HACETTEPE UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BBM460 WIRELESS AND MOBILE NETWORKS LAB - 2025 SPRING

Project Final Report

May 26, 2025

Student names:	*Student numbers:*
İlker AVCI	b2210356061
Buğra Kağan ACAR	b2210356097

Contents

1 Abstract

Our project is about building a system to find earthquake victims using Bluetooth signals from their phones. We're using ESP32 probes to pick up these signals, MQTT to send the data, and a main system to figure out the locations using multilateration. So far, we've gotten a good understanding of how Bluetooth Classic and BLE work, set up discovery-based sniffing for Bluetooth Classic, managed active and passive sniffing for BLE, and made a basic GUI to show where devices are.

2 Introduction

One of the most crucial aspects of responding to natural disasters is locating victims quickly and accurately. This task becomes significantly more challenging when traditional communication networks are unavailable. However, many individuals carry Bluetooth-enabled devices—such as smartwatches, wireless earbuds, and fitness trackers—that continuously emit signals. Our project leverages these ubiquitous Bluetooth signals to develop a system capable of identifying and locating individuals in distress. By harnessing existing technology in a new way, our goal is to enhance the efficiency of search and rescue operations and ultimately help save lives in the aftermath of disasters.

3 System Design

The system is composed of two main parts and the MQTT broker:

3.1 Backend / Probes

Probes are responsible for periodically scanning the environment for Bluetooth Low Energy advertisement packets, determining each packet's corresponding signal strength (RSSI), and publishing the gathered data as MQTT packets through WiFi. In this project, we used four ESP32-based NodeMCU-32s devices as probes.

3.2 Frontend / GUI

The GUI is responsible for processing the raw signal strength data coming from MQTT, using some filtering and sampling techniques to calculate the correct location of the Bluetooth devices, and displaying them on a 2D map.

3.3 Mosquitto MQTT Broker

The MQTT broker is at the heart of the communication between the probes and the GUI. It implements a publish/subscribe pattern for passing messages between publishers and subscribers of different topics. We deployed our own instance of Mosquitto MQTT broker using Podman (a better alternative to Docker) on our own Debian Linux server[3], because the free brokers were too unreliable and had rate limits which didn't work well with our project.

4 Method

As for the method, we gathered the data sent from the probes using MQTT. We initially tried to implement the multilateration by treating the 4 ESP32's as spheres and we tried to calculate their collision point but we quickly figured out that the RSSI readings weren't accurate enough to that.

Regardless, for the multilateration we first had to translate the RSSI values from the probes to real-life distance. We utilized a formula like this:

$$RSSI = -10n \log_{10}(d) + A$$

with the n value being the propagation constant and A value being a calibration constant. We measured the RSSI values of objects at 1m distance and derived the constants from there. Unfortunately for us, all of our ESP32's had enough differences that we had to do this for each probe.

We then fed the distance values to a least squares method multilateration algorithm we found on a paper[1]:

$$A = \begin{bmatrix} 2(x_1 - x_n) & 2(y_1 - y_n) \\ \vdots & \vdots \\ 2(x_{n-1} - x_n) & 2(y_{n-1} - y_n) \end{bmatrix}$$

$$x = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$$

$$b = \begin{bmatrix} x_1^2 - x_n^2 + y_1^2 - y_n^2 + d_n^2 - d_1^2 \\ \vdots \\ x_{n-1}^2 - x_n^2 + y_{n-1}^2 - y_n^2 + d_n^2 - d_{n-1}^2 \end{bmatrix}$$

$$x = (A^T A)^{-1}(A^T b)$$

with the algorithm, we were able to calculate the locations of devices. We debated on implementing the Kalman Filter on the paper but we decided to go with averaging as in the paper the Kalman Filter was not shown to be significantly better than averaging.

5 Technologies

- For the backend: ESP32, C++, PlatformIO
- For the frontend: Rust, iced-rs
- For the MQTT broker: Debian Linux server, Podman, Mosquitto MQTT

6 Limitations

There are two flavors of Bluetooth[4]; Bluetooth Classic and Bluetooth Low Energy (BLE). Our initial plan was passively sniffing the packets for both flavors and thus having a more accurate positioning data. However, we encountered two major limitations on this front.

The first limitation stems from the very nature of the Bluetooth Classic and BLE protocols. Unlike the TCP/IP stack and WiFi protocol, regular Bluetooth packets do not carry the source and destination device addresses. Instead, those addresses are only used during the advertisement, pairing and discovery phases. The data packets use an identifier called access code, which is determined in the pairing stage. That's why we were unable to consider passive sniffing as a reliable option.

The second limitation arose from our device selections. We had chosen NodeMCU-32s boards which are based on the ESP32 platform. NodeMCU-32s was the cheapest ESP32 board option we could find, which had some limitations with the cheap price. The flash storage turned out to be significantly smaller than the original ESP32-based boards, which resulted in smaller possible program sizes. This limitation was so impactful that we couldn't have WiFi and Bluetooth functionalities simultaneously. Upon further investigation, we had found out that there were two Bluetooth stack implementations available for the ESP32[5]: Bluedroid (the original stack with support for both Bluetooth Classic and BLE), and Apache NimBLE[6] (the more lightweight stack with support for only BLE). We had to choose the latter option in order to keep the WiFi (and thus, MQTT) functionality.

However, dropping support for Bluetooth Classic wasn't that much of a loss in our case. Even though both flavors sharing the same name, Bluetooth Classic and BLE are two very distinct protocols. In Bluetooth Classic, the device addresses (much alike to MAC addresses) are only available upon activating explicit discovery mode, and the discovery is initiated by the master (host) devices, rendering passive discovery impossible. Bluetooth Low Energy, on the other hand, has the opposite approach. The pairing is implicitly initiated by the slave (client) devices by periodically broadcasting advertisement packets which include their device address. This approach fits better for our passive discovery purposes, so not being able to support Bluetooth Classic was much less of an issue for us.

Another big limitation we have realized too late is the multipath effect we had encountered with Bluetooth signals. Due to the frequency range of Bluetooth, we had some issue with reflection from the walls and other solid objects which resulted in multipath effect and unreliable signal strength detection. We tried to counter this by using data from multiple probes and some sampling/filtering techniques and we were able to maximize the accuracy to some extent. Also, we had to give up on the 3rd dimension in order to increase the accuracy on 2 dimensions.

7 Conclusion

With this, we conclude this report. Because of the aforementioned limitations, we were unable to fully build the system that we envisioned at the time of the project. However, we still believe that we managed to make a decent project and we learnt a lot of stuff related to wireless communication

(Bluetooth too, specifically) during this. We have also made the code available on GitHub for public benefit[2].

References

[1] Röbesaat, J., Zhang, P., Abdelaal, M., & Theel, O. (2017). An Improved BLE Indoor Localization with Kalman-Based Fusion: An Experimental Study. Sensors (Basel, Switzerland), 17(5), 951. https://doi.org/10.3390/s17050951,

[2] https://github.com/FOSSketeers/bluedar

[3] https://github.com/sukesh-ak/setup-mosquitto-with-docker

[4] https://www.espressif.com/sites/default/files/documentation/esp32_bluetooth_architecture_en.pdf

[5] https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/bluetooth/index.html

[6] https://github.com/espressif/esp-nimble