

Internet of Things Implemented with Mruby

Hattapat Silangern and Kazuaki Tanaka(✉)

Kyushu Institute of Technology Iizuka, Fukuoka 820-8205, Japan
silangern.hattapat647@mail.kyutech.jp kazuaki@ics.kyutech.ac.jp

Abstract. The objective of this paper is to develop an Internet of Things (IoT) system that utilizes fusion sensors implemented with Mruby, a lightweight implementation of the Ruby programming language specifically designed for microcontrollers. The Mruby has been adapted to function on compact devices such as the Internet of Things (IoT) and other embedded systems. In addition, to improve communication efficiency in the IoT system. Thus, LoRa is well-suited for integration with the IoT system. The LoRa module, a low-power transmission technology, facilitates communication between embedded systems and other systems. LoRa is a technology that allows communication over long distances while consuming little energy, which is suitable for integration with IoT systems. As a result, this research has focused on embedded system development, which involves creating software and hardware components for IoT and embedded systems through design, coding, and testing processes. This involves a combination of software and hardware engineering, with an emphasis on creating systems that are efficient and reliable and able to satisfy specific performance standards. The microcontroller has successfully created a functional environment for running Mruby. This involves running the Mruby code in the microcontroller environment. The experiment demonstrates the procedure for setting up a Mruby environment and directory structure, along with the creation of a test program that can access real-time data.

Keywords: Embedded system · Mruby · Ruby · Long Range communication

1 Introduction

Nowadays, electronic devices are rapidly advancing and widely utilized to support customers. The demands of customers today extend beyond mere functionality. They seek seamless integration, emotional interfaces, and innovative features that enhance their overall experience. Thus, integrating electronic devices has become an indispensable part of daily consumer life. For example, a dishwasher machine simplifies their lives. Instead of washing dishes by hand, which can be time-consuming, this electronic device efficiently cleans and dries our plates and utensils. Thus, washing with a dishwasher machine not only saves time but also ensures a thorough cleaning, leaving consumers with more time to do things.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14819, pp. 165–181, 2024.
https://doi.org/10.1007/978-3-031-65282-0_11

On the other hand, when consumers have a desire to use functions in electronic devices, a variety of devices. Therefore, developers emphasize developing systems that meet the requirements of various consumers. Developing each system will be quite expensive, depending on the system. Several years ago, the Information Technology Promotion Agency (IPA) in Japan released reports indicating that the production costs of systems and electronic devices were notably high. Because the production costs depend on the software and hardware architecture and development, the software development cost is a large proportion of the system. In fact, there is a lot of source code for the software available on the internet right now, which makes it simple to implement. However, the source code remains complex and hard to fully understand within a short period of time. Thus, designing smaller and easier software for the system can significantly reduce production costs and conditions for users. Software development is an attractive area for developing embedded systems. To reduce production costs.

Therefore, this research focuses on Mruby, a lightweight implementation of Ruby, which is an object-oriented programming language and scripting language for microcontrollers employed in IoT and embedded systems. This study investigates the potential applications of Mruby within embedded systems, examining its advantages and challenges, particularly in terms of the design of the software and hardware. This research is poised to contribute to more cost-effective and efficient designs, driving advancements in embedded system technologies.

2 Literature Review

This section aims to closely review the work and related information in order to gain a more comprehensive understanding of each important part of this research. Utilizing the relationship between the Internet of Things and embedded systems. Furthermore, this study also emphasizes the selection of suitable microcontrollers and sensors. While analyzing the utilization of long-range and short-range communication to enhance the communication capacity among individual systems to align with the particular system's conditions. The most important part of this section is to focus on developing and designing the programming language Mruby to be suitable for the IoT embedded system that has been planned and designed to meet the goals and objectives that have been set.

2.1 Internet of Things

The Internet of Things (IoT) is an integrated network of devices connected to the Internet, or the linking of smart devices across the Internet to the cloud. To be able to share information between devices, a general IoT system involves collecting and exchanging information in real time and consists of three parts: smart devices, IoT applications, and graphical user interfaces. This research will focus on designing and developing intelligent devices that implement with the programming language Mruby, which will be divided into two parts: hardware and software development.

Embedded System. An embedded system is a system that consists of a computer that has a specific function, namely, a computational system. using microcontrollers that are specifically designed to have pre-defined functionality with specific requirements, for example, smart devices that have small computers embedded in them and control them.

- **Systems design and Development**. This part describes the design and development of the system. From the information provided, it can be concluded that the microcontroller is a crucial component in the design and development of an embedded system. A microcontroller functions akin to a compact computer, brimming with integrated circuits (ICs) that comprise diverse components like memory and peripherals. Therefore, the system design in each system will depend on the purpose of each task that has been designed and programmed into the controller chip to perform a specific task. For example, reading the temperature value from temperature sensor by changing the input of the sensor into output using a microcontroller to read and process the data.

Embedded Software. This part emphasizes the significance of software in facilitating communication between the program and the microcontroller. The design of software specifically caters to the specific microcontroller it operates on. Typically, we employ embedded software to control the specific and programmed operations of a microcontroller.

- **Programing languages**. Programming languages are typically well-suited for writing programs that can communicate with microcontrollers. There are a wide range of options available nowadays. For example, the C programming language is the most widely used programming language for embedded devices. Built to be fast and efficient, C is a low-level language. Therefore, it is a powerful language that allows developers to write code that communicates with the hardware. However, the choice of programming language in software development is dependent on the developer's understanding of programming complexities and the specific requirements of the software. Consequently, the decision about which programming languages to use involves various elements, allowing developers to determine the suitability of each language.

2.2 Microcontroller

This section describes the microcontroller, which is a key part of IoT embedded systems. Microcontroller is a small and low-cost microcomputer. that is designed to control specific tasks within an electronic system. Microcontroller provide a combination of functionality through a central processing unit (CPU), memory (RAM, ROM, and EPROM), input/output interface ports, and other peripherals.

ESP32. Over the years, there have been many microcontrollers designed for use in IoT embedded systems, such as the ESP32, ESP8266, Arduino, Nano,

and the Raspberry Pi Pico. From experimentation and comparison with other microcontrollers. Therefore, we can conclude that the ESP32 is suitable and capable of achieving the objectives of this research. The ESP32 has a combination of performance built-in Wi-Fi and Bluetooth connectivity and is easily integrated with local networks and the Internet. It also comes with a dual-core processor running at up to 240 MHz. which provides more processing power than a single-core processor running at 16 MHz. In addition, ESP32 has a large development community, which has a tutorial library and development tools that can support developers comprehensively. Therefore, making it easier for developers to start ESP32-based projects.

– **Espressif IoT Development Framework (ESP-IDF)**. This part describes the Espressif IoT Development Framework (ESP-IDF) as the official development framework for Espressif Systems' ESP32 and ESP32-S series microcontrollers. This development framework provides a comprehensive set of tools, libraries, and documentation for developers who want to develop firmware and applications for microcontrollers. Therefore, the Espressif development framework is for developing Internet of Things (IoT) applications with Wi-Fi, Bluetooth, power management. and other features of the system. Which is a good help for developers. The framework provides basic hardware and software resources to help application developers realize their ideas using the ESP32 series hardware[1] .

2.3 Software Integration with Mruby

This part focuses on the efficient development of software for integration with microcontrollers. C is currently the most commonly used programming language for communicating with microcontrollers. This is due to the fact that C serves as a general-purpose language for a diverse range of applications, encompassing systems programming, application development, and low-level system programming. On the other hand, C syntax is complicated. Therefore, designing applications for each project requires more time and resources.

At this point, it is necessary to focus on the development of programming languages that are both user-friendly and easy to use. In order to reduce the loss of resources in the design process for each project, the Mruby is the right choice. This is because it has a non-absorbent scripting style and dynamic behavior. It allows developers to create and run Ruby scripts.

Ruby Programming Language. This part describes the Ruby programming language. Ruby is a dynamic, object-oriented programming language known for its simplicity and productivity. Yukihiro Matsumoto publicly released Ruby in 1995. Ruby's main features include dynamic typing, object-oriented principles, clean syntax, meta-programming capabilities, automatic memory management through garbage collection, and a standard library. Therefore, Ruby is widely used for web development, scripting, automation, prototyping, etc., which makes Ruby a popular choice among developers.

On the other hand, Ruby's need for automatic memory management can result in slower execution times and higher memory usage. This is especially true in resource-intensive applications that consume a lot of resources in their operation. However, many Ruby applications are web applications, so there are no problems with consuming resources on the server machine[2]. Ruby retains its powerful meta-programming capabilities and extensive ecosystem. This appeals to developers, who value developer productivity over raw performance.

Mruby. This part describes Mruby. Mruby is a lightweight implementation of the Ruby programming language, compliant with ISO 30170, released under the MIT license[3]. Mruby has different uses than others. Mruby outperforms Ruby due to its compactness and optimization for use cases with limited memory and processing power. Mruby is generally designed specifically for embedded systems, IoT devices, and other scenarios. that focus on efficiency and suitability in using resources in a limited environment.

However, Mruby also aims to achieve this goal by providing a smaller standard library and offering features for customizing the language runtime. Although reduced in size, Mruby retains many of Ruby's core features, such as dynamic typing. Object-oriented programming has meta-programming capabilities.

Mruby Compiler. This part focuses on the functionality of the Mruby compiler, which is an important part of developing a programming language suitable for use with ESP32. The Mruby compiler, or "mrbc," is a tool that can compile Ruby programs into byte code. The byte-code features are compact and efficient, independent of the device. Thus, compiling Ruby programs using the Mruby compiler tool. This process will produce byte code. The next procedure byte code is executed by the Mruby Virtual Machine (VM). The steps for compiling and executing are shown in Fig. 1. MrubyVM is responsible for executing byte code, managing memory, and arranging the environment for the target device. In general, byte code that is compiled does not depend on the target device architecture because MrubyVM is designed to be able to run byte code on various hardware architectures and operating systems. On the other hand, MrubyVM also needs to be optimized for the device because MrubyVM needs to specify a target device. So the bytecode can be used on another device, but MrubyVM must be optimized for the device.

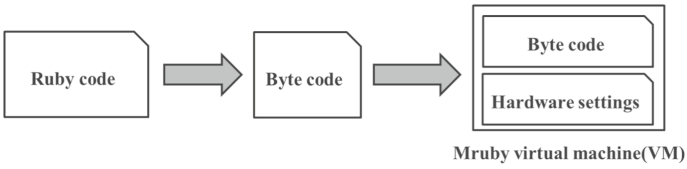

Fig. 1. Mruby compiler

Mrbgems. This part focuses on the capabilities of mrbgems. Mrubgems is a Mruby library that integrates C and Ruby. Which is important in expanding Mruby's functionality and increasing its ability to develop multiple applications. Developers have developed mrbgems with easy-to-use steps and a clear user manual available for download from Github. Therefore, mrbgems has an important function that helps in developing applications. By focusing on the integration of C and Ruby, it extends Mruby's ability to more easily access control of embedded systems, IoT devices, or various hardware architectures.

2.4 Long Range Communication

This section focuses on the use of long-range communication technology to increase the communication and data transmission limits between devices. LoRa is a technology that provides long-range communication and low power consumption. Which is suitable for deployment with IoT systems that require connectivity over extended distances. As mentioned in the Microcontroller section, the ESP32 has a combination of built-in Wi-Fi and Bluetooth connectivity, providing high-speed communication over short to medium ranges that is easy to integrate into local networks and the Internet. But ESP32 still has limitations in terms of communication distance [4]. Therefore, long-range communication technology is a suitable option to increase the communication limit. However, each technology has different strengths and suitability. As shown in the Fig. 2. Therefore, the selection of each technology depends on its suitability for each task, which can be determined by many factors such as range, power consumption, data rates, and usage requirements.

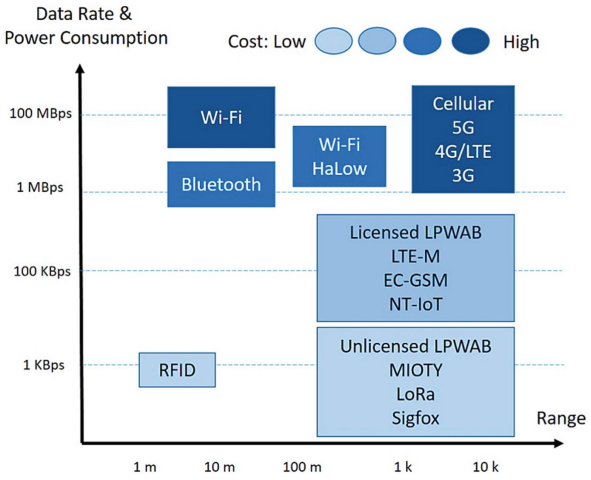

Fig. 2. Comparison between LoRa and other wireless technologies [4]

RYLR896 Lora Module. After studying about long-range communication technology, the RYLR896 module was found to be an appropriate choice. The RYLR896 transceiver module features the LoRa long-range modem, which provides ultra-long-range spread spectrum communication and high interference immunity while minimizing current consumption. The RYLR896 is certified by the NCC and FCC [5].

The RYLR896 module has many functionalities that allow users to design LoRa wireless transceiver applications and to design different network architectures such as "Point to Point", "Point to Multipoint" or "Point to Multipoint". The flexibility of network design is shown in the Fig. 3, the figure shows the structure of network design. Each module can communicate with each other, but only modules that are set to the same network ID can communicate with each other. However, each module's address section prevents communication between data receivers belonging to different groups.

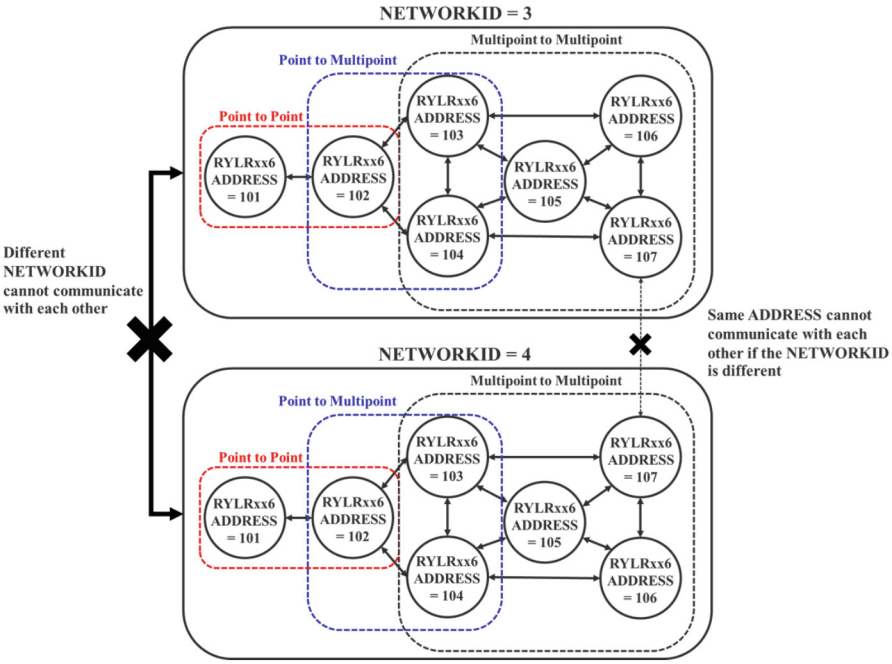

Fig. 3. RYLR896_RYLR406 network structure

Therefore, the LoRa RYLR896 module is an ideal choice for long-range wireless communications applications in IoT applications, providing a balanced range. The module exhibits energy efficiency and facilitates easy configuration.

3 Methodology

This section describes the integration of low-cost and low-power System on a Chip (SoC) microcontrollers called ESP32 which use the object-oriented and scripting programming language Mruby, which was designed specifically for the implementation of an Espresso IoT Development Framework (ESP-IDF)-based microcontroller. Additionally, this section describes the system overview, focusing on the software and hardware design that integrates with Mruby by installing various IoT devices used in the IoT embedded system to design the system to work efficiently. Effective according to requirements.

Furthermore, this section provides a comprehensive overview of the system, with a specific focus on the software and hardware design. Integration with Mruby is achieved by installing various Internet of Things (IoT) devices in the IoT embedded system. The aim is to design a system that functions efficiently and meets the specified requirements.

3.1 Implementing the ESP32 Microcontroller with Mruby

The microcontroller known as ESP32 is employed in the research because of its widespread application and relevance on the Internet of Things and embedded systems. The ESP32 empowers programming developers with a flexible platform by providing a dual-core processor, wireless connectivity, and a variety of embedded application-compatible peripherals. As a result, for developers to integrate the Mruby programming language with ESP32, the ESP-IDF framework, which facilitates application development, must be utilized. The ESP-IDF is the designated development framework for the ESP32. It provides a comprehensive environment for the development of applications based on the ESP32 and includes all the necessary tools and resources.

Another thing to consider is Mruby integration. The integration of Mruby means implementation of a lightweight Ruby programming language into the ESP32 environment. This requires the adaptation of Mruby to the ESP32 architecture in order to optimize Mruby's performance and ensure its compatibility with embedded systems with limited resources. Therefore, the ESP32 environment requires the following three components in order for Mruby to function:

The first component is a virtual machine of Mruby. The runtime environment is the location where the Mruby code is carried out. Mruby scripts are interpreted and executed by the Virtual Machine (VM), which is an embeddable and lightweight framework. By providing an abstraction layer, the virtual machine enables Mruby code to execute on the ESP-IDF framework. It is simple to incorporate the Mruby VM into C and C++ applications. For situations in which scripting capabilities are desired as part of a larger software system, this feature renders it suitable. Due to its compact memory footprint, Mruby is well-suited for environments with limited resources.

The second component can be referred to as 'Collection of Garbage in Increments'. It is a method for modifying unused memory in a program and managing

memory without resulting in significant interruptions in program execution. An important aspect of programming languages

with automatic memory management is garbage collection. To minimize the development of memory leaks and prevent the occurrence of Mruby, Incremental GC possesses the capability to detect and recover unused memory. Furthermore, it effectively manages memory in a manner that is resource-constrained, responsive, and efficient. It enhances the performance and dependability of Mruby applications as a whole, by implementing a system that consistently detects and releases unused memory while the program is running.

The third component is Interface in C library. The software controls the hardware in an embedded system or ESP32 framework using C; therefore, Mruby implementation for ESP32 or embedded systems is significant. A function of Mruby provides support for the C library. The C interface in Mruby serves as an interface between Mruby scripts and C code, allowing for the exchange of data and facilitating integration between the two environments. This includes the management of complex data structures, the passing of arguments, and the return of values. This feature enables Mruby scripts to execute C functions.

As a consequence, upon assembling every component of the Mruby system, which is built upon the ESP32 framework, it becomes apparent that programs can be compiled and executed on the ESP32. One possible application of Ruby language syntax is to instruct the program to display messages according to the user-specified criteria. Concurrently, the written message is displayed on the screen. The outcome, nevertheless, is the compilation and execution of Ruby code on ESP32, which prevents access to ESP32 control as it solely executes and compiles Ruby code. This is the issue that must be resolved in the next phase of Mruby's design in order to access and control the operation of ESP32.

The ability to access the ESP32 library. To get access to implementing the ESP32 library, the most important task in resolving the previously mentioned issue concerning Mruby's design is to establish access to and regulate the functioning of ESP32. Alternatively, one might come up with Mrbgems optimized for ESP32. Mrbgems is a library manager that simplifies the procedure of integrating C and Ruby extensions with Mruby, making it easier to follow standardized and straightforward procedures. Mruby facilitates the interaction and control of ESP32 hardware components and peripherals, including GPIO pins, sensors, actuators, communication interfaces (such as UART, SPI, I2C), and wireless capabilities. Wired technologies, such as Wi-Fi and Bluetooth. As a result, in order to allow Mruby access the functioning of the ESP32, the developer must initially create the architecture of Mrbgems. For example, in order to enable Mruby access the GPIO port of the ESP32, the initial course of action involves creating the Mrbgems directory, an essential structure through which Mruby is able to access the C library. Subsequently, the C library that is approved by the C Interface and lies within the responsibility of the Mruby C function is designed.

3.2 Overview of Systems

This section describes an overview of the system designed by combining hardware and software that have already been designed and developed together. In building IoT embedded systems, it is especially important to take into consideration the system's objectives, efficiency, and reliability. This section focuses on the design and development of IoT embedded systems that work efficiently and reliably.

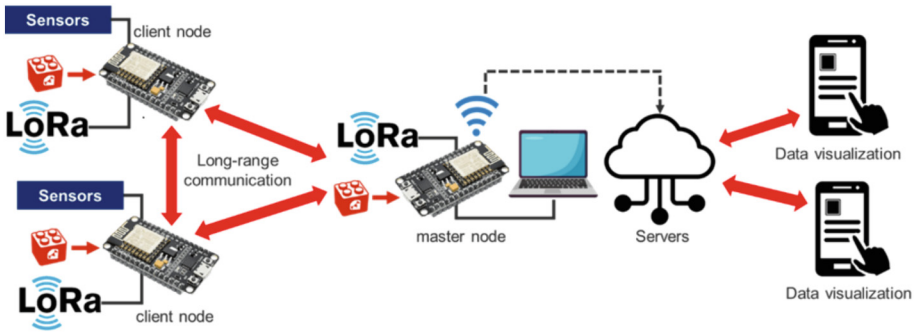

Fig. 4. Overview of the ESP32-designed system's structure

The Fig. 4 provides an overview of the ESP32-designed system's structure, which serves as the prototype structure for the IoT system. This system operates through two separate parts: client nodes and master node, each assigned a specific function. Client nodes have the responsibility of monitoring, collecting data, and controlling electronic devices. The number of client nodes can be more than one node, based on the particular function assigned to each node. For example, the first node is tasked with collecting data from sensors, while the second node is responsible for controlling the other devices in the system. Master node is responsible for remotely combining data from client nodes. Master node is the main node in the system, with the ability to send commands to client nodes. Furthermore, Master node has the capability to establish a connection with Wi-Fi and transmit data to servers in order to display the information collected from client nodes. Additionally, this IoT system possesses the capability to establish communication between client nodes and master node by the utilization of long-range communication technologies, enabling seamless operation within each node.

Consequently, in order to achieve the objective of designing and developing IoT embedded systems, implement the Mruby programming language. The Smart Farm system is a great option due to its excellent compatibility with the designed structure of the system.

3.3 Smart Farm System Design

Smart Farm or Smart Agriculture, refers to the application of advanced technologies to efficiently manage agricultural activities. The entity responsible for collecting, analyzing, and processing data until it achieves a state of automation. Currently, there are many technologies that can be applied to Smart Farm, such as: Greenhouse Automation is a small indoor farm. This technology is used to actively monitor and control the conditions in which plants develop, providing real-time monitoring and control. that can analyze and evaluate many physical characteristics and get access to phenomena of nature[6]. Moreover, artificial intelligence (AI) technology is used to help with data sharing and the management of systems. This technology has the ability to communicate with the Internet of Things, which is accessible in Smart Farm environments, and is efficient in managing and evaluating huge quantities of data [7].

Therefore,the objective of the Smart Farm system is to improve agricultural productivity by increasing crop yields, improving crop quality, reducing unnecessary operational expenses, and increasing profits. In general, the Smart Farm application has multiple functions, including monitoring soil moisture and temperature, controlling water systems, and controlling electrical equipment.

At this point, the objective is to design and develop the Smart Farm system in the form of Greenhouse Automation to operate with optimal efficiency. By expanding the communication limits among nodes, utilizing long-range communication technology, and displaying various data, in order to enhance the reliability of the system. The Fig. 5 provides an overview of the Smart Farm system, which is divided into two parts: the client node and the master node.

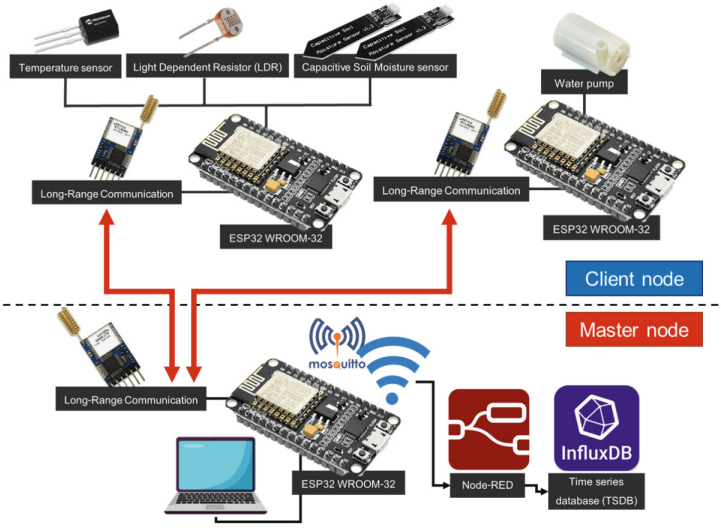

Fig. 5. Overview of the the Smart Farm system

The first part, the client node, has the responsibility of monitoring, collecting data, and controlling the water flow. The client node divides into two sub-nodes. The first node is responsible for measuring the soil moisture, measuring the temperature, and measuring the level of light. The second node is responsible for controlling the water system. Each node has the capability to communicate and send data with other client nodes or master node using long-range communication technology.

The second part, the master node, has the main task of facilitating the transmission of data between client nodes using long-range communication technology. that facilitates long-range communication. Furthermore, master nodes have the capability to establish an internet connection in order to utilize the MQTT protocol for transmitting received data to the time series database (TSDB) via Node-RED. Node-RED is a data bridge that facilitates the transfer of data from MQTT to TSDB. In order to be able to efficiently display information on environmental factors that affect plant growth.

4 Experiments and Results

This section describes an overview of the experiment, which emphasizes the objective of developing a system that can operate with high efficiency. The experiment is divided into two parts: the software and the hardware.

The first part is to carry out a software experiment. The purpose of this part is to evaluate the functionality of the software developed for the Smart Farm system. In general, the software of the Smart Farm system combines with sensors for collecting environmental data. As a result, the software must possess functionalities to facilitate effortless communication with the LoRa module and various sensors such as GPIO, ADC, PWM, UART, Wi-Fi, and MQTT. Each function is designed to be interoperable with Mruby, functioning as a library or Mrbgems.

Listing 1.1 provides a demonstration of utilizing a program in the client node. This node is responsible for monitoring and collecting data. The program utilizes the ADC and LoRa libraries. As shown in Fig. 5, the left client node is connected with four sensors: a temperature sensor, a Light Dependent Resistor (LDR), and two capacitive soil moisture sensors. All of these sensors are connected to the ADC port of the ESP32. Thus, the program operates by utilizing a function to read the analog signal from each sensor and convert it into a digital signal. The signal that is read is a voltage level from 0-3.3 V. Afterwards, the program will send voltage levels to the master node using the LoRa function. The results of the program are shown in Fig. 6.

```
include ESP32::STANDARD
include LORA
lora = LoRa.new NUM_1,25,26,115200
lora.set_node 1,10
lora.set_cmode MODE_1
```

```
while true
    Tv = ADC.read_temp 6
    Tv = Tv.round(2)
    puts "Temp_Volt:#{Tv}V"
    Sv = ADC.sun_check 7
    Sv = Sv.round(2)
    puts "Sun_Volt:#{Sv}V"
    Sr = ADC.soil_R 5
    Sr = Sr.round(2)
    puts "soil_R_Volt:#{Sr}V"
    Sl = ADC.soil_L 4
    Sl = Sl.round(2)
    puts "soil_L_Volt:#{Sl}V"

    Msg = "Sun:"+Sv.to_s+"|"+"Temp:"+Tv.to_s+"|"+"soilL
       :"+Sl.to_s+"|"+"soilR:"+Sr.to_s
    lora.send 3,Msg.to_s
    sleep 15
end
```

Listing 1.1. Demonstration of using the program in the client node

```
######################################################################
Temp_Volt : 0.81V
Sun_Volt : 1.65V
soil_R_Volt : 2.03V
soil_L_Volt : 1.86V
LoRa send : AT+SEND=3,40,Sun:1.65|Temp:0.81|soilL:1.86|soilR:2.03
######################################################################
Temp_Volt : 0.8V
Sun_Volt : 1.64V
soil_R_Volt : 2.02V
soil_L_Volt : 1.85V
LoRa send : AT+SEND=3,39,Sun:1.64|Temp:0.8|soilL:1.85|soilR:2.02
######################################################################
Temp_Volt : 0.81V
Sun_Volt : 1.65V
soil_R_Volt : 2.01V
soil_L_Volt : 1.85V
LoRa send : AT+SEND=3,40,Sun:1.65|Temp:0.81|soilL:1.85|soilR:2.01
######################################################################
```

Fig. 6. the result of the client node program

The second part involves carrying out a hardware experiment. The purpose of this section is to evaluate the system's ability to function efficiently within the Smart Farm system. This section emphasizes the incorporation of embedded systems that are capable of accurately responding to and collecting data from each device, as will be further described in the Application section.

4.1 Application

This section describes the details of the test procedure performed on the system. Therefore, from Fig. 5 presents an overview of the Smart Farm system, dividing it into two main parts: the client node and the master node.

The task of client nodes is to monitor, collect information about the conditions that affect the growth of plants, and control the water flow. Figure 7 provides the installation of two client nodes for the task of collecting data from the real-world environment. Through performing experiments to collect data from small garden plants. The operation of the first node is used to collect data from three types of sensors. The first sensor, which is the temperature sensor, is tasked with measuring the temperature and recording any variations in temperature. The second sensor, known as the Light Dependent Resistor (LDR), is responsible for measuring the intensity of light in real time. To record the illumination level. The third sensor, the capacitive soil moisture sensor, is utilized for measuring the moisture level in the soil. To determine the suitability of the soil moisture for plant growth. Furthermore, the second node has the additional responsibility of controlling the water system by adjusting the water flow according to the soil moisture measured by the soil moisture sensor. In order for this node to control the flow, this node needs to wait for commands from the master node.

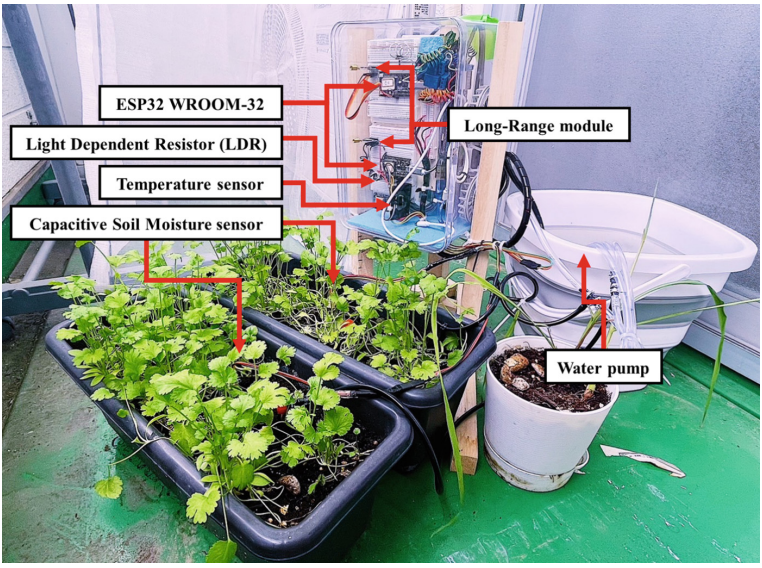

Fig. 7. The installation of client nodes to collect data from small garden plants.

Finally, the client node prepares the data received from the three sensors, and all the data is organized in string format. Subsequently, the client node will

transmit data regarding temperature, light intensity, and soil moisture through the LoRa module. The LoRa module will transmit data to another LoRa module installed on the master node. Figure 8 provides the transmission of data between the client node and master node using the LoRa module.

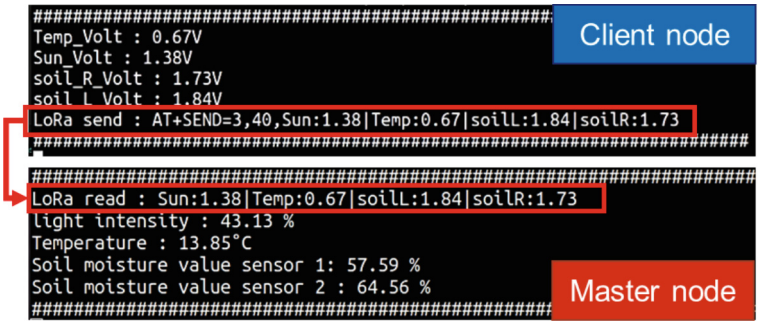

Fig. 8. the transmission of data between nodes using the LoRa module

According to Fig. 8, when the master node receives data from the client node, the program will separate the data into three categories: temperature, light intensity, and soil moisture. Furthermore, the data had been organized in order to calculate light intensity and soil moisture as percentages, which would simplify the process of analyzing the data. Subsequently, the program will make the necessary preparations to transmit the data to the next step, which is to visualize the data on InfluxDB. InfluxDB is a time-series database designed for the purpose of storing and analyzing time series data. It is especially suitable for applications that involve monitoring and recording sensor data in Internet of Things (IoT) systems.

In order to transfer data to InfluxDB, Therefore, utilize the MQTT protocol function to publish data to the MQTT Broker and subscribe a topic by Node-RED. Node-RED is a web-based visual programming tool that enables users to create flow designs and integrate them with various systems. At this point, the main function of Node-RED is to interact with InfluxDB by sending data from Node-RED to InfluxDB for visualization of the data. The data visualization results are provided in Fig. 9. In this figure, the InfluxDB will display a time series of data showing measurements of soil moisture, temperature, and light intensity. These measurements are collected using a temperature sensor, a Light Dependent Resistor (LDR), and two capacitive soil moisture sensors. The data from each sensor will be shown in the form of a line graph and a gauge view. The line graphs display time-series changes in measurement data from several sensors. Users can easily realize this by observing changes in the graph. Moreover, the gauge view represents only the latest data for a time series. The gauge view provides a visual representation of the gauge at scale and allows for customization, making it easy to understand current data.

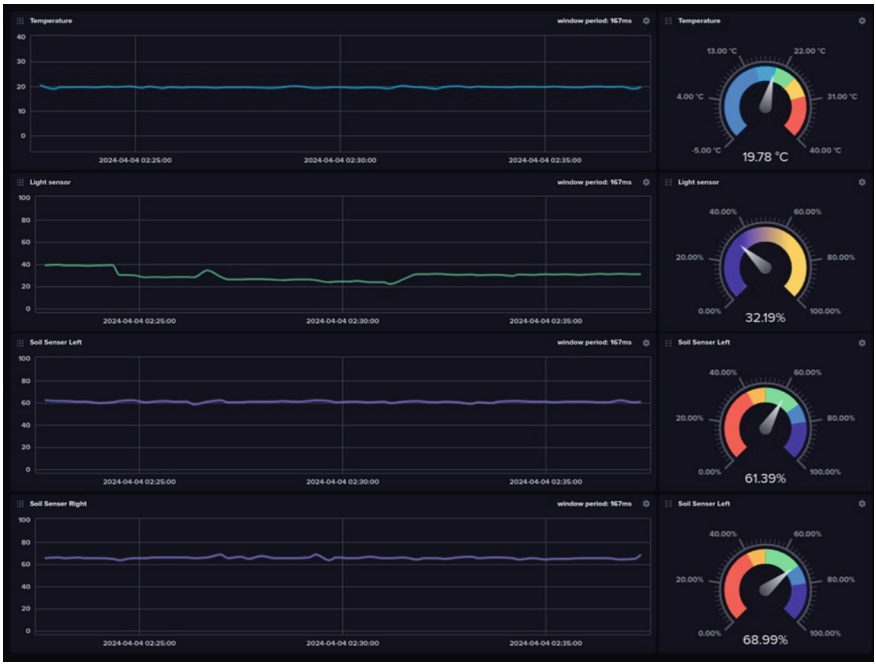

Fig. 9. Visualization of data with InfluxDB

5 Conclusion

This paper presents the design and development of an embedded system with long-range communication technology designed for the Internet of Things (IoT), with a focus on using it in Smart Farm systems. The operation of the Smart Farm system is able to monitoring, collecting, and visualizing environmental data that affects plant growth.

Additionally, the system's software has been implemented with Mruby because of its compactness and optimization for use cases with limited memory and processing power. Mruby's design specifically caters to embedded systems. However, Mruby maintains the features and syntax of Ruby, facilitating users ability to write simple scripts using Ruby's syntax. Thus, using Mruby to program embedded systems is an alternative choice for users.

References

1. Get Started. ESP-IDF Programming Guide. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html
2. Tanaka, K., Higashi, H.: Mruby-rapid IoT software development language. In: 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA), Siem Reap, Cambodia, pp. 415–418 (2017). https://doi.org/10.1109/ICIEA.2017.8282880

3. About mruby. Mruby. https://mruby.org/about/
4. Al-Shareeda, M., Alsadhan, A., Qasim, H., Manickam, S.: Long range technology for internet of things: review, challenges, and future directions. Bull. Electr. Eng. Inf. **12**(6), 3758-3767 (2023). https://doi.org/10.11591/eei.v12i6.5214
5. REYAX LoRa® AT COMMAND GUIDE., REYAX TECHNOLOGY CO., LTD (2022)
6. Astillo, P.V., Kim, J., Sharma, V., You, I.: SGF-MD: behavior rule specification-based distributed misbehavior detection of embedded iot devices in a closed-loop smart greenhouse farming system. IEEE Access **8**, 196235–196252 (2020). https://doi.org/10.1109/ACCESS.2020.3034096 keywords: Monitoring;Kalman filters;Sensors;Intrusion detection;Green products;Smart greenhouse farming (SGF);Internet-of-Things;cyber-agroterrorism;misbehavior detection;specification-based approach
7. AlZubi, A.A., Galyna, K.: Artificial intelligence and internet of things for sustainable farming and smart agriculture. IEEE Access **11**, 78686–78692 (2023). https://doi.org/10.1109/ACCESS.2023.3298215 keywords: Smart agriculture;Crops;Artificial intelligence;Internet of Things;Farming;Monitoring;Soil;Internet of Things;Sustainable development;Smart agriculture;Internet of Things (IoT);artificial intelligence (AI);smart sustainable agriculture (SSA);smart farming