

HO CHI MINH CITY - UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



EMBEDDED SYSTEM ASSIGNMENT

SEMESTER: 1 ACADEMIC YEAR: 2023-2024

MAJOR: COMPUTER ENGINEERING
EDUCATION PROGRAM: HIGH QUALITY PROGRAM

TOPIC

**AN IOT-BASED SYSTEM USING RS485 COMMUNICATION
PROTOCOL TO MONITOR CLIMATE CONDITIONS**

Advisor:	Ph.D. Phạm Hoàng Anh	
Students:	Lê Nhật Đăng	2052950
	Đỗ Quang Khanh	2053106
	Đinh Trục Tâm	2053415

HO CHI MINH CITY, 12/ 2023

Content

1	Member list & Contribution	2
2	About the project	3
2.1	Requirements	3
2.1.1	Requirements Gathering	3
2.1.1.1	Problem Statement	3
2.1.1.2	Scope Definition	3
2.1.2	Requirements analysis	4
2.1.2.1	Identify the variables in performance, hardware, firmware, software	4
2.1.2.2	Estimate cost, complexity	5
2.1.2.3	Determine tradeoffs	5
2.2	Implementation	6
2.2.1	System Design	6
2.2.1.1	Functional Requirements	6
2.2.1.2	Non-functional Requirements	6
2.2.1.3	Hardware-Software Selection	7
2.2.1.4	IoT Architecture	8
2.2.1.5	Block Diagram	8
2.2.2	System Implementation	9
2.2.2.1	Data Frame processing	11
2.2.2.2	WSS02 sensor setup	13
2.2.2.3	OLED display setup	14
2.2.2.4	Adafruit IO setup	14
2.2.2.5	A problem with Adafruit IO server	16
2.2.3	System Integration and Testing	17
2.3	Source code	18
2.4	Incomplete Tasks and Future Work	18
2.4.1	Incomplete Tasks	18
2.4.2	Future Directions	18
2.5	Bibliography	19

1 Member list & Contribution

No.	Fullname	Student ID	% Work
1	Lê Nhật Đăng	2052950	100%
2	Đỗ Quang Khanh	2053106	100%
3	Đinh Trục Tâm	2053415	100%

2 About the project

2.1 Requirements

2.1.1 Requirements Gathering

2.1.1.1 Problem Statement

Monitoring and analyzing climate conditions is essential across various fields, including agriculture, environmental science, and industrial operations. Accuracy and most importantly - real-time response of data is a must for decision-making, resource management, and environmental preservation. Currently, existing climate monitoring systems often lack the necessary precision, scalability, and accessibility, which has led to the need for a more advanced and robust solution.

According to some research, current climate monitoring systems often suffer from the following limitations:

- *Limited Precision:* Many existing systems provide climate data with limited accuracy, which hinders the ability to make informed decisions based on this data.
- *Lack of Scalability:* Traditional monitoring systems struggle to accommodate additional sensors or expand to cover larger areas efficiently.
- *Inefficient Data Retrieval:* Data retrieval and analysis from legacy systems can be time-consuming and prone to errors, as they may rely on manual collection methods.
- *Security Concerns:* Data security is a significant concern, especially when monitoring sensitive environmental or industrial conditions.
- *Outdated Communication Protocols:* Some older systems use outdated communication protocols that limit their connectivity and interoperability with modern technologies.
- *Inadequate User Interfaces:* The user interfaces of many current systems lack user-friendliness, making it challenging for end-users to interact with the data effectively.
- *High Maintenance and Operational Costs:* Legacy systems often require frequent maintenance and operational expenses that are not cost-effective.

2.1.1.2 Scope Definition

In this project, we hopefully provide a solution called an IoT-based system using RS485 communication protocol to monitor climate conditions. In the scope of the project, a wind sensor is used to demonstrate but the system can be expanded with various types of sensor if they have RS485 connection. Some primary objectives are expected to be accomplished as follows:

- *Improved Precision:* Develop a monitoring system that provides highly accurate and real-time climate data by integrating advanced sensors.
- *Scalability:* Create a system that can easily accommodate additional sensors and scale to cover extensive geographical areas or multiple monitoring locations.

- *Efficient Data Retrieval and Analysis*: Enable efficient and automated data collection and analysis, reducing the manual effort required.
- *Data Security*: Implement robust security measures to protect sensitive climate data from unauthorized access or tampering.
- *Stable Communication Protocol*: Utilize the RS485 communication protocol for reliable and efficient data transfer.
- *Visual Interface*: Design an intuitive web-based dashboard for end-users to monitor and interact with climate data seamlessly.
- *Cost-Efficiency*: Develop a system with lower maintenance and operational costs to ensure long-term operation.

2.1.2 Requirements analysis

2.1.2.1 Identify the variables in performance, hardware, firmware, software

Performance:

- *Sensors accuracy*: Sensors quality is essential in measuring wind speed, wind direction, soil moisture, etc. The expected accuracy number should be in the range 80% to 90% of the actual environmental contexts.
- *Network reliability*: The RS485 protocol in transmitting and receiving raw data. And MQTT protocol to publish data from microcontroller to cloud server.
- *Power efficiency*: Efficient use of power by the sensor nodes and MCUs. Measured by battery life, solar harvesting efficiency, etc.
- *Scalability*: The ability to easily handle more sensor nodes in case of expanding farming density area.
- *Durability*: Robustness and weather resistance of the sensor nodes deployed in the field.
- *Data completeness*: Percentage of expected sensor readings successfully received by the MCU.
- *Alert latency*: The time taken to detect threshold breaches and send alerts/notifications to applications.
- *Ease of deployment*: As simple as possible setup and configuration of sensors, gateways and backend infrastructure including software applications.
- *Data visualization*: Usefulness of data dashboards, analytic and visualization tools.

Hardware:

- Sensors accuracy and precision
- Sensors power consumption
- Sensors durability/weatherproofing
- Wiring range

- Latency

Firmware:

- Sensor reading interval optimization
- Data transmission optimization
- Power management

Software:

- Data ingestion rate
- Data validation and error checking
- Database read/write performance
- Web dashboard responsiveness
- Automated alert response time
- Data visualization interactivity
- Ease of software upgrades/updates

2.1.2.2 Estimate cost, complexity

No.	Devices	Cost
1	WSS-02 Wind speed and direction sensor	2,200,000 VND
2	Arduino R3 UNO	135,000 VND
3	NodeMCU ESP8266 WiFi	115,000 VND
4	TTL RS485 communication module	9,000 VND
5	1.3inch OLED display - driver SH110X	110,000 VND
6	Batteries, Wires and Breadboards	50,000 VND

2.1.2.3 Determine tradeoffs

RS485 protocol use wires to communicate with other devices. To some extent, this feature is great in terms of data completeness and stable connection as long as the power source and wires are produced in good quality. Due to the fact that data is collected and transmitted through wires, it cannot be lost during transmitting session unless there are external factors such as water and power lost.

Nowadays, more complex and convenient IoT systems are considered to use wireless communication instead of a traditional way as wires. LoRa, Zigbee or even WiFi has been being applied widely. Although the problem can be occurred with connection range or stability, these wireless technologies are definitely resolve the disadvantages when using wires. With easy installation, convenience and efficiency in data transmission, wireless connection seems to outweigh the advantages of traditional wiring connection.

In this project, RS485 communication are utilized for data transmission and in the near future, we hope that the system can be re-implemented or developed using wireless connection in order to handle all the drawbacks of wiring connection.

2.2 Implementation

2.2.1 System Design

2.2.1.1 Functional Requirements

1. Data Collection

- The system must be able to collect data from various environmental sensors.
- Data should include temperature, humidity, soil moisture, and other relevant microclimate parameters.

2. Data Transmission

- Data collected by the sensors must be transmitted using RS485 protocol.
- Data transmission should occur at regular intervals. ($0.1s < \text{transmission time} < 1s$)

3. Data Storage

- The system should store collected data in a database or cloud storage for historical analysis.
- For our system, data is stored in cloud service platform.

4. Real-Time Monitoring:

- The system should provide real-time monitoring of microclimate conditions.
- Users should be able to access current data through a web interface.

5. Alerting System

- Implement an alerting mechanism to notify users when specific conditions meet or exceeds the defining threshold.

6. Visual Dashboard

- Develop an intuitive web dashboard for interacting with the system.

7. Scalability

- Design the system to scale up easily to accommodate additional sensors or devices.

2.2.1.2 Non-functional Requirements

1. Reliability

- The system should have a minimum uptime of 95%. (5% for unwanted cases: power cut, loss connectivity, errors in sensors and maintenance)
- Sensor data loss rate should be less than 10%. (e.g: in 1 minutes, expect to receive 20 data, so the maximum number of loss data should be 2 data)

2. Scalability

- The system should be able to handle a growing number of sensors without performance degradation.

3. Response Time

- Expected updated data time: 20 data packets / 1 minutes. (can be modified easily through programming)
- Expected warning time: 1 minute after threats are found.

4. Data Accuracy

- Sensors should provide accurate and reliable data. ($\pm 5\%$ accuracy)
- Implement data validation and error correction mechanisms.

5. Power Efficiency

- Optimize power consumption to extend the life of battery-operated sensors.

6. Maintainability

- Make the system easy to maintain, update, and troubleshoot. (smart wiring, box for protection, update information remotely)

7. Dashboard Interface Usability

- The dashboard interface should be as simple as possible and shows only necessary information.

2.2.1.3 Hardware-Software Selection

Hardware-dependent Infrastructure

1. Wind speed/direction sensor – WSS-02

(a) Specifications

- Wind speed range: $0 \sim 60m/s$
- Wind direction range: $0 \sim 360^\circ$
- Start wind speed: $\leq 0.3m/s$
- Accuracy: $\pm(0.3 + 0.03V)m/s, \pm 1^\circ$
- Input Power: $DC5\sim 24V$
- Interface: $RS485$
- Working Temperature: $-30^\circ C \sim 70^\circ C$
- Working Humidity: $< 100\%$ (no dewing)
- Power Consumption: $13mA\sim 12V$

(b) Dimension

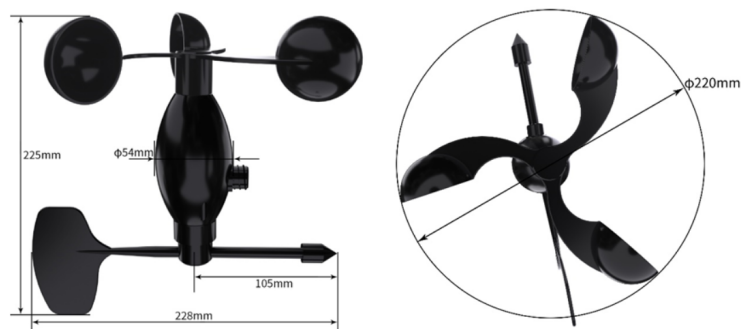


Figure 1: Dimension of the wind sensor

2. Boards and Peripherals:

- *Microcontroller*: Arduino R3 UNO
- *WiFi module*: NodeMCU ESP8266 WiFi
- *Communicating circuit*: TTL RS485 module
- *Display*: 1.3inch OLED display
- *Communication protocol*: RS485, UART, WiFi, I²C

Network Server: Adafruit IO

Software-based Infrastructure:

1. Integrated Development Environment (IDE): Arduino IDE, Visual Studio Code
2. Dashboard: Adafruit IO

2.2.1.4 IoT Architecture

Here is the general architecture for an IoT-based system:

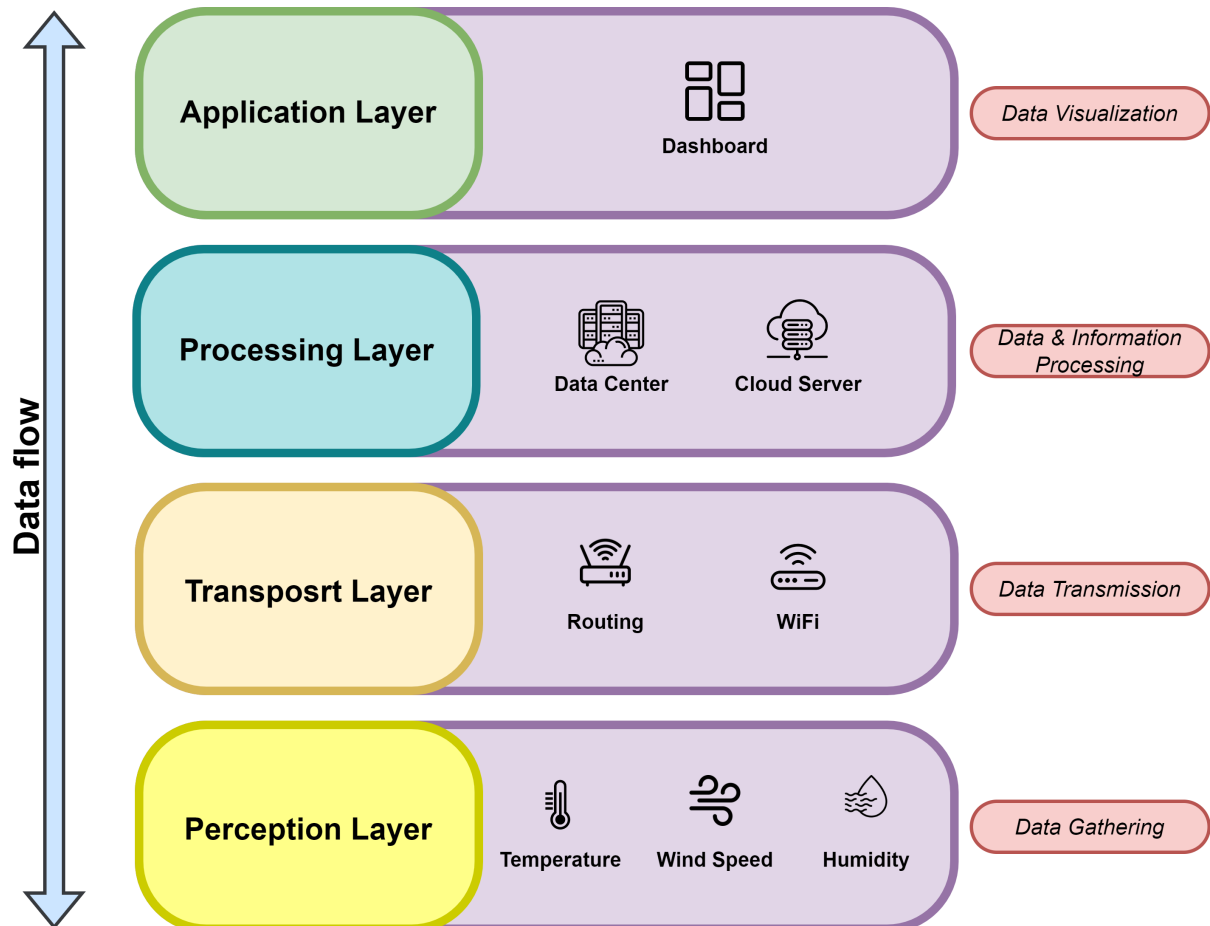


Figure 2: IoT Architecture

2.2.1.5 Block Diagram

Here is the block diagram:

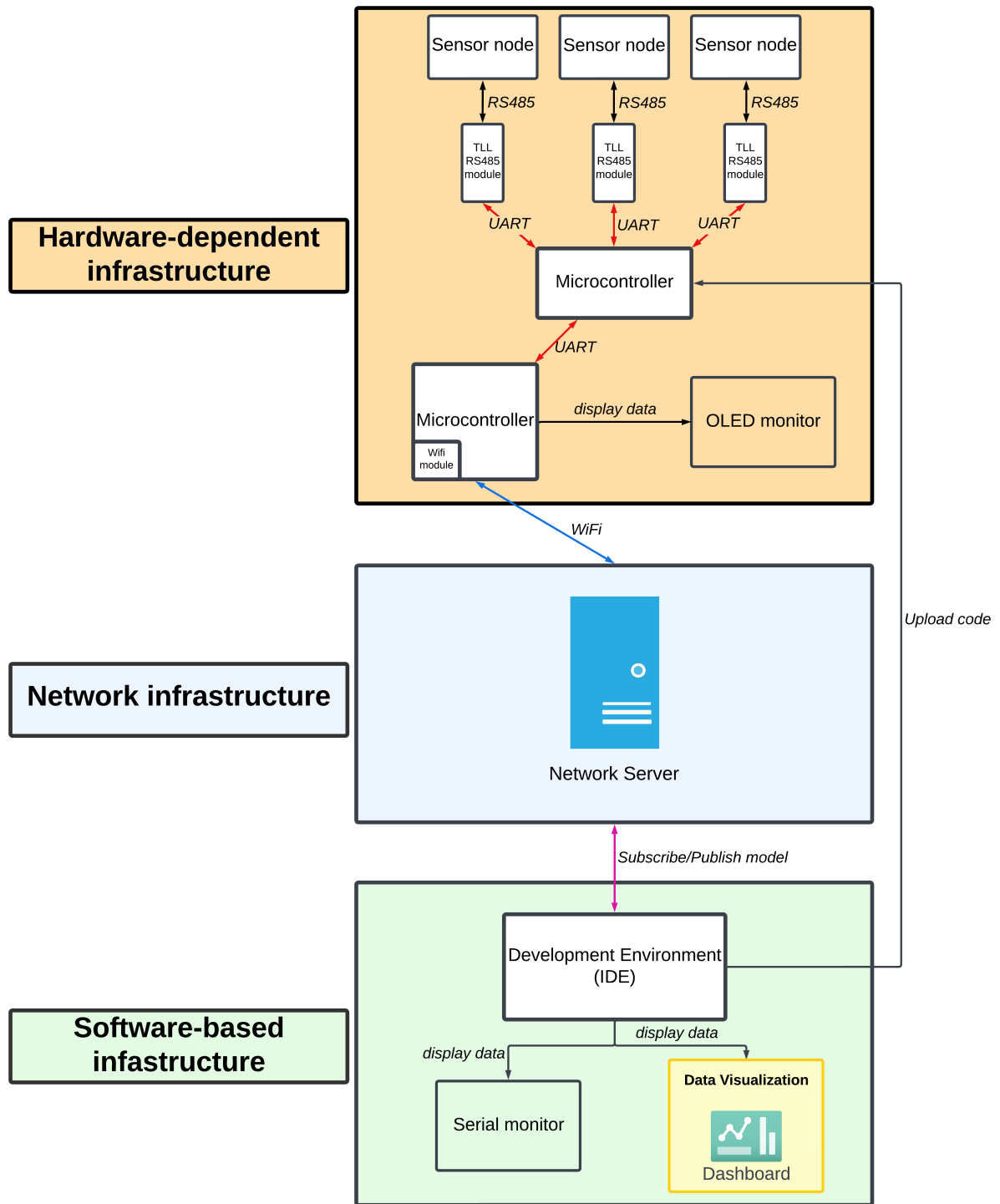
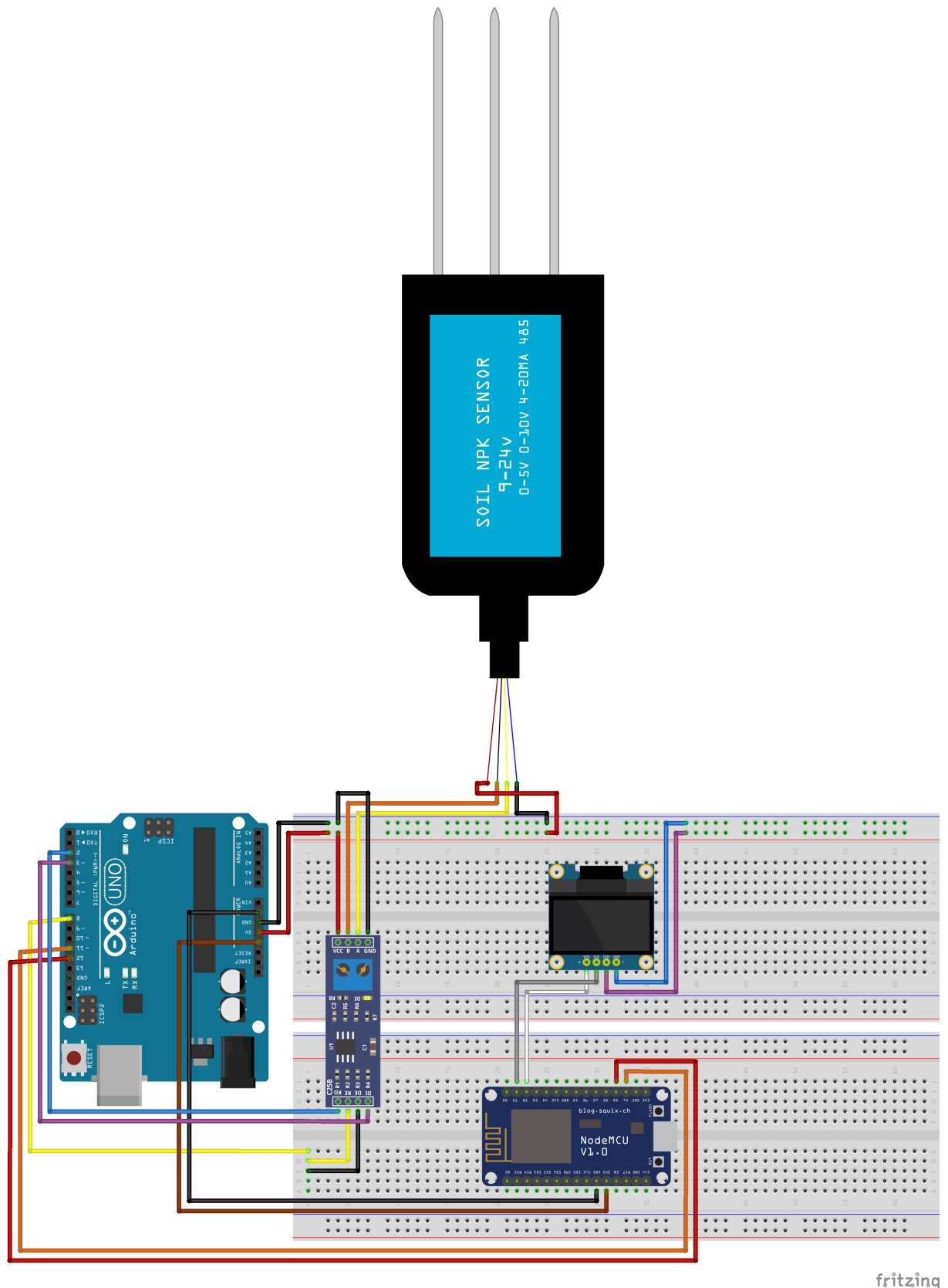


Figure 3: Block Diagram

2.2.2 System Implementation

We provide a schematic of a system as below:



fritzing

Figure 4: Schematic of the system

We can't find the schematic image of our given sensor. Because of that, we decide to use another type of sensor with RS485 communication to replace the image of wind sensor. The sensor, however, still has the right

communication connection.

Here's the connection in words:

- Sensor and TTL RS485 module:

No.	WSS-02 sensor	TTL RS485 module
1	VCC	VCC
2	GND	GND
3	RS485-A	A
4	RS485-B	B

- Arduino R3 UNO and TTL RS485 module:

No.	Arduino R3 UNO	TTL RS485 module
1	PD2 (pin 2)	RO
2	PD3 (pin 3)	DI
3	PB0 (pin 8)	DE + RE

- Arduino R3 UNO and NodeMCU ESP8266 WiFi:

No.	Arduino R3 UNO	NodeMCU ESP8266 WiFi
1	3.3V	3V3
2	GND	GND
3	PB3 (pin 11)	TX
4	PB4 (pin 12)	RX

- NodeMCU ESP8266 WiFi and OLED display:

No.	NodeMCU ESP8266 WiFi	OLED display
1	5V from Breadboard	VCC
2	GND	GND
3	D1	SCK
4	D2	SDA

2.2.2.1 Data Frame processing

In order to communicate with the **WSS02** sensor, using RS485 communication is needed. The format of Modbus-RTU communication protocol is described as follow:

- Initial structure ≥ 4 bytes of time
- Address code = 1 byte
- Function code = 1 byte
- Data area = N bytes
- Incorrect revision = 16 bytes CRC code

- Ending structure ≥ 4 bytes time

Host query frame and slave response frame structure

Host query frame structure		Slave response frame structure	
Address Code	1 byte	Address Code	1 byte
Function Code	1 byte	Function Code	1 byte
Register Origin Address	2 bytes	Effective Bytes	1 byte
Register Length	2 bytes	Data Area 1	2 bytes
Check code Low Bit	1 byte	Data Area N	2 bytes
Check Code High Bit	1 byte	Revision Code	2 bytes

Read the value from wind speed and wind direction sensor

In order to read data from the **WSS02** sensor, the master needs to send the request to the sensor. The sensor will response to the master after getting the request from the master. The picture below shows the inquiry frame and response frame of the **WSS02** sensor.

master→slave							
original address	function code	start register high	start register low	start address high	start address low	CRC16 low	CRC16 high
0X07	0X06	0X00	0X30	0X00	0X02	0X08	0X62
Query data:							
master→slave							
address	function code	start register address high	start register address low	Register length high	Register length is low	CRC16 low	CRC16 high
0X07	0X03	0X00	0X00	0X00	0X04	0X44	0X6F

Figure 5: The Inquiry frame and Response frame

The image below is an example response frame from the **WSS02** sensor:

slave → master					
address	0X07				
function code	0X03				
Data length	0X08				
Register 0 data high	0X00	wind speed : 3.6m/s			
Register 0 data low	0X24				
Register 1 data high	0X00	wind scale : 3级			
Register 1 data low	0X03				
Register 2 data high	0X02	wind angle : 66.6°			
Register 2 data low	0X9A				
Register 3 data high	0X00	Wind direction: east-northeast			
register 3 data low	0X03				
CRC16 low	0XCA				
CRC16 high	0XCB				

Figure 6: An example response frame from the sensor

Decoding and Calculation

- **Wind speed:** $0x0024(H) = 36(Decimal)$. The value of wind speed is 3.6 (m/s).
- **Wind angle:** $0x029A(H) = 666(Decimal)$. The value of wind angle is 66.6°.
- **Wind direction:** $0x0003(H) = 3(Decimal)$. The value of wind direction is east-northeast.

2.2.2.2 WSS02 sensor setup

The <SoftwareSerial.h> library is needed for the **WSS02** sensor to connect to the Arduino. To establish the connection with the **WSS02** sensor, the Arduino must send the request frame to it.

```

1 //start the tranmission from master to slave
2 digitalWrite(DE_RE, HIGH);
3 delay(10);
4
5 // send the request frame to the WSS02 sensor
6 mod.write(sensorRequest, sizeof(sensorRequest));
7 // end the tranmission from master to slave
8 //and start receiving data from slave to master
9 digitalWrite(DE_RE, LOW);

```

In the above code, *sensorRequest* stores the value of inquiry frame. After that, when the Arduino board receives the response frame from the **WSS02** sensor, some vital data from the response frame is displayed in serial monitor as the image below:

```
7 3 8 0 12 0 2 6 AA 0 8 E1 F0
Wind speed: 1.80 m/s
Wind direction: south and 8
Wind angle: 170.60°
7 3 8 0 F 0 1 D A9 0 0 9B 13
Wind speed: 1.50 m/s
Wind direction: north and 0
Wind angle: 349.70°
7 3 8 0 D 0 1 A 2D 0 C F9 8B
Wind speed: 1.30 m/s
Wind direction: west and C
Wind angle: 260.50°
7 3 8 0 C 0 1 3 E9 0 4 AA EC
Wind speed: 1.20 m/s
Wind direction: east and 4
Wind angle: 100.10°
7 3 8 0 B 0 1 4 95 0 5 DD 40
Wind speed: 1.10 m/s
Wind direction: east-southeast and 5
Wind angle: 117.30°
```

Figure 7: Display the data from sensor

2.2.2.3 OLED display setup

We need to install these libraries to config the OLED display: <Adafruit_SH110X.h>, <Adafruit_GFX.h>, <Adafruit_GrayOLED.h>, <Adafruit_SPITFT_Macros.h>, <Adafruit_SPITFT.h>, <Wire.h>, <gfxfont.h>

We also need to configure some specifications to activate the OLED display:

```
1 // I2C default address
2 #define i2c_Address 0x3c //initialize with the I2C addr 0x3C
3
4 // OLED display width and height
5 #define SCREEN_WIDTH 128 // OLED display width, in pixels
6 #define SCREEN_HEIGHT 64 // OLED display height, in pixels
7 #define OLED_RESET -1 // QT-PY / XIAO
8
9 // Create an object called "display"
10 Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT,
11 &Wire, OLED_RESET);
```

2.2.2.4 Adafruit IO setup

First, we need to add this library in order to communicate with the Adafruit server: <AdafruitIO_WiFi.h>

In the code, we add the library, open the **config.h** file and configure some parameters for connectivity:

```
1 #define IO_USERNAME "<your Adafruit username>"
2 #define IO_KEY "your Adafruit key"
```

```
3  #define WIFI_SSID    "your WiFi SSID"
4  #define WIFI_PASS    "your WiFi password"
```

Then, we comeback to the main file and create 3 objects representing 3 feeds on Adafruit IO:

```
1  // Initialize feeds on Adafruit
2  AdafruitIO_Feed *speed      = io.feed("wind_speed");
3  AdafruitIO_Feed *angle      = io.feed("wind_angle");
4  AdafruitIO_Feed *direction = io.feed("wind_direction");
```

In the setup() function, we add the above code. If all above parameters are provided correctly, the NodeMCU ESP8266 WiFi is now connected to Adafruit server.

```
1  void setup() {
2      // start the serial connection
3      Serial.begin(9600);
4      // wait for serial monitor to open
5      while(! Serial);
6      Serial.print("Connecting to Adafruit IO");
7      // connect to io.adafruit.com
8      io.connect();
9      // wait for a connection
10     while(io.status() < AIO_CONNECTED) {
11         Serial.print(".");
12         delay(500);
13     }
14     // we are connected
15     Serial.println();
16     Serial.println(io.statusText());
17 }
```

Whenever we want to publish data to the corresponding feed, we can use this line:

```
1  <object created>->save(<data want to publish>);
```

Finally, we add this command to the loop() function to run our program:

```
1  io.run();
```

We create 3 feeds in the Adafruit server named: "wind_speed", "wind_angle" and "wind_direction" and a dashboard to display these feeds as images below:

<input type="checkbox"/> wind_angle	wind-angle	201.50	about 3 hours ago
<input type="checkbox"/> wind_direction	wind-direction	S-SW	about 3 hours ago
<input type="checkbox"/> wind_speed	wind-speed	00.10	about 3 hours ago

Figure 8: Feeds

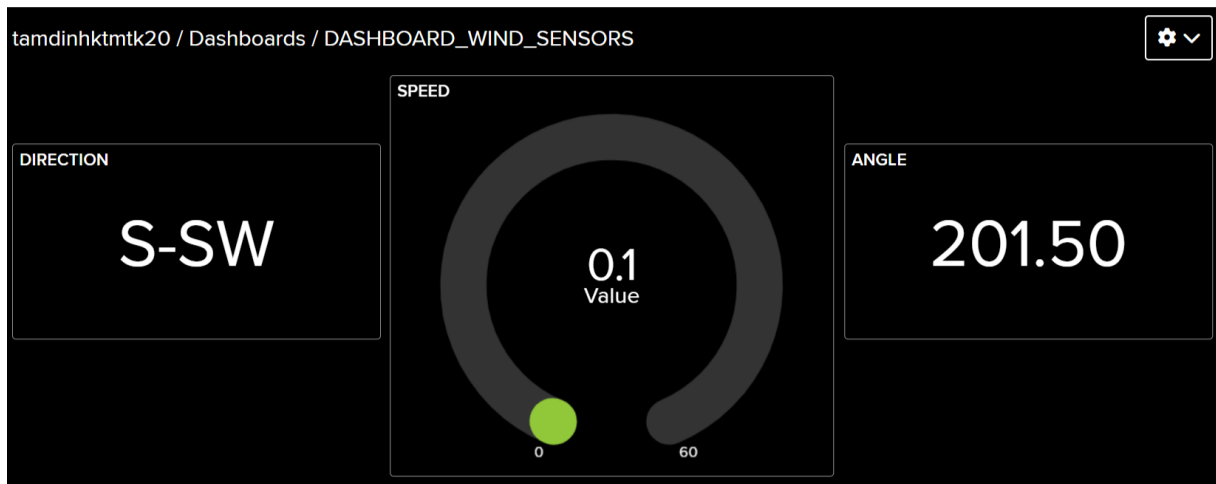


Figure 9: A dashboard

2.2.2.5 A problem with Adafruit IO server

In Adafruit IO, "data limiting" refers to a feature that allows users to restrict the amount of data stored or transferred within a specified time frame. This feature helps prevent excessive data usage, especially in scenarios where users have constraints on data storage or network bandwidth.

By setting data limits in Adafruit IO, users can control the volume of data they send or receive within a given period. This can be useful in IoT (Internet of Things) applications or other projects where data usage needs to be managed to avoid surpassing certain thresholds, such as API rate limits or data storage capacities.

Users can define these limits based on factors like the number of requests, the frequency of data updates, or the amount of data transferred. This allows them to regulate and optimize their usage of Adafruit IO services according to their specific needs and constraints.

In the scope of this project, we decide to use a "free" version of Adafruit IO. A problem definitely occurs when we try to send too much data for a period of time. Data cannot be updated to real-time as our expectations. Hence, it can be difficult to monitor the current environmental condition just by observing the dashboard. That's why in this part, we come up with a very basic solution using Python to store all data the program can collect through time.

Firstly, we create a Python project. Inside the project's folder, we create a .txt file named "arduino_data.txt" as below:

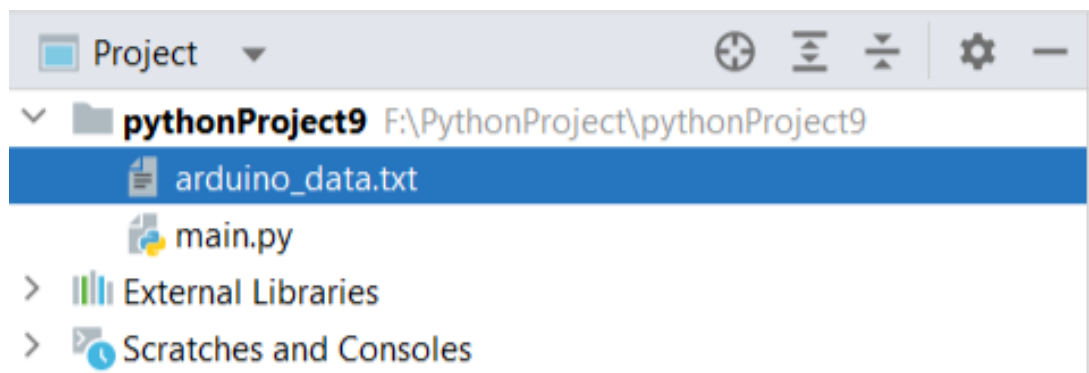
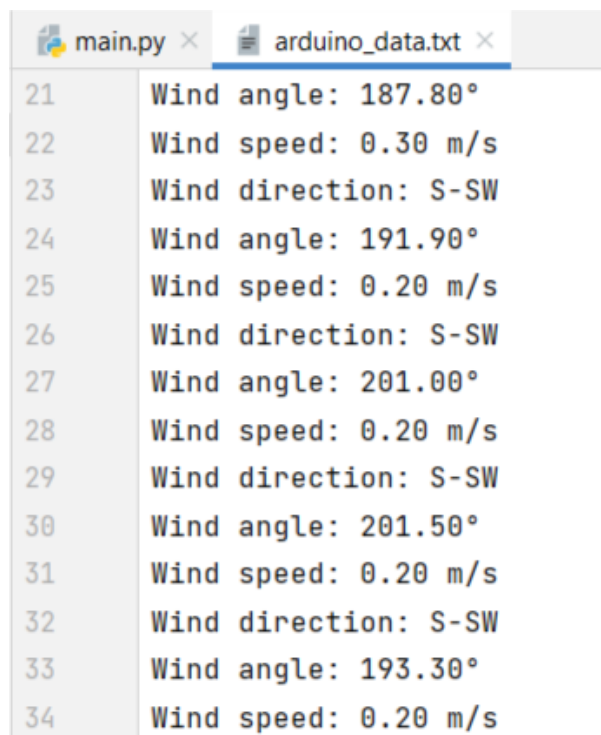


Figure 10: Create a .txt file

Then, we use this code to get all data from the serial terminal:

```
1  import time
2  import serial
3  # Change 'COMX' to your Arduino's port and '9600' to the baud rate
4  ser = serial.Serial('COM5', 9600)
5  # Open or create a file named arduino_data.txt for appending data
6  file = open("arduino_data.txt", "a")
7
8  while True:
9      if ser.in_waiting > 0:
10         # Read data from the serial port
11         data = ser.readline().decode().strip()
12         # Write data to the file
13         file.write(data + "\n")
14         time.sleep(0.5)
```

Finally, we upload the code from Arduino IDE to the Arduino. Then, we need to close Arduino IDE and run the Python code. Please note that we have to close any other serial terminal when we run this Python code. Otherwise, the COM port access right will be denied. Here's the result:



Line	Wind angle	Wind speed	Wind direction
21	187.80°		
22		0.30 m/s	
23			S-SW
24	191.90°		
25		0.20 m/s	
26			S-SW
27	201.00°		
28		0.20 m/s	
29			S-SW
30	201.50°		
31		0.20 m/s	
32			S-SW
33	193.30°		
34		0.20 m/s	

Figure 11: Results on the .txt log file

2.2.3 System Integration and Testing

The link to the video demonstrated our system is provided [here](#).

2.3 Source code

All the files are saved in this [Github link](#). Please feel free to download and enjoy it.

2.4 Incomplete Tasks and Future Work

2.4.1 Incomplete Tasks

1. Alert System Implementation

The development of an alert system was envisioned as a crucial component of this IoT-based climate monitoring system. Regrettably, due to time constraints, the integration of this feature remains pending. The alert system was intended to notify designated stakeholders or users via email, SMS, or a mobile application in the event of extreme climate conditions or anomalies detected by the RS485 sensors. It is recognized that the implementation of such an alert mechanism is pivotal for timely responses to potential environmental concerns and ensuring proactive measures.

2. Testing and Validation Timeframe

The testing phase is an integral aspect of ensuring the reliability, accuracy, and robustness of any IoT-based system. However, due to time limitations, the planned duration for rigorous testing and validation of the RS485-based climate monitoring system was not fully executed. Comprehensive testing is imperative to validate the system's functionality across diverse environmental scenarios and to verify its responsiveness under various operational conditions. An extended testing phase would have facilitated the identification and rectification of potential shortcomings or inconsistencies within the system.

3. Modern Database Mechanism Integration

In today's dynamic technological landscape, leveraging a more modern database mechanism could significantly enhance data storage, retrieval, and scalability aspects of the IoT-based system. Although initial considerations were made for this enhancement, the implementation and migration to a more advanced database system remain pending. A contemporary database structure would have facilitated quicker data processing, improved query performance, and potentially streamlined data management practices.

2.4.2 Future Directions

1. Alert System Integration

Moving forward, dedicating efforts to implement and integrate an alert system remains a priority. This will ensure timely notifications and proactive measures in response to detected anomalies or critical climate conditions.

2. Extended Testing Phase

Allocating a sufficient timeframe for comprehensive testing and validation is crucial. This will involve simulating various climate scenarios and operational conditions to verify the system's reliability and accuracy.

3. Modern Database Integration

Exploring and adopting a more advanced database mechanism, such as NoSQL or cloud-based solutions, should be a focus in subsequent phases. This upgrade would enable improved scalability, faster data retrieval, and enhanced overall system performance.

2.5 Bibliography

References

- [1] Edwin Chen, *WSC1-L-Dragino LoRaWAN Weather Station User Manual*, <http://wiki.dragino.com/xwiki/bin/view/Main/User%20Manual%20for%20LoRaWAN%20End%20Nodes/Dragino%20LoRaWAN%20Weather%20Station%20User%20Manual/>, last updated on 01/11/2023
- [2] How to Electronics, *Connecting RS485 Soil Moisture & Temperature Sensor to Arduino*, <https://how2electronics.com/connecting-rs485-soil-moisture-temperature-sensor-to-arduino/>, 18/09/2023
- [3] *How to Use Adafruit IO with an ESP8266 and the Arduino IDE*, <https://www.digikey.com/en/maker/tutorials/2019/how-to-use-adafruit-io-with-an-esp8266-and-the-arduino-ide>, 01/09/2019
- [4] Random Nerd Tutorials, *Guide for I2C OLED Display with Arduino*, <https://randomnerdtutorials.com/guide-for-oled-display-with-arduino/>