INDIANA UNIVERSITY
SOUTH BEND

Pi-Node Bridge

Samiksha BC

Digital Design CSCI-B541

Liqiang Zhang

Abstract

The "Pi-Node Bridge" project is an innovative Internet of Things (IoT) communication system addressing real-time obstacle detection and message relay challenges. This project acts as a bridge between the Raspberry Pi and NodeMCU (ESP8266), aiming to enable obstacle detection through ultrasonic sensors on the NodeMCU while facilitating seamless communication with the Raspberry Pi via MQTT (Message Queuing Telemetry Transport). Leveraging the NodeMCU's capabilities, the system detects obstacles using ultrasonic sensors, processes the data, and, upon identifying an obstacle, initiates an MQTT message. This message is then transmitted to the Raspberry Pi, creating an efficient communication link between the two platforms. The project's robustness stems from the implementation of MQTT, a lightweight protocol, ensuring low-latency, reliable, and bidirectional communication. Specifically, the NodeMCU detects obstacles, relays messages through the MQTT broker, and the Raspberry Pi responds by blinking an LED, resulting in a responsive obstacle detection system.

TABLE OF CONTENTS

List of Figures

1 Introduction

In an increasingly interconnected world, the Internet of Things (IoT) has emerged as a transformative force, permeating every aspect of our lives. IoT devices are now an integral part of smart homes, industries, healthcare, transportation, and beyond. Within this expansive landscape, the "Pi-Node Bridge" project introduces a groundbreaking IoT communication system designed to solve a critical challenge: real-time obstacle detection and message relay.

The project "Pi-Node Bridge," aims to bridge the capabilities of two formidable yet distinct hardware platforms: the Raspberry Pi and the Node MCU. The driving force behind this endeavor is the need for a seamless and responsive obstacle detection system that can cater to a wide range of applications.

1.1 Hypothesis

The hypothesis of the "Pi-Node Bridge" project is based on the premise that creating a seamless communication bridge between the Raspberry Pi and NodeMCU using MQTT will enhance their collective capabilities for real-time obstacle detection and remote LED control. By integrating these two devices, we anticipate achieving a sophisticated solution that can significantly benefit various domains, such as home automation and industrial monitoring. This hypothesis submit that such a bridge will not only demonstrate the potential for collaboration between different hardware platforms but also contribute to the growing field of Internet of Things (IoT) applications.

1.2 Objectives

The objective of this project are:

1. **Hardware and Software Integration**: The primary objective of this project is to achieve a seamless integration of hardware and software components between the Raspberry Pi and NodeMCU. This will enable these devices to communicate effectively, sharing vital data for obstacle detection and LED control.
2. **Real-Time Obstacle Detection:** A key aim is to equip the NodeMCU with the capability to detect obstacles in real time using an ultrasonic sensor. The objective is to ensure precise, timely detection and data transmission for further analysis.
3. **MQTT-Based Communication:** The project's central objective is to implement the MQTT protocol, serving as the backbone for communication between the Raspberry Pi and

NodeMCU. This objective aligns with enabling seamless data exchange between the two devices.

4. **LED Control:** The project aims to enable the Raspberry Pi to control LED devices based on obstacle detection data received from the NodeMCU. This feature showcases the capabilities of the communication bridge.
5. **Versatility and Scalability:** A crucial objective is to design the "Pi-Node Bridge" to be versatile and scalable, ensuring its adaptability to diverse applications and contexts. This adaptability will help in the expansion of IoT and communication solutions.
6. **Documentation and Assessment:** Lastly, thorough documentation and rigorous evaluation of the project's performance and potential use cases are pivotal objectives. By providing detailed insights into the hardware setup, software infrastructure, and communication protocols, the project aims to contribute to the growing body of knowledge in IoT and communication technologies.

2 Literature Review

2.1 Raspberry Pi in IoT Projects

Raspberry Pi has become a staple in the Internet of Things (IoT) landscape. Researchers and hobbyists have employed Raspberry Pi for various IoT applications, such as home automation, environmental monitoring, and data collection. The flexibility and computational power of Raspberry Pi make it an ideal choice for IoT-related projects[1].

2.2 Node MCU

It focuses on leveraging IoT technology for waste management, aiming to enhance user convenience and implement an intelligent waste container system. Prior research in the field has explored the integration of microcontrollers and sensors to automate lid control and accurately detect garbage levels. The study combines experimental and survey methods, indicative of a practical research approach. Emphasis is placed on user comfort, with remote monitoring through web interfaces and messaging platforms, highlighting the user-centric design of the proposed IoT-based waste container control system[2].

2.3 MQTT Communication Protocol

MQTT, a lightweight and efficient communication protocol, has gained traction in IoT applications. MQTT's publish-subscribe mechanism facilitates communication between devices with low overhead. The protocol is used for remote monitoring, control, and sensor data transmission in various IoT projects[3].

2.3.1 Publish-Subscribe Model

MQTT follows a publish-subscribe communication model. In this model, there are two main entities: publishers and subscribers. Publishers are responsible for sending messages (also known as "publishing") to a specific topic. Subscribers express interest in receiving messages related to particular topics. When a publisher sends a message to a topic, all subscribers interested in that topic receive the message.

2.3.2 Lightweight Protocol

MQTT is designed to be lightweight, meaning it has minimal protocol overhead. This is crucial

for IoT devices with limited processing power, memory, and bandwidth. The protocol header is small, and the binary payload can be as small as one byte.

2.3.3 Quality of Service (QoS) Levels

MQTT supports different levels of message delivery assurance, known as Quality of Service (QoS).

QoS 0: The message is delivered at most once, with no confirmation.

QoS 1: The message is delivered at least once, and the sender receives acknowledgment of delivery.

QoS 2: The message is delivered exactly once by using a four-step handshake.

2.3.4 Persistent Connections

MQTT allows for persistent connections, which is crucial for IoT devices that may go in and out of network coverage. Devices can stay connected to the MQTT broker, and the broker retains information about subscriptions and queued messages for devices that temporarily disconnect.

3 System Requirements

The "Pi-Node Bridge" project necessitates specific hardware and software components. On the hardware side, it requires a Raspberry Pi (Model 3 or higher) and a NodeMCU, both powered and equipped with network connectivity. Additionally, it involves an ultrasonic sensor module, LEDs, a microSD card for the Raspberry Pi, and a set of breadboards and jumper wires for connections. Software requirements include operating systems (Raspbian for the Raspberry Pi and Arduino IDE for NodeMCU), Python, an MQTT broker (e.g., Mosquitto), Paho-MQTT libraries for communication, sensor libraries and drivers, control libraries for LED management, and development environments and tools for coding and testing. These combined hardware and software elements enable seamless communication between the Raspberry Pi and NodeMCU, facilitating obstacle detection and LED control via MQTT.

3.1 Hardware Requirements

3.1.1 Raspberry Pi (Model 3 or higher)

The project necessitates a Raspberry Pi shown in Figure 1 as the central processing unit to receive and process data from the NodeMCU, control the LEDs, and manage MQTT communication.

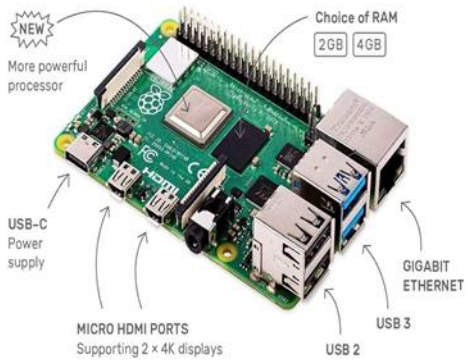

Figure 1: Raspberry PI 4 Model B

3.1.2 ESP8266 (NodeMCU)

The ESP8266 shown in Figure 2 is a versatile and cost-effective Wi-Fi module developed by Espressif Systems, widely used in IoT and embedded systems. It features a microcontroller unit, GPIO pins, and low power consumption, making it suitable for various applications. The module operates at 3.3V and includes onboard flash memory for program storage. Its integrated TCP/IP

stack enables internet communication, and it supports programming through the Arduino IDE or C using the ESP-IDF. With a large community and active support, the ESP8266 is known for its ease of use and flexibility in creating connected devices.

Figure 2: ESP8266 (NodeMCU)

3.1.3 Power Supplies

Both the Raspberry Pi and Node MCU require compatible power supplies to operate effectively. The Raspberry Pi, being a single-board computer, typically requires a micro USB power supply with specific voltage and current specifications to ensure stable and reliable operation. Using an incompatible power supply may result in performance issues or even damage to the device.

3.1.4 MicroSD Card

The Raspberry Pi relies on a microSD card to host the operating system and software, acting as the primary storage medium for the entire system. The microSD card serves as both the boot device and the storage space for the Raspberry Pi. When the Raspberry Pi is powered on, it reads the bootloader from the microSD card, initiating the process of loading the operating system into the device's memory. The operating system, along with applications and user data, is stored on the microSD card. The card's storage capacity determines how much data and software the Raspberry Pi can accommodate. Users can flash a compatible operating system image onto the microSD card using a computer, configuring the Raspberry Pi for specific use cases, such as a media center, web server, or IoT device. The microSD card can be easily replaced or upgraded, providing flexibility and ease of maintenance for Raspberry Pi users.

3.1.5 Ultrasonic Sensor Module

An ultrasonic sensor shown in Figure 3 is essential for obstacle detection and data collection.

Ultrasonic sensors use sound waves with frequencies higher than the human audible range to measure distances and detect objects. Here's how they typically work:

1. Emission of Ultrasonic Waves: The sensor emits ultrasonic waves, usually in the form of a sound pulse.
2. Reflection from Objects: These waves travel through the air and, when they encounter an object in their path, they bounce back toward the sensor.
3. Detection of Return Time: The sensor measures the time it takes for the emitted waves to return after hitting an object.
4. Distance Calculation: Using the known speed of sound in air, the sensor calculates the distance to the object based on the time it took for the waves to travel to the object and back.

Figure 3: Ultrasonic Sensor

3.1.6 LEDs

LEDs as shown in Figure 4 provides visual feedback and status indications, adding a practical and interactive dimension to the project. LEDs are widely employed for their simplicity, low power consumption, and versatility in signaling. In various projects, including those involving Raspberry Pi, Arduino, or other electronic systems, LEDs are used to convey information to users. Here are common ways in which LEDs contribute to this project:

1. Status Indicators: LEDs can represent different states or conditions in a system. For instance, a green LED might indicate that a device is powered on, a red LED could signal an error, and a flashing LED might suggest active communication.
2. Feedback on User Input: In interactive projects, LEDs often provide feedback in response to user actions. For example, pressing a button might trigger an LED to light up, confirming that the input has been registered.
3. Debugging and Troubleshooting: LEDs are valuable during the development and testing

phases of a project. They can be programmed to provide feedback on specific conditions, aiding in debugging and troubleshooting.

4. Notification and Alarming: LEDs are used to notify users of specific events, such as incoming messages, system alerts, or alarms. Different colors or blinking patterns can convey different types of information.
5. Customizable Signaling: LEDs can be programmed to exhibit various patterns, colors, and intensities to represent specific information. This allows for creative and customizable visual signaling in projects.

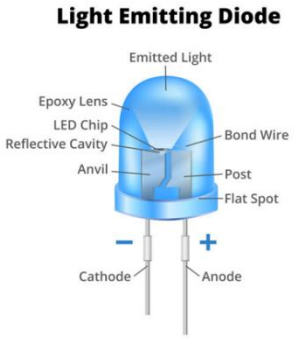

Figure 4: LED

3.1.7 Breadboard

These components are needed to establish connections and interface various hardware elements.

1. Functionality: Breadboards are essential tools for prototyping electronic circuits without the need for soldering. They consist of a grid of holes with metal clips beneath the surface that allow components to be easily inserted and connected.
2. Usage: Components can be plugged into the holes, and their leads make contact with the metal clips. This allows you to experiment with different circuit configurations quickly and make changes without soldering.

3.1.8 Jumper Wires

1. Functionality: Jumper wires are used to create electrical connections between components on the breadboard or to extend connections from the breadboard to other components.
2. Types: Jumper wires come in various types, including male-to-male, male-to-female, and

female-to-female, allowing you to connect different types of components together.

3. Colors and Lengths: Jumper wires are often color-coded for organization, and they come in different lengths to suit various distances within a circuit.

3.1.9 Network Connection

An internet connection is necessary to enable MQTT communication between the Raspberry Pi and the NodeMCU. A network connection is crucial for enabling MQTT communication between a Raspberry Pi and a NodeMCU. MQTT (Message Queuing Telemetry Transport) relies on a network infrastructure to facilitate communication between devices. Here's how network connectivity plays a role in MQTT communication:

3.1.9.1 Broker-Client Architecture:

1. MQTT communication typically involves a broker, which is a central server responsible for receiving messages from clients and distributing them accordingly.
2. The Raspberry Pi and the NodeMCU would function as MQTT clients, connecting to the MQTT broker to exchange messages.

3.1.9.2 Network Communication:

1. If both the Raspberry Pi and the NodeMCU are within the same network, they can communicate directly using IP addresses.
2. Network communication is often faster and doesn't involve the complexities associated with traversing the internet.

3.2 Software Requirements

3.2.1 Operating System and Environment

3.2.1.1 Raspbian Operating System

The Raspberry Pi typically runs on an operating system like Raspbian (now known as Raspberry Pi OS) or other compatible Linux distributions. Raspbian is a Debian-based Linux distribution optimized for the Raspberry Pi, providing a full operating system environment.

3.2.1.2 Arduino IDE for NodeMCU

The Arduino IDE is a popular integrated development environment for programming microcontrollers, including the ESP8266 and ESP32 and so on. Users can write Arduino-compatible code, upload it to the NodeMCU through the Arduino IDE, and execute it on the microcontroller.

3.2.2 Programming Language

Python and C/C++ program is the primary programming language used for scripting and coding in this project.

3.2.2.1 Python on Raspberry Pi

1. Scripting Language: Python is widely used as a scripting language on the Raspberry Pi. Its readability and simplicity make it an excellent choice for quick prototyping, automation, and scripting tasks.
2. Supportive Community: The Raspberry Pi community often favors Python due to its ease of learning and the vast ecosystem of libraries and modules available for various applications, including GPIO control, web development, and IoT projects.
3. Libraries for GPIO and Hardware Access: Python provides libraries like RPi.GPIO for GPIO (General Purpose Input/Output) access on the Raspberry Pi, making it convenient for hardware interfacing.

3.2.2.2 C/C++ on NodeMCU

1. Low-Level Programming: C/C++ is a low-level programming language that allows for fine-grained control over hardware resources. It's suitable for applications where performance and direct hardware access are crucial.
2. NodeMCU Firmware: For the NodeMCU, while it's commonly programmed using the Arduino IDE with Arduino-like languages, the underlying firmware may include components written in C or C++ to handle low-level functionalities

3.2.3 MQTT Broker

Installing an MQTT broker on the Raspberry Pi, such as Mosquitto from test.mosquitto.org or any other compatible broker, is crucial for enabling message transmission and communication in an MQTT-based system. Here are the key points explaining why:

3.2.3.1 MQTT Broker Role

1. An MQTT broker serves as a central hub for managing communication between MQTT clients (devices or applications).
2. It receives messages published by clients and delivers them to clients subscribed to relevant topics.

3.2.3.2 Publish-Subscribe Architecture:

1. MQTT follows a publish-subscribe architecture where clients can publish messages to specific topics, and other clients can subscribe to receive messages from those topics.
2. The broker facilitates the distribution of messages based on the publish-subscribe model.

3.2.3.3 Message Routing

1. The MQTT broker is responsible for routing messages between publishing and subscribing clients.
2. It keeps track of active clients, their subscriptions, and ensures the delivery of messages to the correct recipients.

3.2.4 Paho-MQTT Libraries

The Paho MQTT libraries are essential for facilitating MQTT communication between devices such as the Raspberry Pi and other devices. The Paho MQTT libraries provide implementations in various programming languages, including Python (for the Raspberry Pi).

3.2.5 Sensor Libraries and Drivers

Appropriate libraries and drivers are essential to interface with the ultrasonic sensor on both devices.

3.2.6 Development Environment

Adequate development environments and tools, including text editors and integrated development environments (IDEs), are necessary for coding, debugging, and testing the project components.

3.2.7 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a platform tailored for developing embedded systems. It provides a comprehensive environment where developers can seamlessly write, compile, and upload firmware code to Arduino microcontrollers. The IDE features a user-friendly interface, marrying a powerful code editor with essential tools, making it accessible to both beginners and experienced developers. Its extensibility allows the integration of third-party libraries, contributing to its versatility and appeal in crafting custom solutions.

4 Result

The planned timeline for this project was approximately 4 to 5 weeks, providing a structured framework to execute and accomplish various milestones. This timeframe was meticulously designed to balance efficiency and thoroughness throughout the project's lifecycle. Each week was allocated specific tasks and objectives, ranging from the initial setup of the development environment and hardware configuration to the final stages of testing, debugging, and documentation. The brief description of work done in each week are elaborated below:

4.1 Week 1

In week 1 following task was completed:

1. Successfully set up the development environment, configuring the necessary tools and packages for programming.
2. Installed required software components, ensuring a seamless foundation for the upcoming development stages.
3. Familiarized myself with the project requirements, specifications, and potential challenges.

4.2 Week 2

In week 2 following task was completed:

1. Completed the hardware setup and configuration, assembling components such as the Raspberry Pi and NodeMCU.
2. Ensured proper connections and power supply for all hardware elements, paving the way for reliable functionality.
3. Successfully executed initial hardware tests, including the blink functionality of the built-in LEDs on GPIO pins 2 and 16 on NodeMCU.

4.3 Week 3

In week 3 following task was completed:

1. Focused on software development and IoT communication, establishing independent communication for each device with the MQTT broker.
2. Developed software components, leveraging technologies like PahoMqtt, tailored for both

Raspberry Pi and NodeMCU.

3. Implemented the MQTT protocol to facilitate efficient communication and verified proper data exchange functionality.

4.4 Week 4

In week 4 following task was completed:

1. Implemented the LED and ultrasonic sensor functionalities, seamlessly integrating them into the IoT communication system.
2. Conducted thorough testing of the entire communication system, ensuring robustness and identifying potential issues.
3. Proactively addressed any identified hardware or software issues, ensuring a stable foundation for the final project.
4. Conducted stress testing to gauge the system's reliability under various conditions.

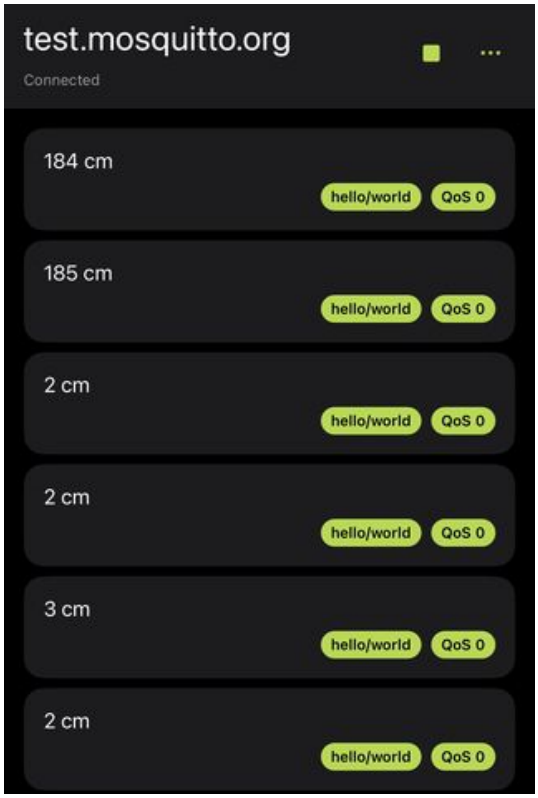

Figure 5: MQTT Broker (test.mosquitto.org)

4.5 Week 5

In week 5 following task was completed:

1. Dedicated the week to meticulous debugging of the implemented system, ensuring all components work cohesively.
2. Prepared diligently for the demonstration of the "Pi-Node" ensuring a successful and seamless presentation.
3. Documented the entire project, capturing the development process, challenges encountered, and outcomes achieved.
4. Ensured comprehensive organization of project documentation for future reference or replication, marking the successful completion of the final project.

The demo of this project is shown in this YouTube video link given below:

https://www.youtube.com/shorts/gt0mtLpTfgc

The outputs are as shown in Figure 6, Figure 7 and Figure 8.

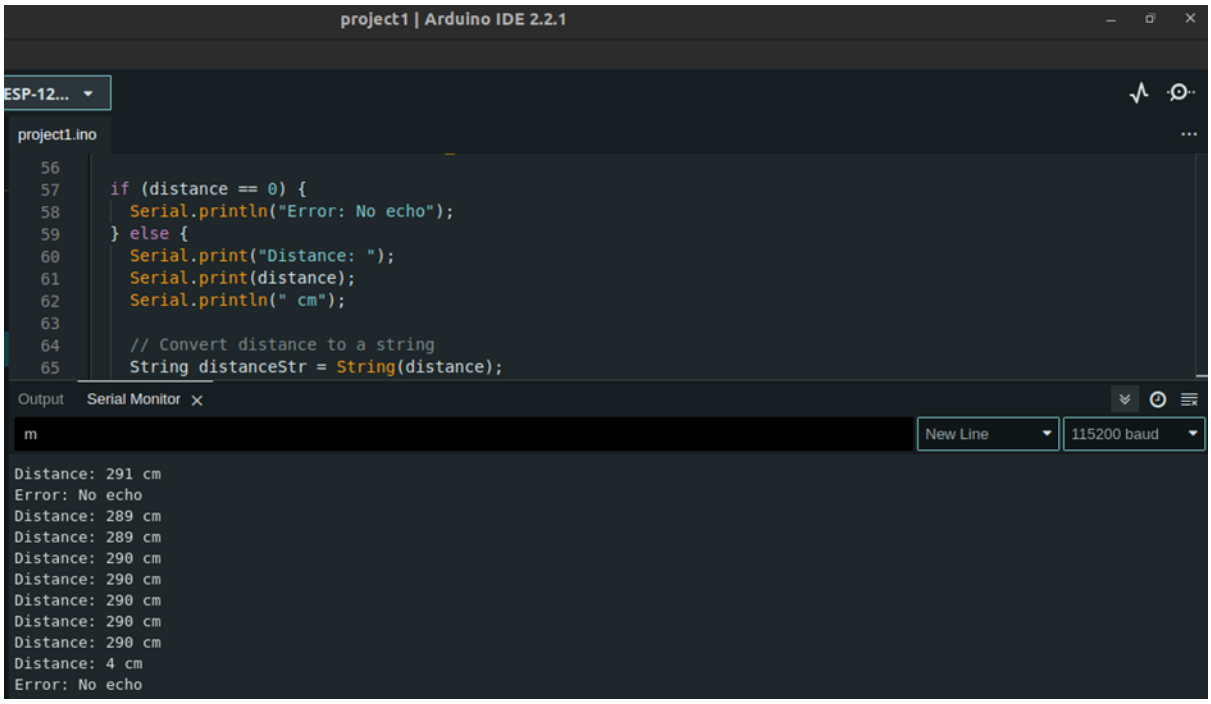

Figure 6: ESP8266 Serial Monitor

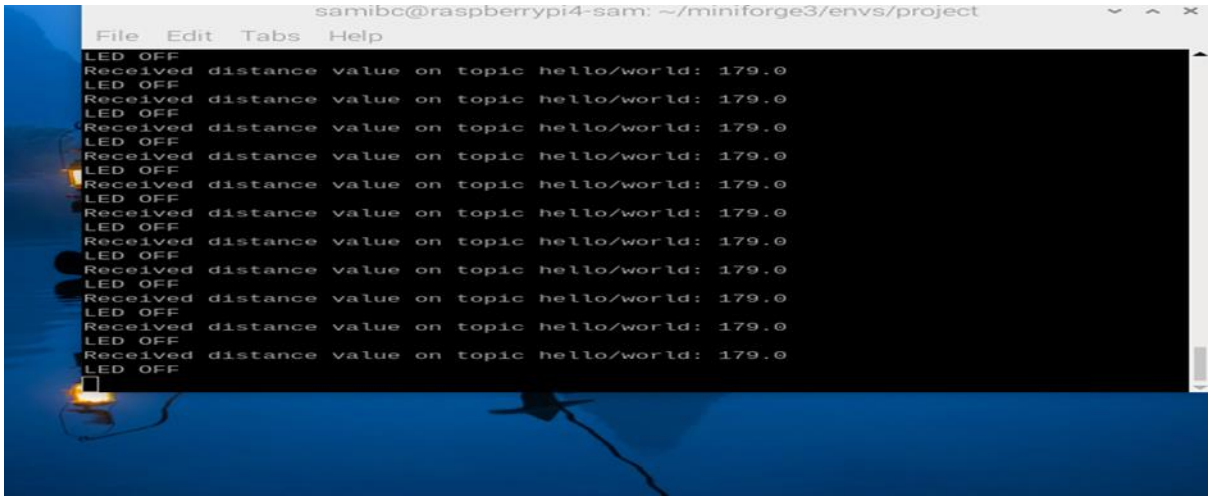

Figure 7: Raspberry Pi receiving data and blink led

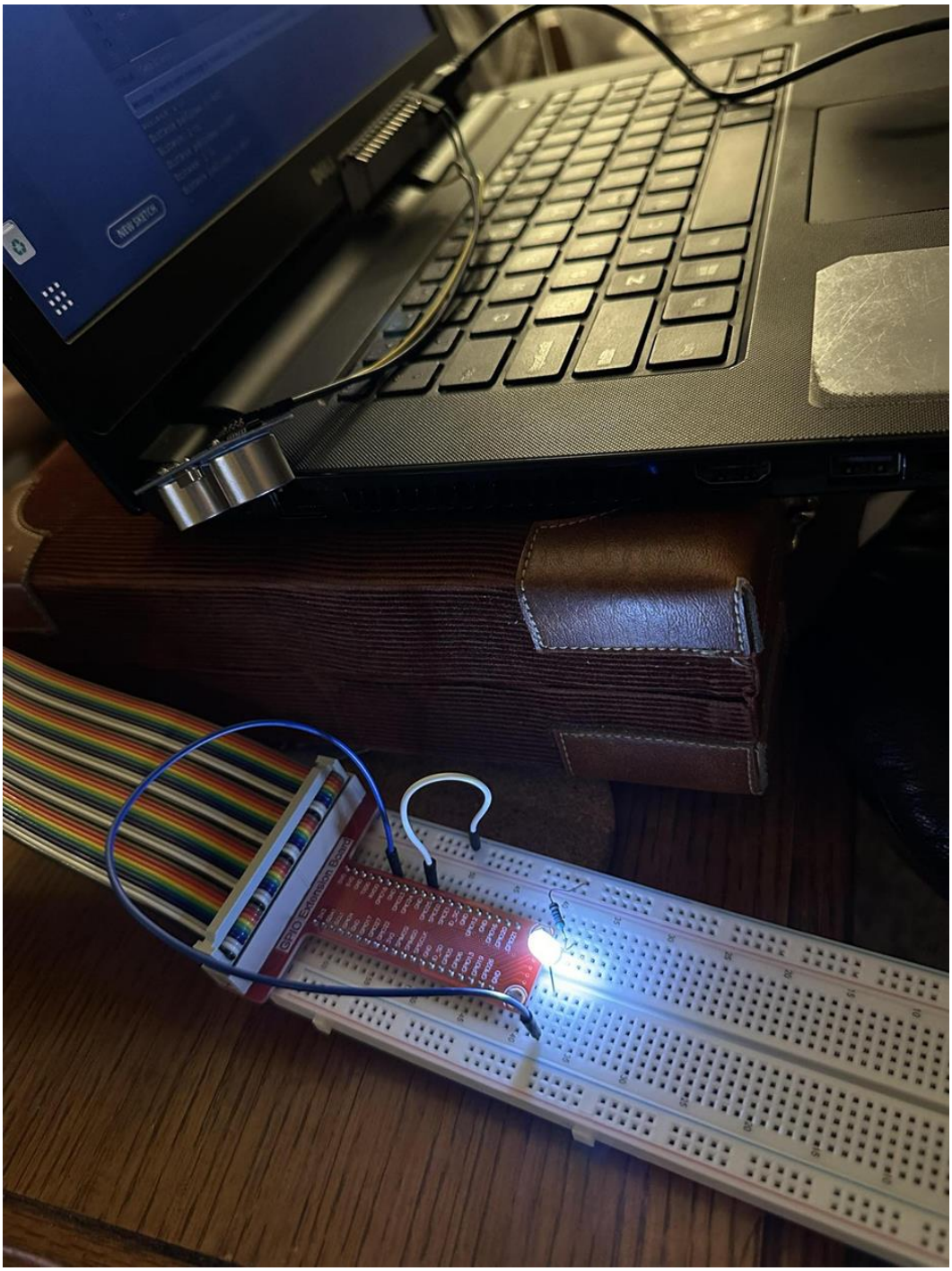

Figure 8: Ultrasonic detecting the obstacle and led blinking

5 Techniques, Difficulties and Solution

For an MQTT communication project between two IoT devices (NodeMCU with an ultrasonic sensor and Raspberry Pi with an LED indicator), here are the major techniques and considerations:

5.1 MQTT Protocol

5.1.1 Technique

Use the MQTT protocol for communication between the NodeMCU and Raspberry Pi. MQTT is a lightweight and efficient protocol designed for reliable communication in constrained environments.

5.1.2 Difficulty

Configuring and setting up MQTT communication require understanding MQTT concepts, such as brokers, topics, and QoS levels.

5.1.3 Solution

I choose a reliable MQTT broker (server) and ensure that both devices can connect to it. Libraries such as PubSubClient for ESP8266(NodeMCU) and paho-mqtt for Python (Raspberry Pi) can simplify MQTT implementation.

5.2 Publish-Subscribe Model

5.2.1 Technique:

Implement the publish-subscribe model where the NodeMCU publishes the ultrasonic sensor data to a specific MQTT topic, and the Raspberry Pi subscribes to that topic to receive the data.

5.2.2 Difficulty

Ensuring proper topic naming, understanding the concept of publishers and subscribers, and handling multiple subscribers.

5.2.3 Solution

Define a clear topic structure, and ensure that both devices are using the correct topics. Consider using unique identifiers or device names in topics.

5.3 Distance Measurement with Ultrasonic Sensor

5.3.1 Technique

Use the ultrasonic sensor to measure distances and send this data to the Raspberry Pi.

5.3.2 Difficulty

Calibration of the ultrasonic sensor, handling noise in distance measurements, and ensuring accurate data transmission.

5.3.3 Solution

Calibrate the sensor in the actual environment, filter out noise in the sensor readings, and implement error-checking mechanisms in your code.

5.4 Programming Languages

5.4.1 Technique

Use simplified version of C++ (Arduino IDE) for NodeMCU and Python for Raspberry Pi.

5.4.2 Difficulty

Integrating code written in different languages, library compatibility, and ensuring synchronization between the two devices.

5.4.3 Solution

Use libraries that are available for both platforms, and make sure that the communication protocol (MQTT) is well-supported in both languages.

5.5 Security

5.5.1 Technique

Implement security measures such as username/password authentication for MQTT and secure communication over the network.

5.5.2 Difficulty

Ensuring secure data transmission and preventing unauthorized access.

5.5.3 Solution

Set up secure MQTT communication with proper authentication, use secure topics, and consider encryption if sensitive data is involved.

6 Conclusion

In summary, the "Pi-Node Bridge" project has accomplished a seamless integration of IoT devices, specifically the NodeMCU and Raspberry Pi, to create an efficient obstacle detection and communication system. Leveraging ultrasonic sensors on the NodeMCU, obstacles are promptly identified, and MQTT serves as the lightweight and reliable communication protocol to relay messages to the Raspberry Pi. The LED blinking response on the Raspberry Pi demonstrates the system's real-time and responsive nature. This project not only showcases the effectiveness of interconnected devices in addressing practical challenges but also highlights the significance of thoughtful communication protocols, providing a foundation for future advancements in intelligent and responsive networks.

7 References

[1] T. Saheb and L. Izadi, “Paradigm of IoT big data analytics in the healthcare industry: A review of scientific literature and mapping of research trends,” *Telemat. Informatics*, vol. 41, pp. 70–85, 2019, doi: 10.1016/j.tele.2019.03.005.

[2] A. Anggrawan, S. Hadi, and C. Satria, “IoT-Based Garbage Container System Using NodeMCU ESP32 Microcontroller,” *J. Adv. Inf. Technol.*, vol. 13, no. 6, pp. 569–577, 2022, doi: 10.12720/jait.13.6.569-577.

[3] M. Kashyap, V. Sharma, and N. Gupta, “Taking MQTT and NodeMcu to IOT: Communication in Internet of Things,” *Procedia Comput. Sci.*, vol. 132, no. Iccids, pp. 1611–1618, 2018, doi: 10.1016/j.procs.2018.05.126.