**

**IoT applications in environmental monitoring**

**Use of Arduino in data acquisition**

**MQTT protocol for IoT communication**

**Data visualization techniques in Node-RED**

**Security considerations in IoT**

**Reliability of IoT systems**

**Power consumption and energy considerations in IoT devices**

**Methodology**

**hardware and Software**

Microcontroller( Arduino Nano 33 IoT): In order to connect hardware to the PC, it requires both a programming language and some software, so the Arduino nano comes into play first

The Arduino Nano 33 IoT is Arduino's smallest board designed specifically for Internet of Things (IoT) applications. Powered by the Arm® Cortex®-M0 32-bit SAMD21 processor, it boasts a range of advanced features including the u-blox NINA-W102 Wi-Fi module and the ECC608A crypto-chip for enhanced security (https://docs.arduino.cc/hardware/nano-33-iot/#features).

Here are a few of the Arduino Nano’s salient characteristics:

**Wi-Fi Connectivity:** Equipped with the u-blox NINA-W102 Wi-Fi module, the Nano 33 IoT provides seamless connectivity to Wi-Fi networks, enabling IoT projects to connect to the internet and interact with online services. This feature facilitates remote monitoring, control, and data exchange, enhancing the versatility of IoT applications (Arduino, n.d.).

**Arduino Cloud Compatibility:** The Nano 33 IoT is fully compatible with the Arduino Cloud platform, allowing users to build IoT projects effortlessly in just minutes. With Arduino Cloud, developers can remotely manage and monitor their devices, access data in real-time, and create custom IoT applications with ease (Arduino, n.d.).

**Bluetooth® Connectivity:** Additionally, the Nano 33 IoT is Bluetooth® enabled, providing support for Bluetooth® Low Energy (BLE) applications. This feature enables users to control peripheral devices via Bluetooth® and implement a wide range of IoT applications that leverage BLE technology (Arduino, n.d.).

**Integrated IMU:** The Nano 33 IoT comes equipped with an integrated Inertial Measurement Unit (IMU) featuring the LSM6DS3 accelerometer and gyroscope. This IMU enables developers to create motion tracking devices and incorporate motion sensing capabilities into their projects, opening up opportunities for innovative applications in areas such as wearable technology and motion-controlled devices (Arduino, n.d.).

**Programming:** The Arduino Nano 33 IoT can be programmed using the Arduino IDE, a free, open-source program available for Windows, Mac, and Linux. This familiar development environment simplifies the process of writing, uploading, and debugging code for IoT projects (Arduino, n.d.).

**Size:** Similar to the Arduino Nano, the Nano 33 IoT boasts a compact form factor, measuring just 0.7 by 1.7 inches. This small size makes it easy to use on a breadboard or in tight spaces, facilitating rapid prototyping and experimentation

With its shared features and additional capabilities, the Arduino Nano 33 IoT builds upon the legacy of the Arduino Nano, offering enhanced connectivity and functionality for modern IoT projects.

A computer chip with many different colors

Description automatically generated with medium confidence

Figure 2.6: A brief description for the the Arduino pins

Using Arduino Nano 33 IOT instead of Arduino/Arduino UNO because of the following Reason:

1: One of the most significant advantages of the Nano 33 IoT is its built-in Wi-Fi and Bluetooth connectivity. Unlike the Arduino Uno, which requires additional shields or modules for wireless communication, the Nano 33 IoT simplifies IoT development by providing native support for Wi-Fi and Bluetooth connectivity

2: The Nano 33 IoT features a smaller form factor compared to the Arduino Uno, making it more suitable for projects with space constraints or those requiring a compact design. This smaller size allows for easier integration into portable devices, wearables, and other space-limited application

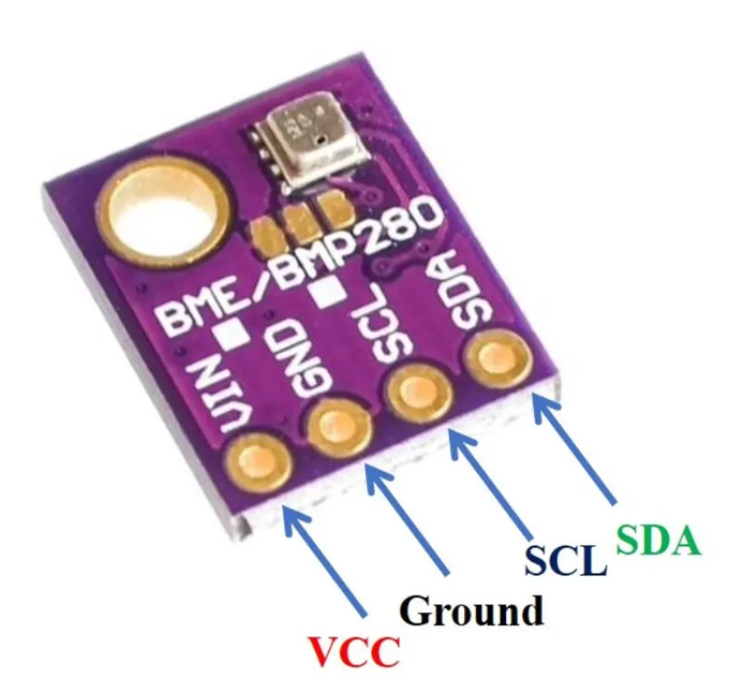
3: The Nano 33 IoT is equipped with a more powerful microcontroller compared to the Arduino Uno. Its ARM Cortex-M0 32-bit SAMD21 processor offers increased processing power and efficiency, allowing for faster execution of tasks and more complex computations. This makes the Nano 33 IoT better suited for applications requiring higher computational performance

4: Nano 33 IoT has two additional analog inputs compared to the Uno, providing slightly more flexibility for sensor interfacing.

However, There are no soldered pins on the Nano, so it must have its pins soldered in order to be used on a breadboard. When soldering, extreme caution must be taken to ensure that the circuit is not shorted. If there are two pins at the same point, then it will be short.

**Sensor(BME 280)**

It has now received the sensor component that it will use. Utilizing the most basic temperature,Humidity and pressure sensor BME280, test the circuit first.



**Features and Specifications of BME280**

**Features**

Compatible with 3.3V/5V microcontrollers

**Environmental monitoring:** temperature, humidity, and barometer

Gravity I2C interface and reserve XH2.54 SPI interface

Small size, convenient to install

**Specifications**

**Supply voltage:** 3.3 V

**Size:** 15 x 10 mm

**Weight:** 1 g

**Diameter of mounting holes:** 3 mm

**Operation range (full accuracy):**Pressure: 300…1100 hPa

**Temperature:** -40…85°C

**Interface:** I2C and SPI

Average current consumption (1Hz data refresh rate):1.8 μA @ 1 Hz (H, T)

2.8 μA @ 1 Hz (P, T)

3.6 μA @ 1 Hz (H, P, T)

Average current consumption in sleep mode: 0.1 μA

Humidity sensor: Response time (τ63%): 1 s

Accuracy tolerance: ± 3 % relative humidity

Hysteresis: ≤ 2 % relative humidity

Pressure sensor: RMS Noise: 0.2 Pa (equiv. to 1.7 cm)

Sensitivity Error: ± 0.25 % (Equiv. to 1 m at 400 m height change)

Temperature coefficient offset: ±1.5 Pa/K (Equiv. to ±12.6 cm at 1°C temperature change)

The BME280 sensor is a versatile environmental sensor capable of measuring temperature, humidity, and barometric pressure. Here are some of its key features:

Temperature Measurement:

The BME280 sensor accurately measures temperature with high precision. It provides temperature data in degrees Celsius (°C) with a resolution of up to 0.01°C.

Humidity Sensing:

In addition to temperature, the BME280 sensor also measures relative humidity. It provides humidity data as a percentage (%), allowing for precise monitoring of ambient humidity levels.

Barometric Pressure Sensing:

The BME280 sensor is equipped to measure barometric pressure, enabling users to monitor changes in atmospheric pressure over time. Barometric pressure data is typically provided in hectopascals (hPa) or millibars (mbar).

High Accuracy:

With its advanced sensing elements and calibration algorithms, the BME280 sensor offers high accuracy in temperature, humidity, and pressure measurements. This ensures reliable data for various applications, including weather monitoring and indoor climate control.

Compact Size:

Despite its advanced capabilities, the BME280 sensor maintains a compact form factor, making it suitable for integration into small electronic devices and IoT projects. Its small size allows for easy placement and installation in a variety of environments.

Low Power Consumption:

The BME280 sensor is designed with energy efficiency in mind, consuming minimal power during operation. This makes it well-suited for battery-powered applications where power consumption is a critical consideration.

I2C and SPI Interfaces:

The BME280 sensor supports both I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) communication protocols, providing flexibility in interfacing with microcontrollers and other devices. This allows for seamless integration into a wide range of projects and platforms.

Wide Operating Range:

The BME280 sensor operates over a wide temperature and humidity range, making it suitable for both indoor and outdoor applications. It can withstand environmental conditions ranging from -40°C to 85°C and 0% to 100% relative humidity.

Overall, the BME280 sensor offers a comprehensive solution for environmental monitoring, combining temperature, humidity, and barometric pressure sensing capabilities in a compact and energy-efficient package. Its high accuracy, low power consumption, and versatile interface options make it an ideal choice for a variety of IoT, weather monitoring, and indoor climate control applications.

Measuring Data/MQTT protocol implementation: To facilitate the measurement of atmospheric data,Some Libraries are installed as described below:

Wire.h: This library enables communication over the I2C bus, facilitating communication with devices such as sensors or displays.

Adafruit\_Sensor.h: Adafruit\_Sensor.h provides an abstract base class for sensor drivers, ensuring consistent sensor data handling across different sensor types.

Adafruit\_BME280.h: Adafruit\_BME280.h is a library specifically designed for the BME280 sensor, offering functions to easily interface with and read data from the sensor.

WiFiNINA.h: WiFiNINA.h is a library for the Arduino WiFi module, allowing the Arduino to connect to Wi-Fi networks and communicate over the internet.

PubSubClient.h: PubSubClient.h is a library for MQTT (Message Queuing Telemetry Transport) communication, enabling the Arduino to publish and subscribe to messages on MQTT topics.

ArduinoJson.h: ArduinoJson.h is a library for parsing and generating JSON (JavaScript Object Notation) data, commonly used for exchanging data between devices and services in IoT applications.

After writing the complete code (refer to the Appendix), the Arduino is connected to the PC via a USB communication/power cable. Upon uploading and burning the code, the corresponding atmospheric data is displayed in the serial monitor.

Subsequently, the Arduino will establish a connection to the local network via Wi-Fi. It will then transmit this data through the MQTT protocol. A specific topic named "sensor data" has been created, and all three values (temperature, humidity, and pressure) will be published to this topic on the server.

This process ensures that the collected atmospheric data is efficiently transmitted and made available for further analysis or utilization.

Configuration of Node-RED on Raspberry Pi for data reception and visualization: To enable data reception via the MQTT protocol, a Raspberry Pi was utilized. Prior to receiving data and making it accessible, the Raspberry Pi needed to be set up. This involved the installation of necessary components. To begin, the Raspberry Pi required the following essentials:

A power supply

Boot media, typically a microSD card with adequate storage and speed

The Raspberry Pi can be configured either as an interactive computer with a desktop or as a headless computer accessible solely over the network. For a headless setup, additional peripherals such as a display, keyboard, and mouse are not required. Instead, preconfiguring a hostname, user account, network connection, and SSH during the operating system installation suffices. However, for direct usage of the Raspberry Pi, the following accessories are essential:

A display

A cable to connect the Raspberry Pi to the display

A keyboard

A mouse

A power supply

Regarding the power supply, different Raspberry Pi models have varying power requirements. For instance, the Raspberry Pi 4 Model B necessitates a 5V/3A power supply, while older models such as the Raspberry Pi 3 require a 5V/2.5A supply. It's crucial to use the appropriate power supply to ensure proper functioning of the Raspberry Pi.

Boot media is essential for Raspberry Pi models, as they lack onboard storage. Typically, microSD cards are used for this purpose. The recommended SD card capacity varies depending on the intended usage, with at least 32GB recommended for Raspberry Pi OS installations. Additionally, the use of a wired keyboard and mouse, connected via USB ports, facilitates interaction with the Raspberry Pi.

For display connectivity, the Raspberry Pi 4 Model B features two micro HDMI ports. This allows for easy connection to displays for visualization purposes.

By adhering to these setup requirements and recommendations, the Raspberry Pi can effectively serve as a platform for receiving and processing data via the MQTT protocol.( <https://www.raspberrypi.com/documentation/computers/getting-started.html>)

To install Node-RED for programming and receiving data, you can follow these steps:

Open the terminal window on your Raspberry Pi.

Use the following command to install Node-RED from the Raspberry Pi OS repositories: sudo apt-get install nodered

Press Enter to execute the command.

Follow any prompts or confirmations that appear during the installation process.

Once the installation is complete, Node-RED will be installed on your Raspberry Pi and ready to use.

Installing Node-RED using the package manager (apt-get) ensures a smooth installation process and allows you to easily manage Node-RED updates in the future.( <https://nodered.org/docs/getting-started/raspberrypi>).To obtain the IP address or link for accessing the Node-RED page online, the command node-red-start can be executed in the terminal. Upon execution, the corresponding IP address, along with the port number (typically 1880), will be displayed. This information can then be used to access the Node-RED editor interface via a web browser.( <https://nodered.org/docs/getting-started/raspberrypi>).

To install the Mosquitto MQTT broker on the Raspberry Pi, follow these steps:

Step 1: Activating Mosquitto Repository

Open the LXTerminal and execute the following commands:

curl -O http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key

sudo apt-key add mosquitto-repo.gpg.key

rm mosquitto-repo.gpg.key

cd /etc/apt/sources.list.d/

sudo curl -O http://repo.mosquitto.org/debian/mosquitto-wheezy.list

sudo apt-get update

Step 2: Installing Packages

Install the Mosquitto packages using the command:

sudo apt-get install mosquitto mosquitto-clients python-mosquitto

After installation, stop the Mosquitto broker using

sudo /etc/init.d/mosquitto stop

Step 3: Configuration

Create a configuration file and add the following lines to restrict anonymous clients:

cd /etc/mosquitto/conf.d/

sudo nano mosquitto.conf

allow\_anonymous false

password\_file /etc/mosquitto/conf.d/passwd

require\_certificate false

Save the file and exit the editor. Then create an empty password file

sudo touch passwd

Use the mosquitto\_passwd tool to create a password hash for the user 'pi':

sudo mosquitto\_passwd -c /etc/mosquitto/conf.d/passwd pi

You will be prompted to enter the password twice. Enter the password according to your preference.( https://iotasmarterplanet.wordpress.com/mqtt/installing-mosquitto-on-to-the-pi/)

With all the necessary libraries installed and everything prepared, it's time to start programming in Node-RED. Below, we outline the required block codes:

MQTT In Block: This block serves as the MQTT input node in Node-RED, facilitating data reception from an MQTT broker.

Function 1: This function node is utilized to extract only the temperature data from the incoming data stream.

Gauge: Representing a gauge node, this block is responsible for displaying data on the Node-RED dashboard. Users can adjust the minimum and maximum values according to their requirements, such as setting a temperature range of 0 to 60 degrees Celsius. Additionally, customization options for the name, unit, and appearance of the gauge are accessible by double-clicking on the block.

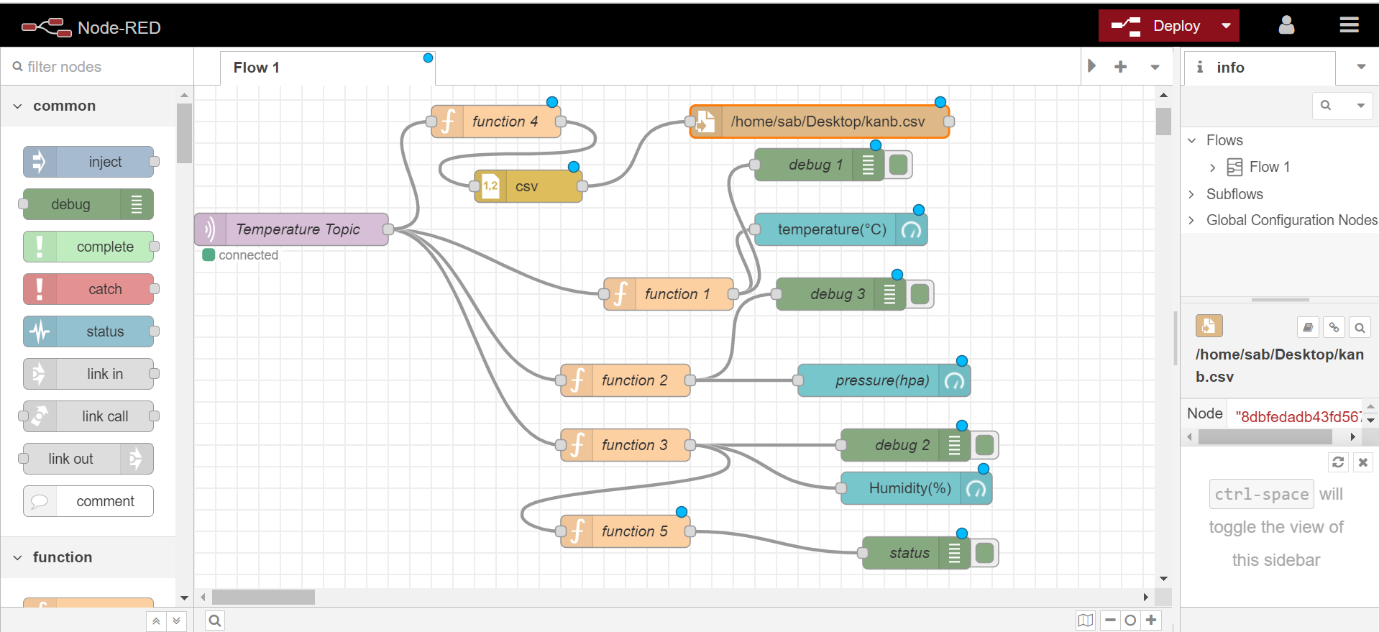
Function 2: This function node extracts the pressure value from the incoming data stream.

Gauge 2: Similar to the previous gauge, this block displays the pressure value on the Node-RED dashboard, typically in hectopascals (hPa). Users have the flexibility to customize the appearance and range of the gauge as needed.

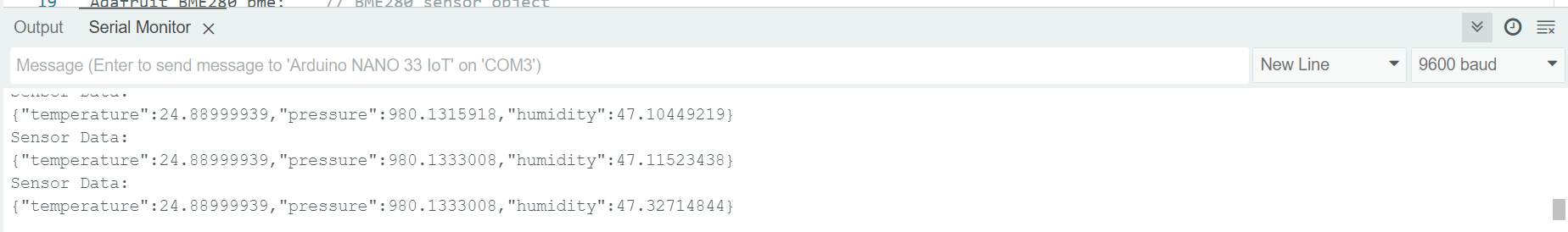
Function 3: Responsible for extracting the relative humidity value, this function node typically operates within a range of 0 to 100 percent.

Gauge 3: This gauge node visualizes the relative humidity value on the Node-RED dashboard, often within a range of 0 to 100 percent.

Function 4: Designed to create a .csv file on the desktop and write real-time data into it, this function node enables data storage on external devices like USB drives or SD cards without requiring a display screen. Users can tailor the file path to suit their preferences.



Results: Once all components are set up and the program is executed, including the debugging and uploading processes, the MQTT communication is initiated. Subsequently, the temperature, pressure, and humidity values are displayed in the serial monitor, as depicted in the illustration below.

Similarly, upon deploying our Node-RED program, we successfully received the data on the Raspberry Pi side. This signifies that the communication setup established between the sensor nodes and the Raspberry Pi was effective. As a result, the Raspberry Pi was able to receive and process the incoming data, allowing for further analysis or visualization through the Node-RED platform. This successful data reception on the receiving end validates the robustness of our deployed solution and ensures that our system is functioning as intended

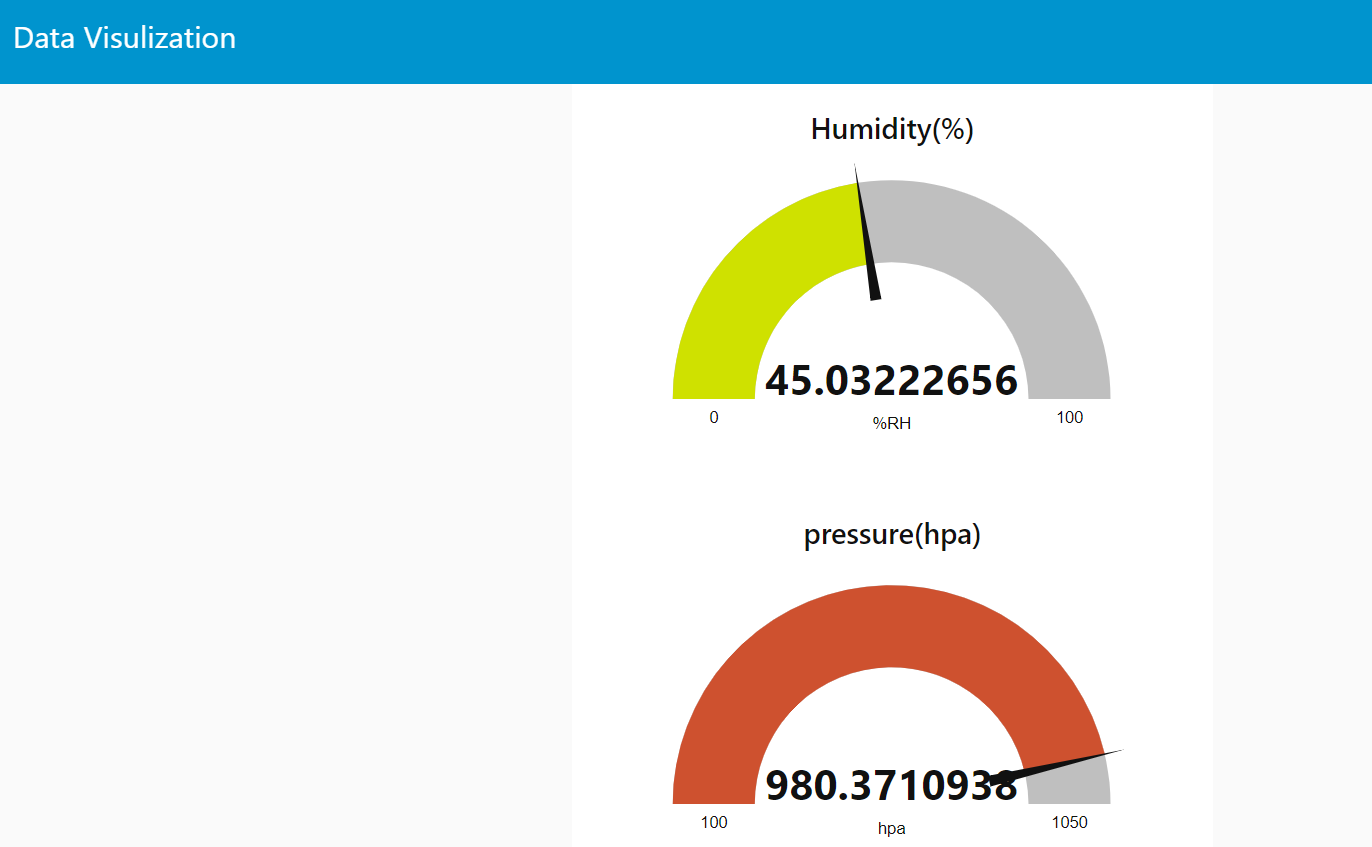
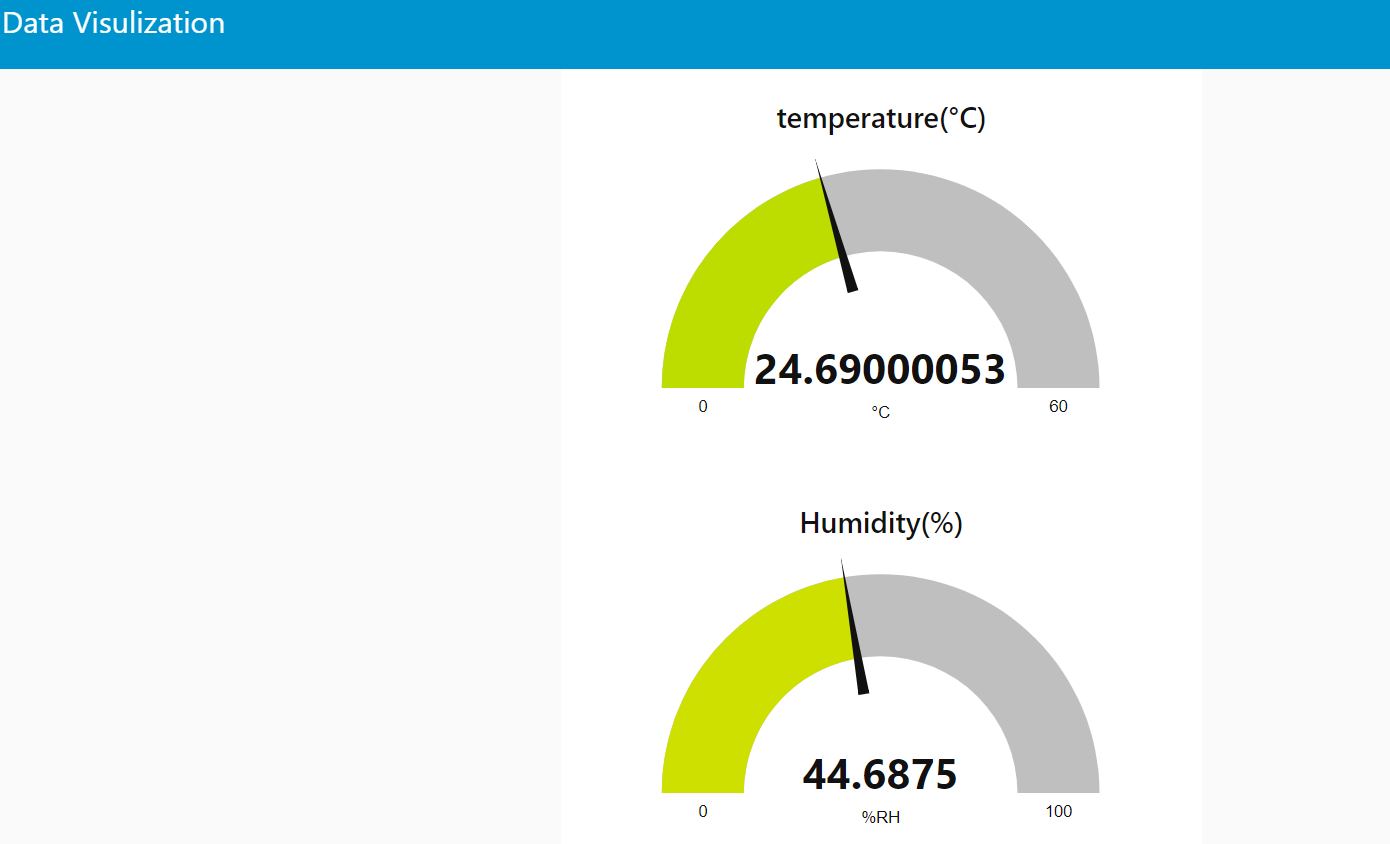
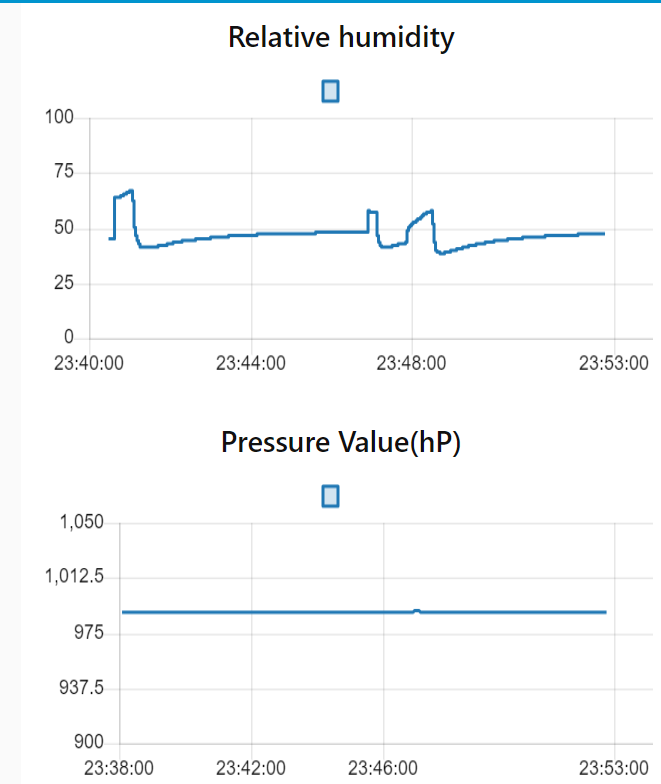
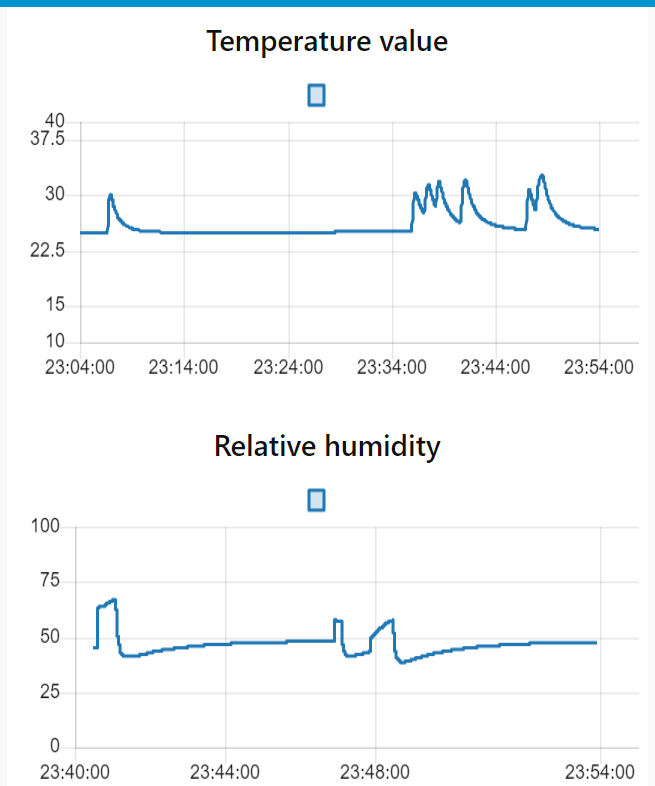
Data visulization: The received data can be observed in the Node-RED debug window. However, for more comprehensive analysis and visualization, we utilize the Node-RED dashboard. This dashboard can be accessed using the same IP address followed by "/ui" in our case, such as 192.168.1.102:1880/ui. Here, "1880" represents the port through which the Node-RED dashboard is accessible. By accessing this dashboard, we can effectively visualize and analyze the received data in a user-friendly interface, facilitating better insights and decision-making.

FIGURE:Show the pressure temperature humidity value

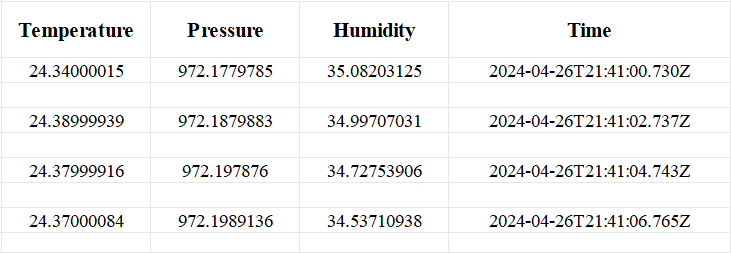
The figure above depicts real-time temperature, pressure, and humidity values. The temperature is indicated as 24.5 degrees Celsius, while the pressure is measured in hPa (hectopascals), and the relative humidity is represented as a percentage ranging from 0 to 100. In this context, 0 signifies completely dry air. To enhance visualization, we have configured the temperature range from 0 to 60 degrees Celsius, the pressure range from 100 to 1050 hPa, and the humidity range from 0 to 100. Adjustments to these ranges can be made by double-clicking on the corresponding gauge block within the Node-RED interface, allowing flexibility to tailor the visualization according to specific requirements and sensor characteristics.

Comparison of different graph: By connecting the chart block in Node-RED, we can visualize our data in various formats such as line graphs, pie charts, and more. Unlike displaying only the current value, these graphs provide a comprehensive view of the data over a specified time interval. This feature enables us to analyze trends and patterns over time, allowing for deeper insights into the data. Whether examining data on a weekly, monthly, or custom timeframe, the visualization capabilities provided by Node-RED empower users to gain valuable insights and make informed decisions based on historical data trends.



The line graph in the above figure depicts the variation of temperature, pressure, and humidity values over a specified period, offering a dynamic representation of these parameters. Notably, the humidity and pressure values fluctuate, as indicated by the varying peaks and troughs in the graph. This fluctuation occurs due to experimental manipulation, such as touching the sensor to the body, altering the immediate environment's humidity and pressure conditions. In contrast, the pressure remains relatively constant, reflecting the stable conditions within the room environment, where positional changes do not significantly impact atmospheric pressure.

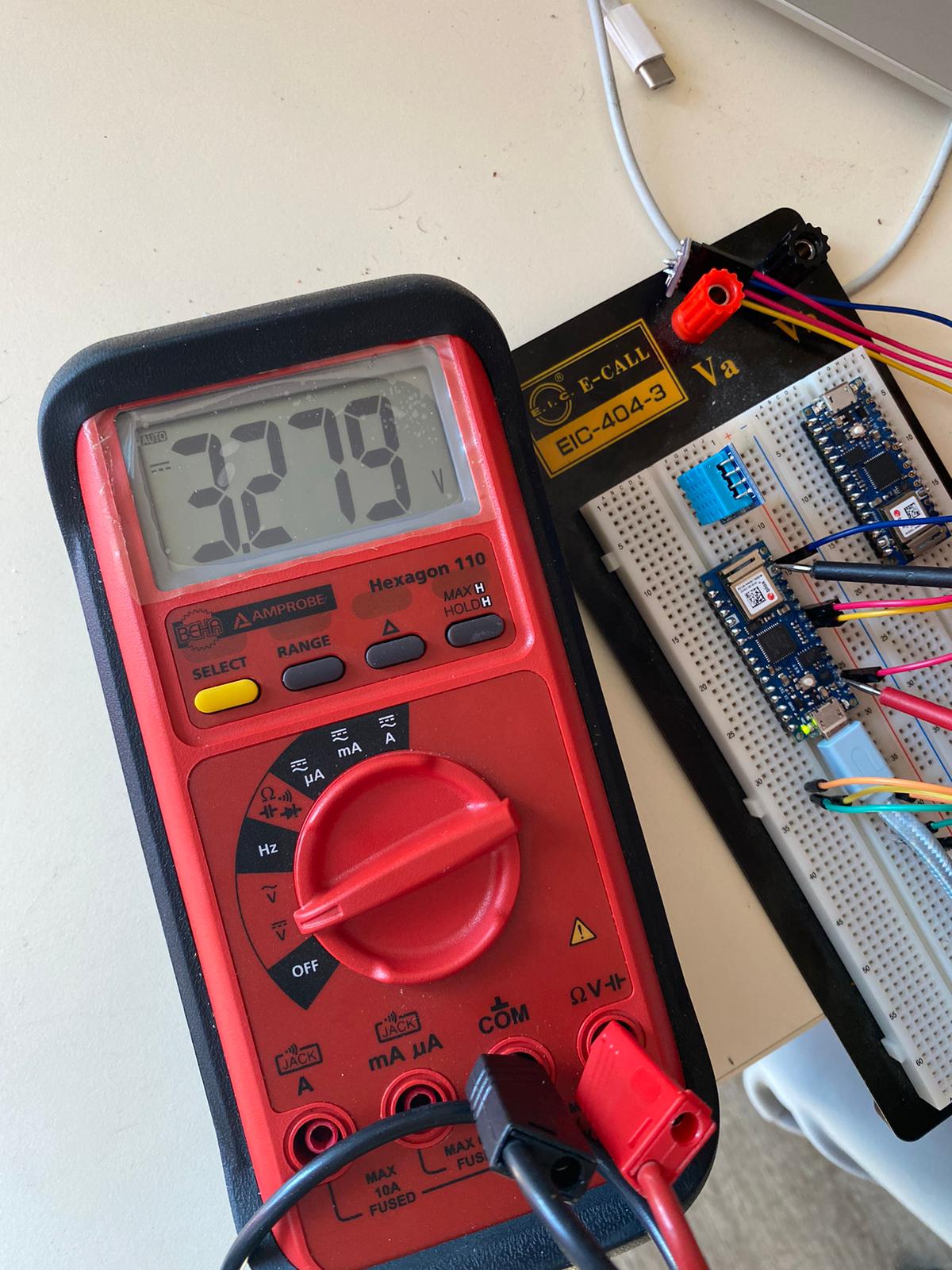
By analyzing trends and previous values, we can forecast future data points using various algorithms such as linear regression, moving averages, exponential smoothing, and more. These algorithms utilize historical data patterns to predict future trends and provide valuable insights for decision-making. Whether it's anticipating future demand, predicting equipment failures, or forecasting weather patterns, leveraging these algorithms allows us to make proactive decisions and mitigate risks. Additionally, by integrating machine learning techniques, we can enhance the accuracy of our forecasts and adapt to changing conditions more effectively.

Data Storage: In order to facilitate efficient data analysis for subsequent research endeavors, the collected data is systematically stored in a CSV (Comma Separated Values) file format. The implementation involves four function blocks dedicated to data management, each designed to write specific data streams into the CSV file. These streams include pressure, temperature, and sensor readings, with each datum accompanied by a timestamp for temporal context. The CSV file is created on the desktop and named 'kanb' for ease of access and organization. 

This structured approach ensures the integrity and accessibility of the collected data for comprehensive analysis and interpretation.

To enhance flexibility and accessibility, the data storage location can be modified to an external USB drive connected to the Raspberry Pi. By adapting the file path accordingly, the data remains securely stored and can be conveniently analysed at any given time. This configuration ensures portability and ease of access, enabling seamless data analysis processes while leveraging the Raspberry Pi's capabilities for efficient data management.

Power Consumption of Arduino Nano 33 IOT: One of the objective of this section is to calculate the power consumption of the Arduino Nano 33 module. This is a crucial consideration in assessing the device's operational efficiency. The Arduino Nano 33 typically consumes 3.279 V and 1.35 mA of power as shown in the figure



Using the formula

P = V \* I (where P represents power, V represents voltage, and I represents current), we calculate:

𝑃=3.279 V×1.35 mA

P=4.4265 mW

P=3.279V×1.35mA

P=4.4265mW

This indicates that the Arduino Nano 33 consumes approximately 4.4265 milliwatts of power during operation.

To evaluate the annual power consumption, we apply the formula:

𝐸=𝑃×365×24

Substituting the calculated power consumption value:

𝐸=4.4265 mW×365×24

E=38.7666 mWh

E=4.4265mW×365×24

E=38.7666mWh

Thus, the Arduino Nano 33 consumes approximately 38.7666 milliwatt-hours of energy over the course of a year.

For scenarios where the Arduino Nano 33 operates outdoors on battery power, considerations must be made regarding battery selection and efficiency. Utilizing a lithium-ion battery with a capacity of 2000mAh and an operating voltage of 4.5V as a reference, the expected runtime can be determined using the formula:

𝑇=2000 mAh/1.35 mA

T​=1481.48hours

This indicates that the Arduino Nano 33 can operate continuously for approximately 1481.48 hours, or around 61.7 days, on a single charge. However, practical considerations such as battery discharge curves, intermittent power spikes during data transmission, and battery aging effects must be taken into account to optimize operational efficiency."

The power rating obtained for the BME280 sensor, as measured in our experiment, aligns with the thermal characteristics reported in the referenced paper. Specifically, our measured power consumption falls within the range of values provided for the BME280 sensor's sleep mode (0.17–0.36 μW) and active mode (595–1260 μW). This consistency reinforces the validity of our experimental setup and measurements, indicating that our findings accurately reflect the expected performance of the BME280 sensor under different operating conditions(https://www.bosch-sensortec.com/media/boschsensortec/downloads/product\_flyer/bst-bme280-fl000.pdf)Top of Form

Security Consideration: The data being transmitted via the MQTT protocol is securely configured, utilizing the local WiFi network for communication. This configuration requires the use of both a SSID (Service Set Identifier) and a password, ensuring that only authorized devices can access and exchange data within the network.

To receive this data, one must first connect to the same local WiFi network by providing the correct SSID and password. Once connected, access to the data becomes available through the network's assigned IP address. This IP address can be used not only for data reception but also for accessing other network services, such as browsing the programming interface of tools like Node-RED and visualizing the transmitted data through dedicated visualization platforms.

In summary, the secure configuration of SSID and password ensures the integrity and privacy of the transmitted data, while access to the local WiFi network enables seamless interaction with the data and related services, enhancing the overall functionality and utility of the system

Accessing the data doesn't necessitate an external display monitor; instead, it can be conveniently done using a smartphone or any device with network capabilities. Once connected to the local network by entering the correct SSID and password, users can simply input the corresponding IP address into their smartphone's browser or a dedicated application.

This streamlined process enables seamless access to the data without the need for additional hardware. It enhances flexibility by allowing users to monitor and interact with the data remotely, using their preferred devices. Whether at home, in the office, or on the go, users can effortlessly tap into the network and access the valuable data insights provided by the system.

"Future Work and Improvements:

Cloud Integration: Consider integrating the data into cloud platforms like AWS to enable remote accessibility from any location on Earth. This would enhance the scalability and flexibility of the system.

Security Enhancements: Implement a username and password system within the Node-RED platform to add an extra layer of authentication. Additionally, explore options for role-based access control to manage different levels of access for users.

Enhanced Encryption for Energy Systems: Investigate the implementation of advanced encryption techniques tailored specifically for energy-related data. This could involve exploring options such as end-to-end encryption, homomorphic encryption, or quantum encryption to ensure the security and privacy of sensitive data.

Energy Efficiency Considerations: While implementing these improvements, it is essential to consider energy efficiency. Optimize data transmission protocols and minimize computational overhead to ensure minimal impact on energy consumption.

These future enhancements aim to further enhance the reliability, accessibility, and security of the system, paving the way for more robust and efficient energy management solutions

Presentation of your collected data (temperature and humidity readings)

Use graphs, charts, and tables to showcase the data.

Analysis of security measures and their effectiveness

Discussion on system reliability based on experimental results

Evaluation of power consumption and energy efficiency

Discussion (6-8 pages)

Interpretation of the results

Comparison with existing studies or standards

Analysis of the performance, security, reliability, and energy aspects of your setup

Discuss the implications of your findings

Conclusion (2-3 pages)

Summary of the key findings

Contributions of your work to the field

Limitations and future work

Recommendations (1-2 pages)

Suggestions for further research or improvements related to security, reliability, and energy efficiency

References

Appendix:

#include <Wire.h> // Include the Wire library for I2C communication

#include <Adafruit\_Sensor.h> // Include the Adafruit sensor library

#include <Adafruit\_BME280.h> // Include the Adafruit BME280 sensor library

#include <WiFiNINA.h> // Include the WiFiNINA library for Wi-Fi communication

#include <PubSubClient.h> // Include the PubSubClient library for MQTT communication

#include <ArduinoJson.h> // Include the ArduinoJson library for JSON data handling

#define SEALEVELPRESSURE\_HPA (1013.25) // Define the local sea level pressure for altitude calculations

#define SECRET\_SSID "TP-LINK\_AC2C06" // Wi-Fi network SSID

#define SECRET\_PASS "38951107" // Wi-Fi network password

const char\* mqtt\_server = "192.168.1.102"; // IP address of the MQTT broker

const char\* mqtt\_topic = "sensor\_data"; // MQTT topic to publish sensor data

WiFiClient wifiClient; // Instance of WiFiClient used for Wi-Fi communication

PubSubClient client(wifiClient); // Instance of PubSubClient for MQTT communication

Adafruit\_BME280 bme; // BME280 sensor object

void setup() {

Serial.begin(9600); // Initialize serial communication at 9600 baud

connectWiFi(); // Connect to Wi-Fi network

client.setServer(mqtt\_server, 1883);// Set MQTT server and port

if (!bme.begin(0x76)) { // Initialize BME280 sensor with I2C address 0x76

Serial.println("Could not find a valid BME280 sensor, check wiring!");

while (1); // Halt program if sensor is not found

}

Serial.println("BME280 sensor found.");

// Set BME280 sensor sampling settings

bme.setSampling(Adafruit\_BME280::MODE\_NORMAL, // Operating mode

Adafruit\_BME280::SAMPLING\_X2, // Temperature oversampling

Adafruit\_BME280::SAMPLING\_X16, // Pressure oversampling

Adafruit\_BME280::SAMPLING\_X16, // Humidity oversampling

Adafruit\_BME280::FILTER\_X16, // Filtering

Adafruit\_BME280::STANDBY\_MS\_500); // Standby time

}

void loop() {

if (!client.connected()) { // Check if MQTT client is connected

reconnect(); // Reconnect if not connected

}

client.loop(); // Maintain MQTT connection

// Read sensor data from BME280 sensor

float temperature = bme.readTemperature(); // Read temperature in Celsius

float pressure = bme.readPressure() / 100.0F; // Read pressure in hPa (Pa to hPa conversion)

float humidity = bme.readHumidity(); // Read humidity in percentage

// Create JSON object to store sensor data

StaticJsonDocument<200> jsonDoc; // Define JSON document with buffer size 200

// Populate JSON object with sensor data

jsonDoc["temperature"] = temperature; // Add temperature data to JSON object

jsonDoc["pressure"] = pressure; // Add pressure data to JSON object

jsonDoc["humidity"] = humidity; // Add humidity data to JSON object

// Serialize JSON object to a string

String jsonString; // Define a string to store JSON data

serializeJson(jsonDoc, jsonString); // Serialize JSON object to string

// Print JSON string to Serial Monitor

Serial.println("Sensor Data:"); // Print header for sensor data

Serial.println(jsonString); // Print JSON string

// Publish JSON string via MQTT

client.publish(mqtt\_topic, jsonString.c\_str()); // Publish JSON string to MQTT topic

delay(2000); // Delay between sensor readings

}

void connectWiFi() {

Serial.println("Connecting to WiFi"); // Print status message for Wi-Fi connection attempt

WiFi.begin(SECRET\_SSID, SECRET\_PASS); // Connect to Wi-Fi network with SSID and password

while (WiFi.status() != WL\_CONNECTED) { // Wait until Wi-Fi connection is established

delay(500);

Serial.print("."); // Print dot to indicate connection attempt

}

Serial.println(""); // Print new line

Serial.println("WiFi connected"); // Print status message for successful Wi-Fi connection

Serial.println("IP address: "); // Print header for IP address

Serial.println(WiFi.localIP()); // Print local IP address

}

void reconnect() {

while (!client.connected()) { // Loop until MQTT connection is established

Serial.print("Attempting MQTT connection..."); // Print status message for MQTT connection attempt

if (client.connect("arduinoClient")) { // Attempt to connect to MQTT broker with client ID "arduinoClient"

Serial.println("connected"); // Print status message for successful MQTT connection

} else {

Serial.print("failed, rc="); // Print status message for failed MQTT connection

Serial.print(client.state()); // Print MQTT connection state

Serial.println(" try again in 5 seconds"); // Print suggestion for reconnection delay

delay(5000); // Delay for 5 seconds before retrying

}

}

}

Sample MQTT messages

Node-RED flows

Page Allocation

Abstract: 1-2 pages

Introduction: 4-5 pages

Literature Review: 10-12 pages

Methodology: 8-10 pages

Results: 8-10 pages

Discussion: 6-8 pages

Conclusion: 2-3 pages

Recommendations: 1-2 pages

References

Appendices