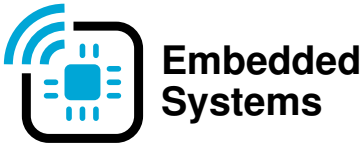
Delft University of Technology
Master of Science Thesis in Embedded Systems

Federated Approach to Bluetooth Indoor Localisation Models at The Edge

Chiel Jacobus Johannes van Diepen



TU Delft Delft University of Technology

Federated Approach to Bluetooth Indoor Localisation Models at The Edge

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Chiel Jacobus Johannes van Diepen

21-04-2024

Author
Chiel Jacobus Johannes van Diepen
Title
Federated Approach to Bluetooth Indoor Localisation Models at The Edge
MSc Presentation Date

27-05-2024

Graduation Committee

dr. Przemysław Pawełczak (Chairman)	Delft University of Technology
dr. Marco Zuñiga Zamalloa (Committee)	Delft University of Technology
Peet van Tooren (Company supervisor)	Almende

Abstract

Indoor localisation is a well-researched topic and it is a challenge to improve the accuracy of existing techniques. In recent years, edge computing and federated learning have opened up new possibilities and challenges for indoor localisation. This thesis presents a federated implementation for spatial mapping of the network based on the RSSI signal between nodes. The proposed approach decentralises indoor localisation and produces multiple coordinate maps of the nodes in a network. The coordinate maps and edge data are from multiple node perspectives, which can improve the aggregation of the coordinates into a single map. The algorithm is tested in a BLE network and is compared with other current methods. In addition, a simulator is built that serves as a proof of concept for the actual implementation of the algorithm. The results of the simulator prove that the mathematical aspects are correct and that the algorithm consistently reproduces the same structure with a stable signal. Time-series analysis of RSSI measurements along with environmental factors unveils periodic noise components such as humidity and human presence, along with the notable influence of node placement and ambient noise. Consequently, the conventional RSSI-to-distance model proves inadequate in adapting to such noise sources. The spatial mapping algorithm helps in creating a 3D structure in a federated manner and provides a framework and location reference for future research that employs adaptable machine learning models. However, its localisation accuracy is still limited in comparison to other preexisting techniques.

"Indoor localisation has the potential to revolutionise the way we navigate and interact with indoor spaces. However, it is important to consider the privacy implications and ensure that technologies like this are used ethically and responsibly." – ChatGPT

"Be careful BLE is watching you!" – Crownstone

Preface

This thesis research is conducted at Almende to obtain a Master's degree in Embedded Systems. Upon completion of a hardware project focusing on IoT and edge AI for Dr. Przemysław Pawełczak, known as OpenEarable, my enthusiasm for IoT and making machine learning more accessible to users persisted. Almende is similarly investigating this area through their Crownstone products, which presently utilise various indoor localisation methods reliant on BLE signal information. A compelling query arises regarding the potential improvement of BLE localisation directly on the device by utilising federated algorithms and implementing localisation models onboard. Almende has given me the opportunity to further explore this research area.

First, I want to thank Almende and Crownstone for the graduation place and in particular my company supervisor Peet van Tooren for all the help and his infinite knowledge of programming and embedded systems. Peet is like a walking encyclopaedia providing all the knowledge you need (and more). Thank you for all the sparring sessions on various topics and guidance. Second, I would like to express my gratitude to my colleague Sander Steeghs-Turchina for assisting with programming and testing. At last, I want to thank my TU Delft supervisor Dr. Przemysław Pawełczak for the guidance and all the feedback on my thesis.

Chiel Jacobus Johannes van Diepen

Delft, The Netherlands
22nd May 2024

Contents

Chapter 1

Introduction

The practice of outdoor localisation of individuals is a well-established field, with numerous reliable techniques and technologies in place. However, when it comes to indoor settings, there is a distinct set of challenges that conventional methods such as GPS struggle to address. This situation has led to the need for the development of innovative techniques specifically tailored to indoor localisation. Extensive research efforts have been dedicated to indoor localisation methods over the years, resulting in significant progress. These methods often rely on wireless technologies or movement sensors. Despite these advances, the field of indoor localisation faces continuous challenges due to the unique characteristics of indoor environments, including multipath propagation, signal interference, and variable structures. In recent years, edge computing and federated learning have opened up new possibilities and challenges for indoor localisation. However, its application in indoor localisation remains relatively unexplored and under-researched.

The thesis begins with the background and problem statement of indoor localisation, which will give a comprehensive overview of indoor localisation technologies. It introduces the concept of federated algorithms and spatial mapping. Following the literature review, which summarises the existing indoor positioning systems (IPS) in use and presents additional perspectives, the problem statement, contributions and challenges will be outlined. This will be followed by a discussion of the methodology used in creating the simulator and algorithms, along with details on network specifications and initial assumptions. In the forthcoming chapter, the implementation of the simulator and the federated spatial mapping algorithm will be described, providing important code concepts and code illustrations. In addition, the test setup will also be discussed. Subsequently, the results of both systems will be evaluated and the federated spatial network mapping algorithm will be compared with other current methods. Finally, the thesis concludes by outlining limitations, presenting final remarks, and suggesting avenues for future work, accompanied by an appendix with additional information.

Chapter 2

Background and Problem Statement

The demand for accurate indoor location services has experienced rapid growth across diverse industries, such as healthcare and smart homes. This surge is attributed to the limitations of GPS and outdoor positioning technologies when applied in indoor environments. Achieving indoor localisation is possible with different levels of accuracy. It can range from room-level (2D) to more precise measurements in square meters or even a few centimetres. However, when aiming for 3D positioning, the approach becomes more complex but provides even more realistic localisation.

This chapter will provide supplementary details about indoor localisation technologies, federated algorithms, and the ongoing research concerning current BLE Indoor Positioning Systems. Furthermore, the chapter will offer additional insights from related work, outline the problem statement, and highlight its contributions.

2.1 Indoor Localisation Technologies

Various technologies can be used for indoor localisation, each with its advantages and disadvantages, as outlined in Hayward et al. [20]. GNSS, commonly used for outdoor localisation, suffers from low indoor accuracy and requires expensive large antennas, rendering it unusable indoors. Line-of-sight-dependent technologies like vision, visible light, or acoustic (ultrasound) are not very useful in offices or homes where items could easily block the sensors. Instead, a low-power MEMS sensor can be installed on people or assets to track their movements. However, this method provides only movement and direction. Combining it with other techniques can solve this issue and improve the accuracy of these technologies. Wireless technologies like WiFi, UHF, and Ultra Wideband are expensive to deploy and consume a lot of power, making them unsuitable for low-power applications. However, Zigbee or Bluetooth Low Energy (BLE) offer low power consumption, small size, low cost deployment, and high scalability, and are often used for home automation. Using these popular home automation technologies, it is possible to measure distances and track assets with existing systems.

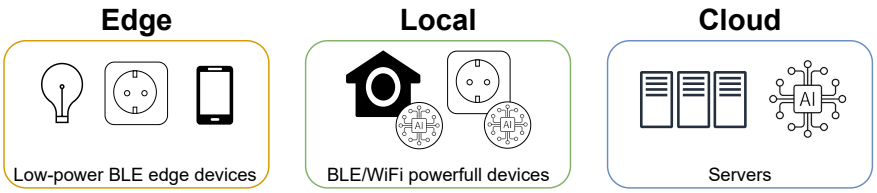


Figure 2.1: **Hierarchy of computing devices: Edge, Local and Cloud.**

BLE technology stands out due to these advantages and the fact that most people have a BLE-enabled phone that can be tracked. Moving forward, the thesis will concentrate on BLE indoor localisation. There are two distinct categories of BLE techniques in use: range-techniques or signal properties, and range-combining or positioning algorithms. Ranging techniques include Angle of Arrival (AoA), Time of Arrival (ToA), Time Difference of Arrival (TDoA), Channel State Information (CSI) and Received Signal Strength Indication (RSSI). On the other hand, range-combining or positioning algorithms consist of triangulation, trilateration [15], proximity, fingerprinting, particle filters, anomaly detection, and machine learning. Fingerprinting and the combination with machine learning are widely used by other researchers and in publications [25] [34] [32] [2] [6] [21] [29] [23]. Section 2.3 provides a more comprehensive overview of relevant BLE indoor localisation methods from the past decade.

2.2 Federated Algorithms at The Edge

'Federated algorithms' is a topic that has gained significant attention from researchers in recent years. They offer a solution to distributed computing problems where data is spread across multiple nodes or devices. Federated algorithms are not limited to machine learning and can be applied to localisation, optimisation problems, data synchronisation methods, distributed databases, and other scenarios that require decentralised processing. One exciting application of federated learning is its combination with a BLE mesh indoor localisation sensor node network, which allows for a decentralised approach to localisation, machine learning and other algorithms. Each node runs the algorithm with its local data at the Edge. At "The Edge" refers to computing infrastructure that is closer to the source of data generation or consumption. In Figure 2.1, the device hierarchy is illustrated, representing devices on the edge, local, or in the cloud. Every edge device is capable of communicating with its immediate neighbouring edge nodes but is kept to a minimum. The Edge nodes share only its outcome with a local hub, which in turn may transmit the data to the cloud server. This enables collaboration on global problems without the need to know the whole network. The cloud can be used for larger operations which demand more power like large AI models. More power-demanding applications or processes can also be offloaded to other local devices like a hub.

There are several key concepts related to federated algorithms, each with its challenges, which will be discussed in Section 2.6.

1. Decentralisation: distributed data computation across multiple nodes or

devices leads to parallelisation and easy scaling.

2. Data privacy-preserving techniques: location data for example is very privacy-sensitive and should be kept private between the nodes by generalising the shared results and by keeping it local.

3. Aggregation and fusion: the results of multiple nodes will be combined to form a larger linked result or model. In our situation, this task will be delegated to a nearby hub that is capable of handling larger workloads and computations.

4. Convergence: ensuring that the algorithm converges to meaningful results and meets the specified criteria so that it can be deployed and used.

2.2.1 Federated Spatial Mapping

The concept of federated localisation involves multiple nodes in a BLE network, each determining the locations of the other surrounding nodes, tags, or objects with minimal communication. The results will be sent to a central point for aggregating to a final location. Federated localisation can be divided into two steps: the mapping of the spatial network and the localisation within this spatial map.

Federated spatial mapping is a process in which multiple nodes construct a three-dimensional spatial structure of the best-connected surrounding BLE nodes. This process is based on the neighbour's received signal strength indicator (RSSI). This is the only information that a node can gather or request. The results are then aggregated into a comprehensive map of the entire BLE network at a central location, improving the locations and reducing noise through multiperspective analysis for improved structural integrity.

After a successful run of the spatial network mapping algorithm, the 3D spatial structure with coordinates can be used for localisation. The structure information can also be used in combination with other localisation techniques to enhance the accuracy.

2.2.2 Federated Learning

Federated learning is probably the most well-known federated algorithm and is a machine learning technique that allows multiple devices to work together to improve the accuracy of a global model without compromising the privacy of their data. Rather than sending data to a central server for processing, federated learning allows devices to learn from each other's experiences by exchanging only the necessary information, such as the model parameters. This decentralised approach to machine learning is particularly useful in situations where data privacy is a concern and enables devices to continuously improve their performance over time.

2.3 Related Work

This section focuses on providing insight into the latest techniques for indoor location. Various new techniques have been implemented that have led to significant advancements in this field. To recap, this research will focus only on BLE

Table 2.1: **Overview of relevant BLE indoor localisation techniques with criteria ratings**

Techniques/Methods	Accuracy	Devices	Bluetooth	Energy Supply	Usage[1]	Data flow[2]
Fingerprinting (FP.)(RSSI)	1.42m [9] [3]	nodes[3], sniffer, phone/tags	BLE v4+	Battery/USB	High	centralised
Multi/Trilateration	0.45-1.85 m [33]	nodes, tags, phone	BLE v4+	Battery/USB	High	centralised
Angle of Arrival	0.36m [8]	nodes, phone, tags	BLE v4+	Battery/USB	Low	centralised
BLE landmark: DTW	0.42-0.75m [40]	nodes, phone	BLE v4+	Battery/USB	Low	centralised
Time of Arrival + RSSI	2.0–2.7m [13]	nodes, sniffer	BLE v4+	Battery/USB	Low	centralised
Bloc: (CSI)	0.86m [7]	nodes, master, tags	BLE v4+	Battery/USB	Low	centralised
ML (Classification, FP.)	0.3-2.5m	nodes, sniffer, phone/tags	BLE v4+	Battery/USB	Med	centralised
Xgboost: CWT, RSSI	1.5m [25]	”	”	”	”	”
KNN (RSSI)	1m [29]	”	”	”	”	”
LSTM	2.44m [36]	”	”	”	”	”
SVM RMSE, DNN	0.5-1m [35][34][2]	”	”	”	”	”
RF classification	0.3m [21]	”	”	”	”	”
Linear Regression	0.9m [34]	”	”	”	”	”
FPFE (Encoders)	0.68-1.43m [23]	”	”	”	”	”

[1] The application of this technique in commercial products can be categorised as high, medium, or low.
[2] Computation tasks are delegated to the cloud (centralised) or executed locally or at the Edge (federated).
[3] Nodes are the same as anchors.

technology for indoor localisation 2.1. Section 2.3.1 provides a comprehensive overview of relevant BLE indoor localisation techniques. Section 2.3.2 discusses other insights and techniques that have been used in this field.

2.3.1 Current BLE Indoor Positioning Systems

Indoor Positioning Systems (IPS) aim to solve the challenging task of locating objects or people indoors. IPS can utilise different techniques or a combination of them, each with its advantages and disadvantages. To gain a better understanding of the differences between them, extensive research has been conducted on BLE localisation techniques. The findings have been summarised in a table for ease of reference. Method-specific articles and surveys of indoor localisation techniques [20] [16] have been used.

Table 2.1 presents the most relevant indoor localisation methods and rates each method according to various criteria. It includes the accuracy of localisation in meters, the devices needed for deployment, and the data flow. This table will give valuable insight to identify key areas for improvement. The inclusion of BLE guarantees that the methods mentioned in the table are extremely scalable in terms of the number of devices, which is why they are not included in the table.

2.3.2 Additional Insights

This section offers additional insights derived from the related literature. Certain indoor localisation techniques exhibit promise when integrated with machine learning, as previously discussed. The convergence of machine learning with BLE microcontrollers presents an intriguing avenue due to their low energy consumption and adaptability to local contexts, a concept often referred to as Edge AI [28], extensively explored in the literature. For further exploration, interested readers can refer to my preliminary survey report [37].

Moreover, the potential of indoor localisation methods in conjunction with machine learning extends to federated learning approaches. In such schemes, training or inference tasks can be distributed across nodes, which collectively contribute to decision-making or update a global model. Federated learning, a

growing area of research [17, 27, 22, 1], holds promise for applications such as fingerprinting methods [10]. Notably, Alsmadi et al. [4] propose employing a Filtered RSSI and Beacon Weights approach, advocating the initial filtering of RSSI data with a Kalman filter, while Wang et al. [38] advocate for Gaussian and Bootstrap filters on the RSSI. Additional techniques such as Gaussian and Gray filters are also explored. Astafiev et al. [5] demonstrate the superior performance of a feedforward neural network over an exponential approximation function with a coefficient for RSSI to distance conversion.

Innovatively, Naghdi et al. [31] augment RSSI fingerprinting with additional features extracted from vision devices, suggesting that integrating diverse data sources can enhance accuracy. Similarly, researchers like Guidara et al. [18, 39, 19] incorporate environmental data such as humidity and temperature to augment their models. Various feature extraction methods including FFT and Recursive CWT are also explored. Jianyong et al. [24] propose a weighted sliding window approach for RSSI values and advocate for offline training coupled with periodic online learning, particularly effective with Taylor series expansion-based cooperative localisation, a finding with potential implications for machine learning models.

In addition, strategies such as interpolating RSSI data during training and employing rolling windows to extract additional features from time-series data are suggested in the literature. Additionally, the utilisation of multichannel RSSI for noise reduction represents another avenue for exploration in enhancing localisation techniques.

2.4 Problem Statement

As discussed in Section 2.3, significant progress has been made in achieving high accuracy with indoor localisation. However, certain regions can still be improved, such as the current centralised techniques, the use of batteries, and techniques that do not take local environmental influences, like humidity, temperature and reflections, into account.

To address the first limitation, this research will focus on decentralising the localisation of nodes in a network. A new indoor localisation technique will be developed, the results of which can later be used by other techniques or combined in a federated way. The accuracy of this spatial mapping model will be compared with existing methods. The primary research objective of this thesis is to investigate whether Bluetooth-based indoor localisation models can be improved using federated algorithms.

Consequently, the primary research question is:

Can a federated approach improve Bluetooth-based indoor localisation models at The Edge?

Additionally, this research aims to evaluate whether this federated approach can be improved by using machine learning at the Edge and incorporating local environment data.

2.5 Contributions

This work represents an advancement in indoor localisation and federated learning by introducing a new federated localisation technique that runs on battery-free devices. The contributions of this work are as follows:

- **A new federated spatial mapping model:** This algorithm can be deployed on BLE edge devices within a network. It overcomes the limitations of traditional centralised localisation methods by allowing edge devices to collectively contribute and refine their spatial map while maintaining privacy and minimising data transfer.
- **A new localisation framework on the Edge:** The model/algorithm serves as a framework for other localisation or machine learning models. The Edge framework can provide low latency as a result of edge computing, making it ideal for applications that require local inference and machine learning. The coordinate map can be used as a location reference for other localisation methods.

2.6 Challenges

The following challenges will be expected when developing the federated localisation algorithm:

- **Ensuring a Stable RSSI-to-Distance Model:** The entire algorithm relies on maintaining a stable relationship between the Received Signal Strength Indicator (RSSI) and distance. Fluctuations in signal strength due to noise can lead to cumulative errors, emphasising the need for a robust model.
- **Handling Asynchronous Communication:** The potential for deadlocks arises from asynchronous communication between nodes. Information dependencies for further calculations may create challenges, necessitating careful consideration of communication protocols and state machine logic to avoid bottlenecks.
- **Addressing 3D Spatial Mapping:** The structure's orientation is intrinsically tied to the perspective of itself towards another node on the x-axis. The lack of predefined alignment poses a challenge. A mechanism must be devised to aggregate spatial structures into a global reference frame.
- **Neural network for the RSSI-to-Distance Model:** Implementing a neural network for the RSSI-to-distance model requires careful consideration of memory constraints on the BLE test hardware platform, ensuring that the model fits within the available resources.
- **Filtering of Triangle and Tetrahedron structures:** It may be essential to implement effective filtering mechanisms to ensure optimal network coverage during the construction of triangles and tetrahedrons.

Chapter 3

Federated Localisation Approach

This chapter will outline the proposed method for addressing the research question. Initially, the assumptions and specifications of the network structure will be examined and summarised. Next, the development of the mapping algorithm will be described, which will incorporate the same principles as the simulator, with an additional emphasis on node communication. Finally, the approach to implementing the simulator will be discussed, serving as a proof-of-concept for implementing the actual algorithm.

3.1 Network Specifications and Assumptions

For this research, a BLE network will be analysed and used to implement the federated mapping algorithm. This section will describe this network's assumptions and specifications in more detail. Table 3.1 provides a quick reference to the assumptions and specifications.

The network structures that will be examined consist of 3 to 12 nodes at different heights. The minimum requirement of three nodes is due to the algorithm's aim of creating triangle structures. Each node is equipped with an nRF52832 SoC, which has sufficient computational power to execute complex sensor algorithms in parallel with low latency Bluetooth Low Energy. The distance between nodes will range from 1 to 15 meters, but each node will prioritise establishing connections with its best neighbours to form a structure. As a result, the average distance between the nodes will be approximately 4 m. While some internode links may have a line-of-sight (LOS), the majority will be non-line-of-sight (NLoS) due to the surrounding environment. The network environment will be an office space with desks and plants scattered between the nodes. However, the simulation will assume an ideal environment with LoS connections and no noise to easily verify the algorithms' calculations.

Table 3.1: **Network assumptions and model specifications**

# (static) Nodes	3 - 12	**SoC**	nRF52832
Max RSSI	-95 dBm	**Micro Controller**	64MHz Cortex-M4
Links	LoS, NLoS	**Bluetooth**	v5.4
Distance	1-15 M	**Protocol**	BLE, mesh
Environment	Office, Simulation	**Flash/RAM**	512 KB, 64 KB
Tx power	4 dBm	**Antenna**	Single

3.2 Spatial Network Mapping

When locating a person, node or tag within a network, a map is often used to represent the estimated location of the node. The objective of the spatial network mapping algorithm is to create a three-dimensional reference coordinate map or structure of its surrounding nodes. The only data source that can be used on a node in a BLE network is the RSSI, which the algorithm relies solely on. By communicating with neighbouring nodes, RSSI values can be retrieved for the algorithm.

The spatial mapping algorithm is derived from trilateration, where three anchor points determine the position of the fourth node. However, this approach adds a twist by using combinations of triangle nodes which consist of the best-connected surrounding nodes. Valuable information can be gathered by also using triangulation and considering various characteristics of these triangles, such as angles, altitude, altitude x position, and area. This information is used to map the triangle structures in a 3D space, allowing for accurate spatial representation. Other methods can later use this spatial representation as a location reference in a federated way. The structure will always be built from a self-perspective over the x-axis. As a result of the use of both trilateration and triangulation, the structure becomes overdetermined. Each node should ideally see its neighbour with the same RSSI as the neighbour sees him; this will be checked. Additionally, the longest side of the triangle formed by the nodes should always be longer than the sum of the other two sides, as per the Pythagorean theorem.

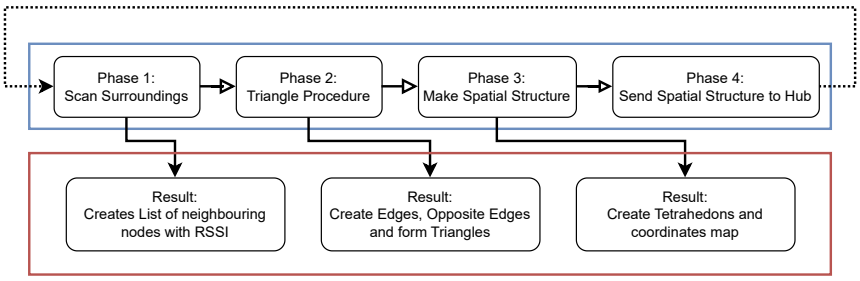


Figure 3.1: **High-level overview of the spatial network mapping algorithm's four phases that each node will walk through.**

The spatial network mapping algorithm deployed on a node performs four phases, each with its procedures. The four phases and the outcomes of each phase are illustrated in Figure 3.1. The four phases of each node are:

1. Scanning its surroundings, creating a list of neighbouring nodes and their RSSI values.

2. Form edges with its neighbouring nodes, creating opposite edges between its edges and ultimately forming the triangles themselves.

3. Creating a spatial structure by mapping the triangles around a specific base edge and forming tetrahedrons. Additionally, it generates a coordinate map of all the nodes with which edges are formed.

4. The coordinate maps are transmitted to the Hub for additional processing.

More details on the implementation of the spatial network mapping algorithm can be found in Chapter 4 Section 4.2.

3.3 Simulator Approach

The spatial mapping algorithm will be initially developed in Python for ease of development due to the absence of memory constraints. A simulator will be designed to simulate multiple parallel nodes in an ideal environment with LoS and a noise-free RSSI. The simulator will have a predefined network that will serve as a reference for the signal strength between the nodes. Each node in the simulation will have access to this information, making them all knowledgeable. As a result, there is no requirement for communication logic within the simulator, as it is primarily intended to demonstrate the mapping method.

The objective of the simulator is to determine whether a simulated node can reconstruct the predefined networks' spatial structure solely based on the received signal strength indicator (RSSI) of its surrounding nodes. The output of each simulated node will be a coordinate map of all nodes in the predefined network. The simulator will be evaluated with multiple tests that check the implemented maths functions. The predefined network will also be evaluated by comparing the coordinates with the output coordinate map. Further details about the implementation of the simulator can be found in Chapter 4 Section 4.1.

The code of the simulator will later be converted into the spatial network mapping algorithm written in C++. This conversion aims to make the algorithm compatible to run on the designated BLE hardware platform named *Crownstone* [12]. This hardware platform includes an nRF52 series BLE chipset, a switch and some additional sensors. The reason for using this platform is the open-source nature of its firmware, named *BlueNet* [11], which can be modified to our needs.

Chapter 4

Implementation

This chapter presents the implementation of the federated algorithm for spatial network mapping as detailed in Chapter 3. The execution will be segmented into the algorithm simulation tool in Section 4.1, and the federated spatial mapping algorithm that functions on the nodes in Section 4.2. These sections will include detailed explanations of the algorithms, pseudocode, and flowcharts. In addition, the chapter will outline the test configuration and the steps involved in validating the algorithm.

4.1 Simulator

This section presents the simulator designed for the algorithm implemented in Python within a Jupyter Notebook. The purpose of this simulator is to facilitate the testing, debugging, and validation of the algorithm's mathematical aspects, as well as to provide visualisation capabilities. The simulator is intended to replicate a network of BLE nodes, using a predefined network to establish RSSI strength relationships between nodes. Consequently, each node can ascertain the signal strengths between the other nodes and itself. The simulation of the nodes will occur concurrently through the use of multithreading. The simulator's code closely resembles the actual implementation, hence certain fundamental concepts will not be elaborated on here but in Section 4.2. This section will commence with a segment detailing the setup and execution, followed by a broad overview of the main routine of the simulator's algorithm. This section concludes with a discussion of the key distinctions between simulation and implementation of the real algorithm.

4.1.1 Setup and Run

The simulator consists of two steps: the setup and the run, Figure 4.1. The objective of the setup is to establish a reference network that all simulated nodes can utilise to obtain RSSI values. The configuration will receive a predefined list of coordinates embedded in the code. These coordinates will be associated with a node ID and transformed into a reference list (`RefNodeCoords`) with `Node()` pointers. The result is a list of `Node` objects with a specific node ID and coordinates. Using this list, the configuration will determine network con-

nections by computing the Euclidean distance between nodes and translating it into RSSI values. The result of the configuration will be a dictionary called `NetworkEdges`, which will include all edges for each `Node` object sorted according to RSSI. This dictionary will be utilised in the multi-threading run step, where each thread can access this object reference list of network edges. In this way, each `Node` object running in a thread will be all-knowing and easily get information from other `Node` objects. Utilising multi-threading allows the simulator to run the algorithm concurrently on multiple nodes. Each thread will go through the steps and stages of the algorithm, saving the information in its `Node` object for further analysis and debugging purposes.

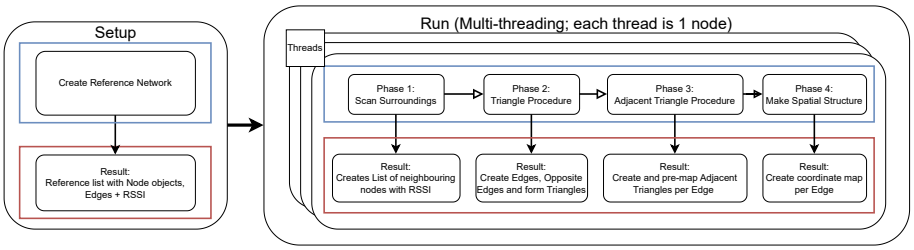


Figure 4.1: **High-level overview of the simulator's setup and four main phases that each node-thread will walk through.**

4.1.2 Main Loop

The main routine of a simulated node, which will be threaded, has three out of four phases as mentioned in the high-level overview in Section 3.2. The other phase differs because there will be no sending to the Hub. The `Make Spatial Structure` phase is moved to phase four. The third phase will be the new additional phase `Adjacent Procedure`; see Figure 4.1. Upon execution, the main function of each node object begins by retrieving the neighbouring nodes from the `NetworkEdges` and storing them into a list of surrounding nodes (`surNodes`). It then proceeds to the `triangleProcedure`, where it will iterate over all surrounding node combinations, creating edges, opposite edges and triangles for the `adjEdgeList`, `oppEdgeList` and `triangleList` respectively.

Subsequently, the recently introduced `Adjacent Procedure` first establishes a dictionary that links edges to a set of adjacent triangles through the `createAdj Triangles` function. This is achieved by iterating over both edges and triangles to identify common edges. Upon finding a match, an "Adjacent" object is generated and included in the dictionary associated with the corresponding edge. Following this, the `mapAdjacent` function sequentially processes the dictionary of adjacent triangles for each edge. The initial adjacent triangle serves as the reference triangle, and its "third node" position is calculated with the altitude and its x-axis position. Subsequently, the remaining adjacent triangles for the same edge are handled. Their altitude, x position, and the angle between the reference triangle (base triangle) and themselves are computed using the relevant functions. These results are then used to calculate the mapped coordinate. All the computed values are stored within the respective `Adjacent` objects. The final stage involves creating a coherent spatial map of the previously computed adjacent triangles. The `makeMap` function is similar to the `createCoordMap`

function, which will be elaborated on in Section 4.2.7. The `makeMap` function utilises the precalculated mapping details from the `Adjacent` objects for mapping.

4.1.3 Differences

This section focuses on highlighting the distinctions between the simulator and the actual implementation. As previously explained, the primary divergence lies in the phases and procedural logic within the simulator's main loop. This disparity arises from the omniscient nature of the simulated nodes, eliminating the necessity for communication between node threads. Moreover, as previously stated, the lack of a central Hub for data transmission also impacts the design of the simulator. Throughout the development of the real implementation, numerous adjustments and optimisations have been incorporated into the logic and data flow.

Some of the notable changes are summarised below. For a more in-depth understanding and additional details, please refer to Section 4.2 in the documentation:

- **Main routine:** The procedural nature of the simulator's main routine has been transformed into a state machine. This modification enhances flow control and aligns with the firmware's built-in coroutine throttling mechanism.
- **Non-ideal:** The actual implementation does not rely on an ideal environment; therefore, margins were introduced to enable continued progress in subsequent calculations. In addition, numerous edge cases have been addressed. For instance, when creating a spatial map, the algorithm checks for distance, considering scenarios where the node might lack a specific edge or its opposite. In such cases, the node cannot map the triangle and needs to skip it.
- **Communication:** addressing the asynchronous nature of the algorithm communication, a specialised logic approach has been adopted. This involves integrating a state machine into the triangle procedures to handle the transmission, waiting, and processing of messages.
- **Hub synchronisation:** This additional phase was introduced to transmit all data collected and generated from the algorithm to the hub for aggregation and subsequent processing.
- **Computation resources:** Due to the substantial difference in computing power and memory between the simulator and the actual test hardware, the simulator is tasked with calculating and generating a spatial map for every edge. However, in the real implementation, this process is adjusted so that it only constructs a spatial map for the most optimal connected edge.

4.2 Federated Spatial Mapping Algorithm

This section delves into the operational aspects of the federated spatial mapping algorithm. It discusses key elements such as the Log-distance path loss model,

Table 4.1: **Spatial network mapping algorithm specifications**

Max_Surrounding nodes	15	**Sample duration**	10s
Max_Edges	7	**Algorithm interval**	20 min.
Max_Opposite Edges	13	**RSSI deviation**	20%
Max_Triangle	13	**Heartbeat timeout**	5s
Max_Tetrahedrons	5	**Request timeout**	10s

the primary state machine, inter-node communication protocols, and functions associated with triangles. Furthermore, it explains the procedural steps for triangle construction, dihedral angle calculations, and the establishment of topology. Specific details regarding variable specifications for the algorithm's fundamental procedures are presented in Table 4.1.

4.2.1 Log-distance path loss model

The function `rssiToDistance` algorithm currently employs the log-distance path loss model to convert RSSI to distance. Although this model 4.1 is widely used, it faces certain limitations when implemented in a dynamic environment. The parameter values of this model are based on the NIST PAP02-Task 6 office model [30], but are fine-tuned to fit our test environment. The reason for using this model presently is that the neural network is too large to fit the memory on a BLE node. Nevertheless, a neural network could have been established, as mentioned in Appendix D, and trained with mean RSSI, variance, standard deviation, temperature, and humidity being utilised as input.

The Log-Distance Path Loss Model is given by:

$$PL_d = PL_0 + 10(n_0)\log_{10}\left(\frac{d}{d_0}\right) + X \tag{4.1}$$

where:

PL_d is the path loss at distance$d$,
PL_0 is the path loss (at ref. distance of 1m), which is tuned to **45.8 dBm**
n_0 is the path loss exponent, which is tuned to **4.5**
X is a normal random variable representing the shadowing effect.

The RSSI to distance function will be calculated with (4.2) and a plot can be seen in Figure 4.2.

$$Distance = 10^{\left(\frac{-PL_d - PL_0}{10 \cdot n_0}\right)} \tag{4.2}$$

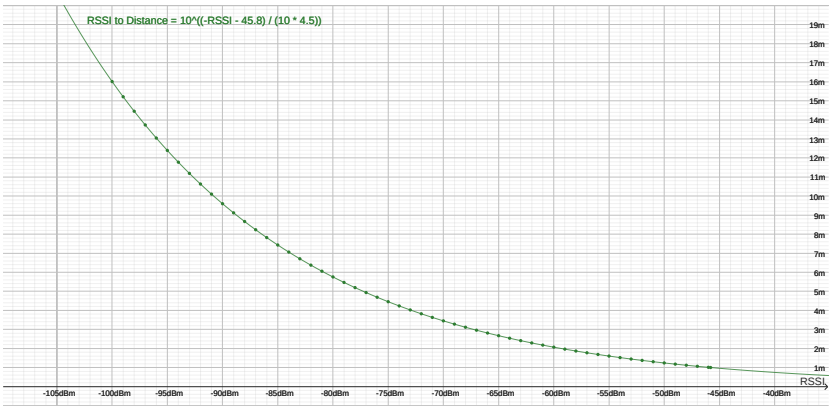


Figure 4.2: **Plot of RSSI to distance, calibrated for an RSSI 45.8 dBm at 1 meter.**

4.2.2 Main State Machine

The algorithm operates through a state machine that comprises six distinct states, as illustrated in Figure 4.3. Within each state, the state machine can be reset or restarted using the `RESET_TOPOLOGY` and `RESTART_TOPOLOGY` flags, respectively. The reset flag is triggered by a special node message ID designed for debugging purposes, while the restart flag is activated by an internal timer that runs every 20 minutes to ensure that all nodes are synchronised and to obtain new results.

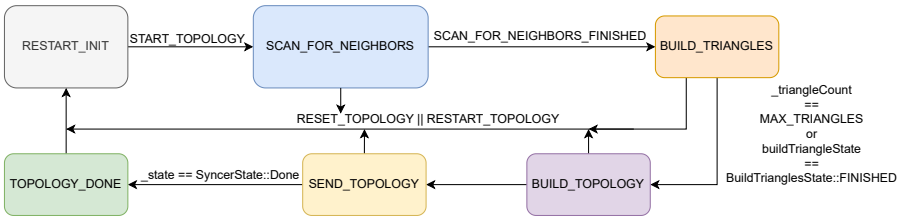


Figure 4.3: **Main state machine diagram of the spatial network mapping algorithm.**

The entire process begins anew from the `RESTART_INIT` state when the state machine is reset. Once the `START_TOPOLOGY` flag is set, which is also a special node message ID for debugging purposes, each node enters the `SCAN_FOR_NEIGHBOURS` state. During this phase, the node listens to BLE beacons and mesh messages to search for neighbouring nodes. Each neighbour message received adds a new entry or updates the entry's values in the surrounding node list, as explained in more detail in Section 4.2.3. The scanning phase lasts only 10 seconds to capture enough messages for an RSSI with less periodic noise. Once the timer runs out, the process of building edges, opposite edges, and triangles begins in the `BUILD_TRIANGLE` state. This is a separate state machine that is discussed in more detail in Section 4.2.5. The `BUILD_TRIANGLE` state is deemed complete when the number of built triangles (`_triangleCount`) reaches

the maximum number of triangles (MAX_TRIANGLES) or when the triangle procedure state machine reaches its finished state.

In the BUILD_TOPOLOGY state, the formed triangles are mapped into a spatial structure around the best edge, and the resulting three-dimensional coordinates are stored in a coordinate map. Finally, all the resulting coordinates and network information are transmitted and synced to the local hub in the SEND_TOPOLOGY state. The state machine enters its rest state (TOPOLOGY_DONE) when it reaches the end of the coordinate map.

4.2.3 Inter-Node Communication

The algorithm incorporates mesh communication between the nodes to compare the stored measured RSSI values, enabling the creation of edges and opposite edges. Additionally, the algorithm sends the coordinate map and other network data, such as the build edges, opposite edges, and triangles, to a local hub for further processing. To facilitate this, the algorithm introduces two new message types: node_request, which is a BLE mesh message, and topology_map, which is a direct BLE message. Each message type requires specific protocols for transmission and reception. This section will delve into the procedures for sending and receiving these messages, along with the mechanism employed by the algorithm to acquire neighbouring nodes' measured RSSI values.

BLE Beacon and mesh messages

On the left side of Figure 4.4, there is a high-level communication diagram illustrating the beacon and mesh messages. The diagram also indicates that the list of neighbouring nodes will be sorted according to the RSSI value. When a BLE message is received, it undergoes internal processing within the firmware. Subsequently, it is treated as an event by the algorithm. The algorithm then utilises the handleEvent function to determine the event type. If it is a device-scanned event, it signifies the reception of a beacon from a neighbouring node. The handleScannedDevice function is responsible for identifying the node and adding it to the list of neighbouring nodes. This list retains all neighbouring nodes as long as a heartbeat message is received on time for each entry; otherwise, the entry is removed from the list. If the event is identified as a received mesh message, it is forwarded to the onMeshMsg function. This function examines the packet type and selects the appropriate function to parse the packet accordingly. Additionally, any mesh message that is received, not relayed, and possesses a valid MAC address will also be included in the list of neighbouring nodes. The function addSurNode is responsible for adding nodes to the list. When a node is added, its ID and the RSSI of the last hop are included in the list, provided that the node does not already exist in the list. In case the node is already present in the list, the RSSI value gets updated by computing the average RSSI of all the previously received messages along with the new RSSI value. To achieve this, an aggregator class is employed, which computes not only the mean RSSI but also the variance and standard deviation.

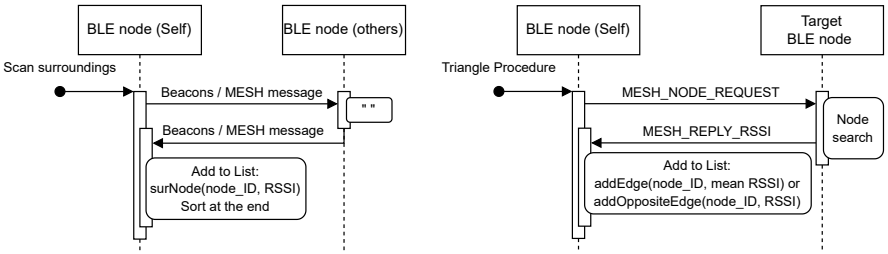


Figure 4.4: **Communication flow of Beacons and mesh messages (left) and edge forming negotiation (right).**

Node Request Messages

The `node_request` message is processed by the `onNodeRequest` function. The function first checks whether the message is an incoming request or a reply. This is necessary because the `node_request` type is used for both purposes. If the message is an incoming request, it will be handled by creating and sending a reply message of the same type. This reply message will contain the `targetId` and the corresponding RSSI value obtained from the surrounding node list. On the right side of Figure 4.4, the received incoming reply message can be processed in two different ways. The processing depends on the value of the packet's `targetId` parameter. An edge will be created if the `targetId` is the same as its node ID. In this case, the message is parsed and an edge is added to a list. On the other hand, if the `targetId` is different from its node ID, it indicates that an opposite edge is being formed. Again, the message is parsed and an opposite edge is added to the list.

For both types of edges, there is an additional step before adding them to the list. For a normal edge, the received RSSI value will be compared to the surrounding list value; if the deviation is good, it will be added to the edge list. If the edge already exists in the list, the RSSI value will be updated by taking the mean of the old and new RSSI values. The node request and its corresponding RSSI comparison will serve together as an extra validation for measured RSSI. For the opposite edges, the first reply will add a new list entry if there is no RSSI error value. The second reply will update this same entry value by taking the mean of the RSSI and the distance. The RSSI values from both directions are not compared, due to the issue that sometimes only one side replies, but both should see each other. If the second responded RSSI is equal to `RSSI_ERROR`, this indicates that the requested node does not have this target node and the possible specific entry will not be updated.

Each time a new opposite or normal edge is added to the list, it will increment its specific list counter. This counter keeps track of the number of edges in that particular list. The opposite list counter will be updated only when there is a new entry. In addition, the specific request counter associated with the edge will be decreased. The request counters are used to monitor the success of the creation and request processes.

Topology Messages

Following the completion of topology data gathering and calculation, the information is transmitted to the Hub. This process is facilitated by the `sendNext` function within the `MeshTopologySyncer` class. The `sendNext` utilises a state machine to iterate through four distinct payload transmission phases. Each time the `SEND_TOPOLOGY` state is reached, a new payload is generated and sent, and the state transitions to the next. The four packet categories and sending stages entail:

1. **Edges:** This packet includes details regarding all established edges, including the total number of edges and, for each edge, the target node ID and a compressed distance value.

2. **Opposites:** This packet contains information about the opposite edges created. It specifies the total number of opposite edges and for each one, its edge ID, which is a compression of the opposites' edge source and target ID, and a compressed distance value.

3. **Geometry:** This packet delves into the geometric structure of the mesh, specifically triangles and tetrahedrons. It includes the total count of both shapes and for each triangle, the compressed ID of its opposite edge. Additionally, a lookup table is created to efficiently reference triangles during tetrahedron data encoding. The information of each tetrahedron consists of two bytes, where the first combines the compressed IDs of its first two triangles, and the second holds the compressed ID of the third triangle.

4. **Coordmap:** This packet is responsible for conveying the coordinates of nodes in the mesh. It starts with the total number of coordinates and then iterates through pairs of consecutive coordinates, combining their compressed node IDs into a single byte. If there is an odd number of coordinates, the last node's ID is included separately. Finally, for each valid coordinate, its compressed x, y, and z distance values are added.

Moreover, the `sendNext` function encompasses two primary: packet creation (`createPacket`) and sending. The first step generates the message payload based on the current state. The aforementioned data are compressed and incorporated into the payload of the packet. The fabricated packet serves as the payload and is transmitted directly to the hub (not through other mesh nodes). This approach facilitates the transmission of a larger volume of data per message. Following each transmission, the state is updated for the subsequent topology message, and a random delay of approximately 10 seconds precedes the next transmission. Once all payload types have been sent, the topology data synchronisation with the hub is complete. All nodes will send their data to the same single hub, the hub's ID is defined in the `MeshTopologySyncer` class with the constant `HUB_ID`.

The receiving part of the topology messages, the `onTopologyMap` function, is responsible for handling the receiving end of topology messages. Incoming `TOPOLOGY_MAP` messages are converted to the appropriate payload message type based on their `TopologyMapType`. Subsequently, the message is forwarded over the UART to the Hub's Python parser code, which writes the received data into a database.

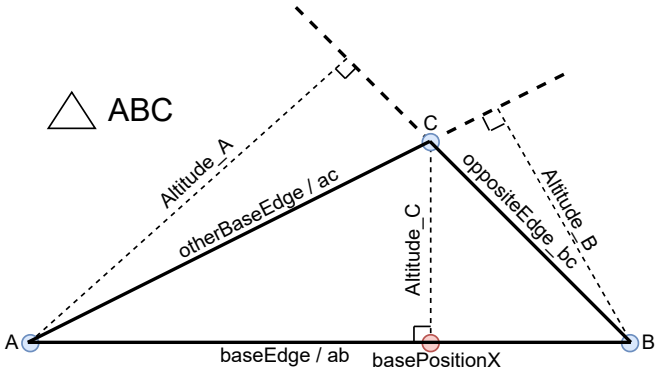


Figure 4.5: **Node triangle** *ABC*, **blue dots being nodes, with concerned altitude lines** *Altitude_A*, *Altitude_B*, *Altitude_C*. **The red dot being the intersection point of the** *altitude_C* **with the edge** *ab* **called** *basePositionX*.

4.2.4 Triangle Functions

By examining the distances between the three nodes of a triangle, a significant amount of information can be obtained, such as the area, altitude, and x-coordinate of the altitude. In the firmware, a `Triangle class` is implemented with various functions that include formulas for calculating the semiperimeter, area, altitude, and position of a triangle ABC with edges ab, ac, and bc, as shown in equations (4.3), (4.4), (4.5), and (4.6), respectively. A visual representation of the triangle can be seen in Figure 4.5. The algorithm that runs on each node assumes that the base edge of the triangle always lies on the x-axis and its node lies at the origin. In addition to the triangle math functions, there are also functions such as `getEdge`, `hasEdge`, `getOtherBase`, and `getThirdNode`.

$$\text{Semiperimeter (s)} = \frac{ab + ac + bc}{2} \tag{4.3}$$

The semiperimeter is used to calculate the area in 4.4.

$$\text{Area} = \sqrt{s \cdot (s - ab) \cdot (s - ac) \cdot (s - bc)} \tag{4.4}$$

$$\text{Altitude} = \frac{2.0 \cdot Area}{\text{ab} \parallel \text{ac} \parallel \text{bc}} \tag{4.5}$$

One remark for equation (4.6), depending on the angle ∠BAC of the ABC triangle, if it is greater than 90.0 degrees, the basePositionX should be interpreted as negative on the x-axis.

$$\text{basePositionX} = \sqrt{\text{otherBaseEdge.distance}^2 - \text{Altitude}^2} \tag{4.6}$$

4.2.5 Build Triangles Procedure

The triangle-building procedure aims to maximise the formation of triangles with their surrounding nodes. This process follows the state machine illustrated

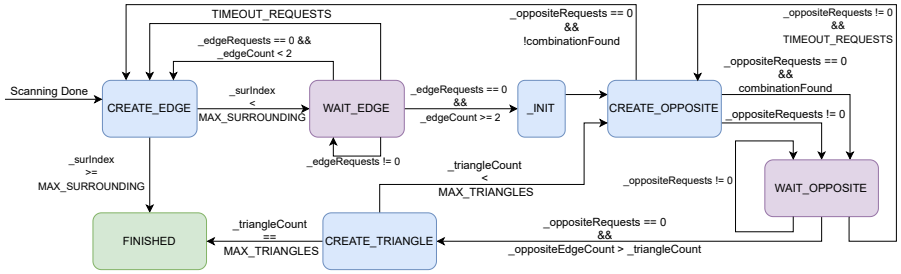


Figure 4.6: **State machine diagram of the build triangle procedure of forming edges, opposite edges and triangles.**

in Figure 4.6. The state machine initiates after the scan of the surrounding nodes is complete.

In the initial state, `CREATE_EDGE`, an edge-forming request is sent to the next entry in the surrounding node list each time this state is entered. The `_surIndex` keeps track of the next entry. After sending the request, a timer starts and the `_edgeRequests` counter is incremented. The state machine transitions to the `WAIT_EDGE` state, waiting for the reply. If an edge request fails, the request timer redirects the state machine to the creation state when it times out and raises the `TIMEOUT_REQUESTS` flag. The request counter (`_edgeRequests`) is decremented and the retry counter (`_edgeRetry`) is incremented. After three retries, the current target node is skipped by incrementing the sent index (`_surIndex`). If a reply has been handled successfully, the request counter (`_edgeRequests`) is zero. If there are at least two edges, it becomes possible to create an opposite edge.

The `CREATE_OPPOSITE` state will try to make opposite edges between all possible combinations of the already-made edges. New edge combinations will be initialised in the `_INIT` state each time a new edge is created. The algorithm will globally keep track of already handled edge combinations so that no unnecessary opposite edge requests will be sent. When entering the `CREATE_OPPOSITE` state, the algorithm attempts to create opposite edges for the current combination of edges. It first checks whether the maximum limit of opposite edges (`MAX_TRIANGLES`) has been reached. If this limit is exceeded, the state machine transitions to the `FINISHED` state, concluding the triangle construction process. Otherwise, the state enters a loop to iteratively generate new combinations until a valid one is found or all combinations are exhausted. If no new combination is discovered, the algorithm checks whether the maximum edge limit (`MAX_EDGES`) has been reached. If not, it transitions back to the `CREATE_EDGE` state; otherwise, it transitions to the `FINISHED` state, marking the completion of the overall triangle construction. If a valid combination is identified, opposite edge requests are dispatched based on the selected edges. The state machine transitions to the `WAIT_OPPOSITE` state, initiating a timer (`_requestTimer`) to handle timeouts during the opposite edge creation process. Sending the two requests will also increment the `_oppositeRequests` by two.

In the `WAIT_OPPOSITE` state, the algorithm oversees the waiting period for opposite edge request fulfilment and the re-sending. Upon successful receipt of both opposite edge requests, it resets the retry counter and transitions to

`CREATE_TRIANGLE` if all possible triangles are made; otherwise, it returns to `CREATE_OPPOSITE`. In case of no timeout, it continues waiting for replies. During a timeout, if the retry counter (`_oppositeRetry`) is less than two, the algorithm increments the retry counter. If two opposite requests fail, it decrements the request count (`_oppositeRequests`) and re-sends the requests. For a single failed request, it checks which one failed and compensates accordingly. After each resend the timer is re-initiated and the `TIMEOUT_REQUESTS` flag is reset. In case of two time-outs in a row, indicating excessive failures, it resets the retry and opposite request counters. It also compensates the `_oppositeEdgeCount` by the number of failures and returns to `CREATE_OPPOSITE` state.

In the `CREATE_TRIANGLE` state a triangle will be created using the newly formed opposite edge and its combination of edges. Each new triangle will increment the triangle counter (`_triangleCount`) only when the area is larger than 0.2, otherwise it will be discarded. If the `_triangleCount` reaches its maximum, then the state machine will transition to its final state `FINISHED`. If not, the state machine transitions back to the `CREATE_OPPOSITE` state to process the next edge combination.

4.2.6 Dihedral Angle

To determine the angle of the mapping or the angle between two triangular planes that share a common base edge, one must have the altitude heights, the difference in the x positions of the altitude lines and the distance between the third nodes. While the first two quantities can be computed using the methods outlined in Section 4.2.4, the distance must be obtained by inspecting the list of opposing edges. Consequently, the angle can only be computed when a triangle is established with the two third nodes of the input triangles. For example, the angle between the triangles ABC and ABD can be calculated as follows.

First, the delta x between the two altitude base positions will be calculated with (4.7).

$$\text{DeltaX} = abs(altitudeBasePostion\_ABC - altitudeBasePostion\_ABD) \tag{4.7}$$

With the delta X position and the distance between the two third nodes C and D, the projection distance perpendicular to the x-axis can be calculated with (4.8).

$$\text{Projection} = \sqrt{distance\_CD^2 - deltaX^2} \tag{4.8}$$

Next, the angle ratio of the newly formed triangle ACD can be calculated with (4.9). The visualisation of the projection and the triangle formed can be seen in Figure 4.7.

$$\text{Ratio [-1,1]} = \frac{altitude\_ABC^2 + altitude\_ABD^2 - projection^2}{2 \cdot altitude\_ABC \cdot altitude\_ABD} \tag{4.9}$$

At last, the triangle edge distances ratio is converted into degrees with (4.10).

$$\text{Dihedral angle} = \arccos(Ratio) * \frac{180}{\pi} \tag{4.10}$$

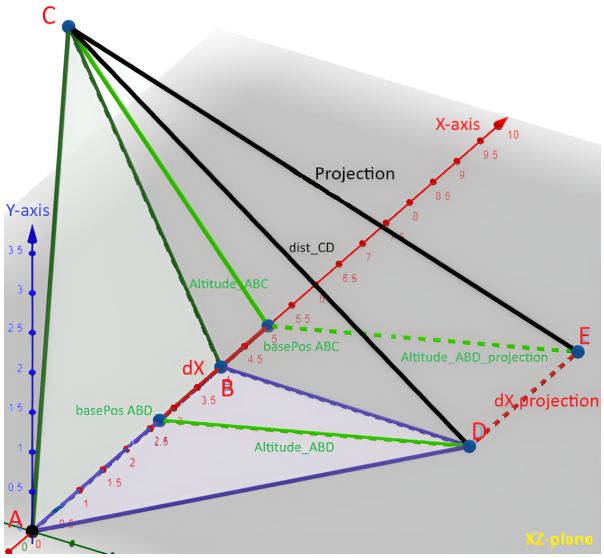


Figure 4.7: **Visualisation of dihedral angle calculation, $\angle C, basePosABC, E$, between two triangles planes (ABC and ABD) with the same base edge ab; Green lines are the altitude lines or its projection; Red lines are delta X between $basePosABD$ and $basePosABC$ or its projection; The black line $dist\_CD$ is a requested distance and the other is the translation of this distance called the $Projection$.**

4.2.7 Build Topology Procedure

The `createCoordMap` function is designed to establish a coordinate map (coordmap) for a mesh topology based on a specified base edge. The primary objective is to iteratively map triangles around a chosen base triangle (base edge) positioned along the x-axis. The mapping involves utilising calculated angles to establish their spatial relationships and applying transformations to align the triangles appropriately. The primary goal is to form tetrahedrons around the base triangle. A high-level flow diagram can be seen in Figure 4.8.

The function commences by initialising the relevant variables, the first-match indicator, and the last node ID. A special case is handled if the list of edges comprises only one edge. The function adds coordinates for the base edge and terminates. Subsequently, the function iterates through all triangles in the triangle list. For each triangle, it checks whether the specified base edge is present. If not, the triangle is skipped. Upon identifying the first triangle that matches the base edge, it is designated as the base triangle, and its coordinates are placed on the x-axis $(baseEdge.distance, 0, 0)$. The third node of this base triangle is located on the xy-plane $(altitudeBasePositon, altitude, 0)$. The coordinates are added to the coordmap with the `addCoord` function. The base triangle index, a pointer to the other base edge and the last mapped node values will also be updated.

For subsequent triangles, the `calculateMapAngle` function calculates the angle between the base triangle and the current triangle. The current triangle

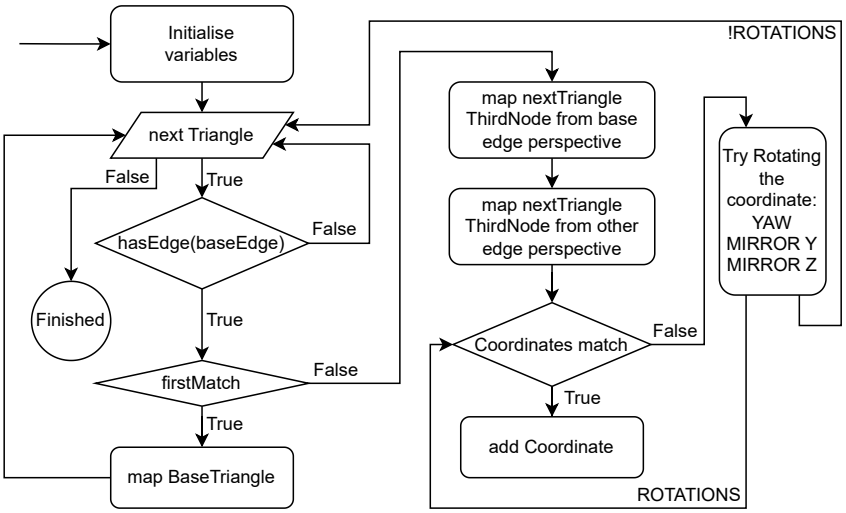


Figure 4.8: **High-level Flow diagram of making a coordinate map of the spatial structure.**

is then positioned on the xy plane and rotated accordingly onto the z-axis. The function then checks whether the mapped coordinates match the coordinates of a triangle connected to another base edge for verification. Various transformations and validity checks are performed on the mapped coordinates. Initially, it checks if the mapped coordinates already match those of the connected triangle. If a match is found, then it calculates the distance between the third node and the last node. The distance check will give valuable information about the orientation of the newly formed tetrahedron. If this distance is within an acceptable deviation, the coordinates are added, and a tetrahedron is formed. If no match is found in the initial orientation, then it explores possible alternative orientations by rotations and mirroring operations in Z and Y. The following rotations are used: yaw, $yaw + mirrorZ$, $mirrorY + yaw$, $mirrorY + yaw + mirrorZ$. The yaw rotation is the angle between the base edge and the other base edge, this will align the other base edge onto the x-axis. The mirror in Y compensates for the right-hand rotation direction when looking from the other perspective of the base edge. The mirror in Z is to compensate for the orientation in Z that was checked with the distance check.

If a match is identified after a rotation or combination, the corresponding coordinates are added. In addition, the last node is also updated and a tetrahedron is formed. A tetrahedron consists of three triangle pointers, four stone ids and six edge pointers. The stone ids and edge pointers are derived from the triangles and are used for the identification of the tetrahedron. The tetrahedron is included in a collection (`tetrahedronList`) and the counter (`_tetrahedronCount`) is increased. A criterion for determining when to include a tetrahedron may be incorporated at a later stage. Presently, all conceivable tetrahedrons are included.

The aforementioned process is carried out for every triangle in the list. If no appropriate match is found, the triangle will be disregarded and not included in the mapping.

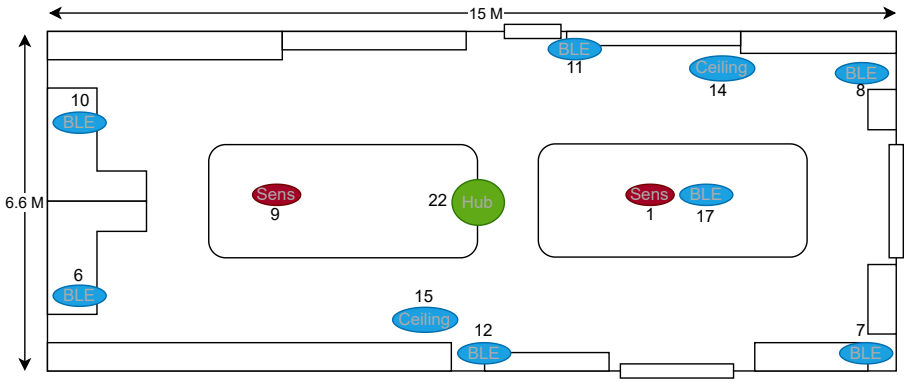


Figure 4.9: **The network test environment office contains nine normal BLE nodes (blue) from which two are placed in the ceiling to account for height differences. A single BLE hub (green) is positioned centrally to both transmit data to the database and function as a BLE node itself. Additionally, two BLE nodes (red) are modified to measure local data such as humidity. The number represents the node ID.**

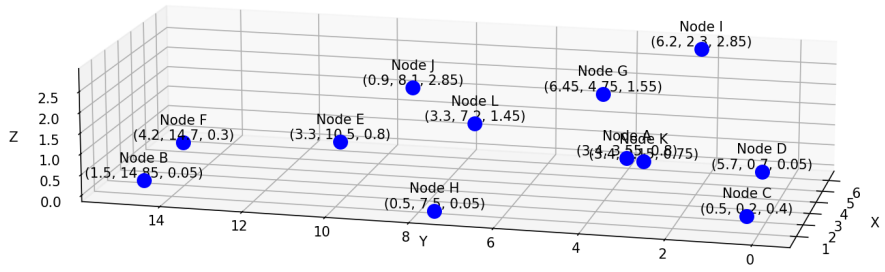


Figure 4.10: **Visualisation of the nodes in the test environment in 3D, see Table 4.2 for reference.**

4.3 Test Setup and Algorithm Verification

After developing the algorithm within the Python simulator, it will be tested on the Crownstone hardware platform. A test environment is built to evaluate the algorithm's operation in a real network of BLE nodes and to determine the accuracy.

The test setting depicted in Figure 4.9 illustrates an office layout featuring numerous cabinets positioned along the walls, with desks arranged in the central horizontal area. Within this office configuration, several BLE nodes are placed at predetermined coordinates, establishing a BLE-mesh network. The specific spatial positions of the nodes in the test configuration are identified by their corresponding coordinates as listed in Table 4.2. The 3D representation is visualised in Figure 4.10. The distances between the nodes are also calculated and can be seen in the distance matrix Figure 4.11

The algorithm has already been verified through the development of a simulator, which has shown that it works effectively. The spatial mapping algorithm

Table 4.2: **The coordinates of the twelve nodes in the test setup**

Node ID	Coordinate	Node ID	Coordinate
1 / A	(3.4, 3.55, 0.8)	11 / G	(6.45, 4.75, 1.55)
6 / B	(1.5, 14.85, 0.05)	12 / H	(0.5, 7.5, 0.05)
7 / C	(0.5, 0,2, 0.4)	14 / I	(6.2, 2.3, 2.85)
8 / D	(5.7, 0.7, 0.05)	15 / J	(0.9, 8.1, 2.85)
9 / E	(3.3, 10.5, 0.8)	17 / K	(3.4, 3.15, 0.75)
10 / F	(4.2, 14.7, 0.3)	22 / L	(3.3, 7.2, 1.45)

⇄	**1**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**14**	**15**	**17**	**22**
1	0	11.48	4.45	3.74	6.95	11.19	3.36	4.96	3.69	5.58	0.4	3.71
6	11.48	0	14.69	14.76	4.77	2.72	11.35	7.42	13.69	7.33	11.87	7.98
7	4.45	14.69	0	5.24	10.68	14.96	7.58	7.31	6.55	8.28	4.15	7.61
8	3.74	14.76	5.24	0	10.12	14.08	4.38	8.56	3.26	9.25	3.43	7.07
9	6.95	4.77	10.68	10.12	0	4.32	6.6	4.17	8.94	3.97	7.35	3.36
10	11.19	2.72	14.96	14.08	4.32	0	10.28	8.1	12.82	7.81	11.59	7.64
11	3.36	11.35	7.58	4.38	6.6	10.28	0	6.72	2.78	6.61	3.54	3.99
12	4.96	7.42	7.31	8.56	4.17	8.1	6.72	0	8.21	2.89	5.27	3.14
14	3.69	13.69	6.55	3.26	8.94	12.82	2.78	8.21	0	7.86	3.6	5.86
15	5.58	7.33	8.28	9.25	3.97	7.81	6.61	2.89	7.86	0	5.93	2.92
17	0.4	11.87	4.15	3.43	7.35	11.59	3.54	5.27	3.6	5.93	0	4.11
22	3.71	7.98	7.61	7.07	3.36	7.64	3.99	3.14	5.86	2.92	4.11	0

Figure 4.11: **Distance matrix of all the nodes of the test environment**

will undergo further testing by deploying it on each node in the test environment. To initiate the spatial mapping procedure, a start signal will be sent via Bluetooth mesh to all nodes in the BLE network, synchronising the start of the algorithm at each node. At the end of every iteration of the algorithm, all nodes will send the created coordinate map and other information about the structure of their surrounding nodes to the local hub. This entire process will be repeated every fifteen minutes. The results will then be stored in a database at the hub for data analysis and visualisation. The coordinate maps obtained will be compared to the built test environment coordinate map of the BLE nodes, and the accuracy or error will be calculated based on this comparison.

Chapter 5

Evaluation

This chapter will discuss the evaluation of the federated indoor localisation algorithm. After development, the algorithm is tested in our controlled test environment in Section 4.3. After that, the resulting data with the node localisation estimates were processed. The results of the simulation and the federated mapping algorithm can be found in Sections 5.1 and 5.2. Finally, the obtained results are compared with the related work in Section 5.3 to assess if any improvements in accuracy are made.

5.1 Simulation Results

As mentioned previously, the purpose of this simulator is to aid in testing, debugging, and validating the algorithm's mathematical aspects while also providing visualisation capabilities. A predefined network is used as input to test the simulator, which can be seen on the left side of Figure 5.1. The simulator generates a set of coordinates mapped around a specific base edge. The output coordinates are collected and used as a metric to compare with the input coordinates. Due to the ideal test environment in the simulation, there is no difference in results after each run.

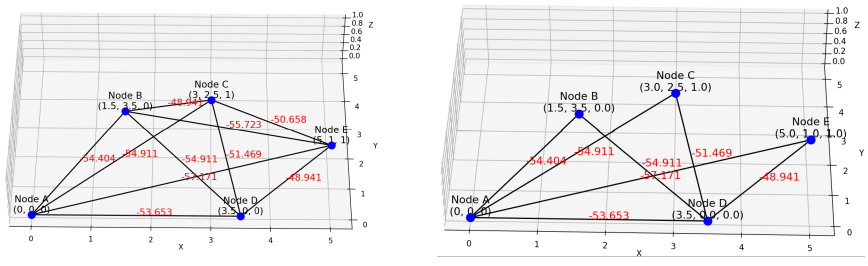


Figure 5.1: **Double tetrahedron input reference network for the simulator (left) and the output result (right).**

The simulator output is shown on the right side of Figure 5.1 and the collected coordinates can be seen in Table 5.1. It is obvious that only the base edge AD matches because the coordinates of node D are on the x-axis. The other coordinates of the other base edge represent the same network structure but

under a different angle. When rotating the configuration to align with the base edge AD, it becomes apparent that the spatial arrangement closely corresponds to its original configuration. The structure is indeed overdetermined from multiple perspectives in this simulator. The necessary rotations can be determined using the singular value decomposition method (SVD) in conjunction with the covariance matrix, see Appendix E.

Table 5.1: **Output of the simulator showing reference input and the output coordinates of the mapped nodes plotted around a specific base edge**

Nodes	Input coordinates	Output coordinates per base edge			
ID	Reference	AB	AC	AD	AE
A	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
B	(1.5, 3.5, 0)	(3.808, 0, 0)	(3.287, -1.417, 1.300)	(1.5, 3.5, 0)	(2.117, 3.165, 0)
C	(3, 2.5, 1)	(3.480, 1.956, -0.562)	(4.031, 0, 0)	(3, 2.5, 1)	(3.560, 1.805, 0.562)
D	(3.5, 0, 0)	(1.379, 3.130, 0.745)	(2.605, 2.338, 0)	(3.5, 0, 0)	(3.368, -0.594, -0.745)
E	(5, 1, 1)	(2.889, 4.319, 0)	(4.589, 2.372, -0.557)	(5, 1, 1)	(5.196, 0, 0)

5.2 Spatial Mapping Algorithm Results

During development, the debug output was printed to the UART. The results logs of the coordinate map procedure can be seen in Appendix F, these logs show two successful creations of one or multiple tetrahedrons and their coordinate map. This result initiated the real tests with the test setup. The algorithm's results are collected via the hub and stored in an Influx database. The gathered federated output coordinates are used to visualise the structure. Figure 5.2 shows two structures built in two different intervals of node 1. Due to the different node perspectives of build structures (rotations), the varying usage of node IDs in the structure and the noise, the coordinate data can not easily be compared with the input or aggregated into one structure. Therefore, the edge distance, RSSI, mean and standard deviation values will be used to make this method comparable and determine accuracy. The edge distances are also compared with the distances between the test setup coordinates to calculate the error.

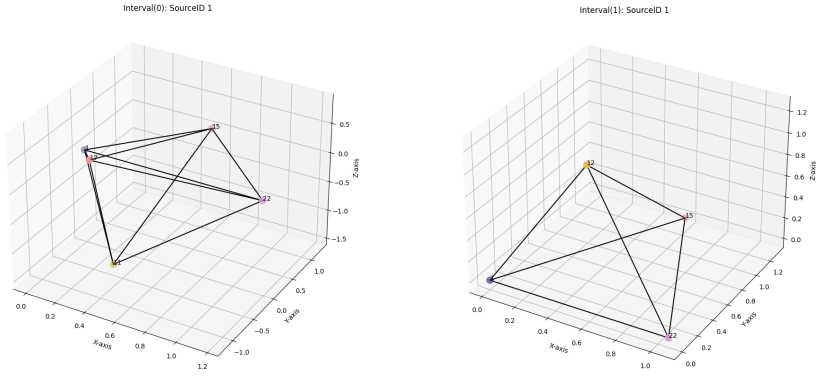


Figure 5.2: **Comparison of two coordinate map plots of node 1 on two different intervals. (Interval 0: node ID 1,11,15,17,22; Interval 1: 1,12,15,22)**

The coordinate maps of nodes 1, 11 and 12 can be seen in Figure 5.3. These nodes show the most consistency in their point clouds and show the best clustering of nodes after filtering. The algorithm demonstrates the ability to reproduce the same structure with a stable signal consistently. The consistency is probably due to the low variation in edge distances and the log-distance model, which allows the algorithm to match and make more triangles and tetrahedrons.

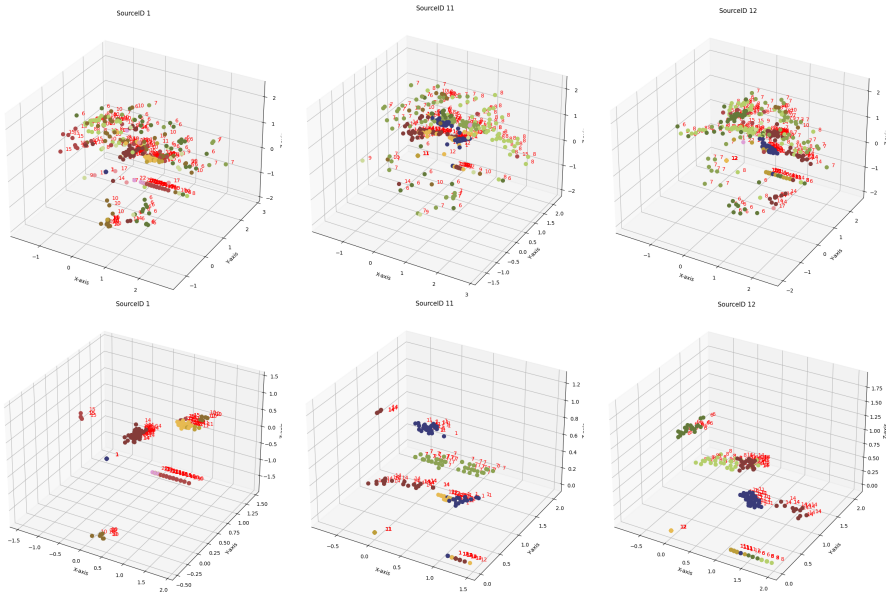


Figure 5.3: **Point clouds representing nodes 1, 11 and 12, showcasing their respective calculated coordinates across all intervals and filtered point clouds illustrating the consistency in coordinates for nodes 1, 11, and 12 across multiple intervals.**

RSSI values are extracted from the edge data and plotted in Figure 5.4 to see the behaviour of the signal over time. The RSSI values for each node show a

lot of variation. Therefore the signal is smoothed with a window of 20 intervals. The figure shows that almost all RSSI trends are around -55/60 dBm, except for node 1. A horizontal line means that no edge has been made with this node during this interval. Another thing that can be seen is that the RSSI values show sinusoidal movement over time, indicating periodic influences such as humidity or the presence of people.

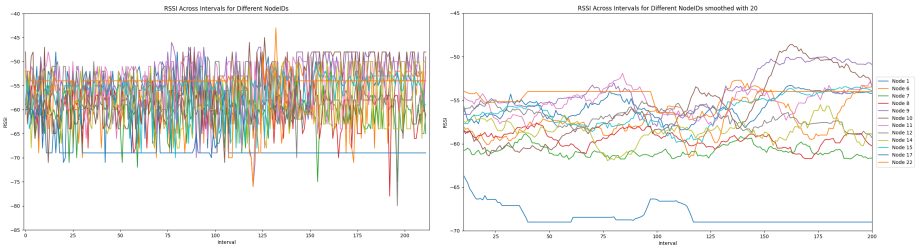


Figure 5.4: **Time series plot displaying RSSI values for each node. On the left, the unfiltered data is depicted, while on the right, the filtered data is presented.**

After statistically processing the data, the distribution of the edge data of each node is visualised with box plots for each `sourceID`, see Appendix G. The left figures show the distance distribution and the RSSI's right. In both figures the median is highlighted with the orange line and the green dotted line indicates the mean. For each `sourceID` the nodes are sorted according to the distance from left to right, the left being the closest to the specific `sourceID`. Figure 5.5 shows the most interesting results.

At `sourceID` 1, node 17 should be the closest in distance, but this is not the case. Possible explanations for this are that the antennas are not aimed well or that there are too many blockages between the nodes, which can lead to higher RSSI values. Nevertheless, checking its distribution does not show many outliers, and thus a fairly stable RSSI. Checking the distribution of `sourceID` 7 also shows a similar bad connection to node 1. `sourceID` 6, 7 and 10 have a lot of outliers visualised by the circles in their distributions. Probably because it is located further away from the other nodes. It can also be seen that `sourceID` 22, the hub, has a very narrow distribution for its nearby nodes. The nodes further away show much more variation. The exceptions are nodes 1 and 7 which suffer from ambient noise and blockages because it is in the corner and therefore show more variation.

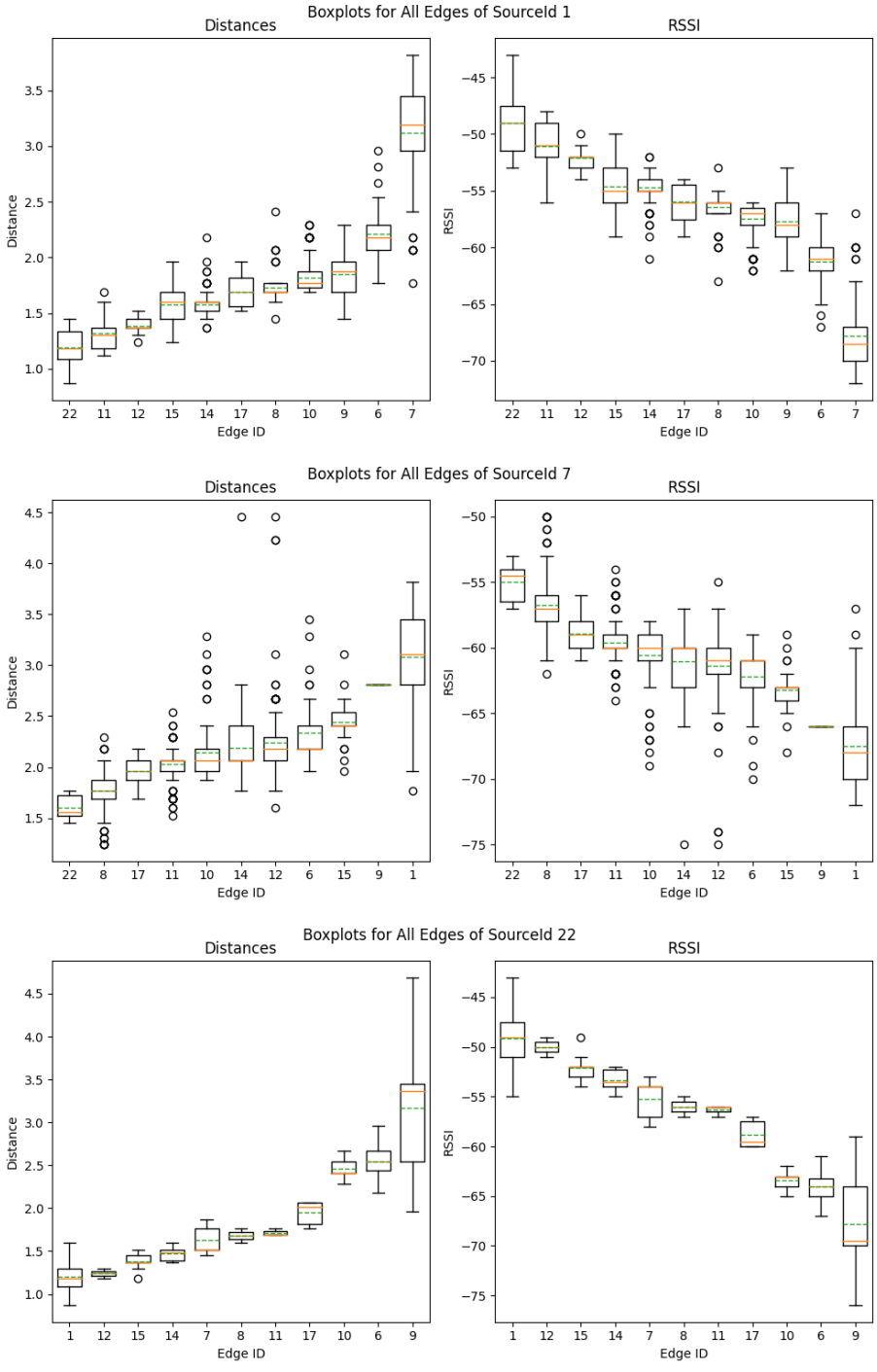


Figure 5.5: **Box-plots of edge statistics for distance and RSSI, nodes sorted on RSSI.**

The means and standard deviation of the distance and RSSI have also been

calculated for each edge target of a node. This statistical data can be seen in Table H.1 till H.6 in Appendix H. It can be seen that the mean RSSI values are all very close to each other, with an overall minimum of $-70dBm$ and a maximum of $-47.9dBm$. The small standard deviations for the distance are linked to a not adequately fine-tuned RSSI to distance formula. Small changes in RSSI should result in greater changes in distance when at a certain distance, because of the logarithmic function.

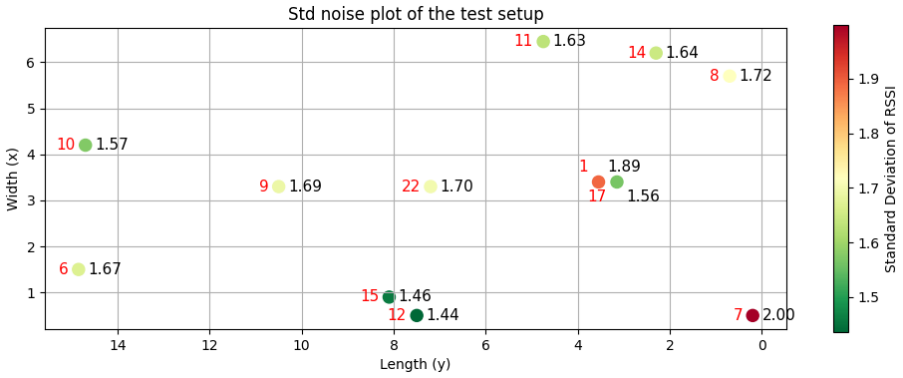


Figure 5.6: **2D map depicting the noise level at each node location within the test setup, showcasing the RSSI standard deviation in colour.**

The noise of each node is also calculated to show which node locations have more noise than others, Figure 5.6. The noise is indicated by the mean standard deviation of the RSSI and visualised with a colour scheme. It can be seen that nodes 1 and 7 have the most noise and nodes 12 and 15 are the best. When comparing these results with the environment or location of the node it can be noted that node 1 has too much ambient noise from its neighbour 17. Node 7 is in the corner of a kitchen that blocks or reflects too many signals. Node 8 lies on the ground, resulting in more blockages for the signal that needs to pass through more office furniture.

The edge distances obtained from the tests are compared with the official distance computed from the test setup coordinates. For each interval, the RMSE (root mean squared error) and the MAE (mean absolute error) are computed for each node across all its edges. The results are shown at the top of Figure I.1 in Appendix I. The largest difference between the lowest and highest RMSE and MAE values is 10.7 meters. Due to an unstable RSSI, the RMSE and MAE are also unstable. The flat lines in the figures are due to the interpolation of missing points. These results show that the RSSI to distance function is not accurate enough and that other variables are needed to compensate for the environment.

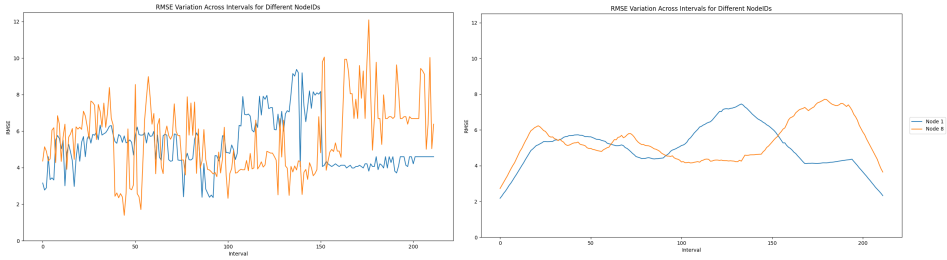


Figure 5.7: **Comparison of RMSE variations across intervals for different Node IDs, illustrating both the original and smoothed versions filtered with a 35-interval window. The left figure depicts the normal data, while the right shows the smoothed versions.**

From the results, two nodes are picked and smoothed with a 35-interval window and visualised at the right of Figures 5.7 and 5.8. It can be seen that a lot of the nodes' RMSE and MAE values reduce over time. Some plotted lines also show sinusoidal movement, which can be explained by the people working during the day causing more reflections and noise. The periodicness of the error could also be related to the variation in humidity over the day which is already mentioned in Appendix C. The other nodes are placed further away from the floor and roof probably resulting in less reflections and noise.

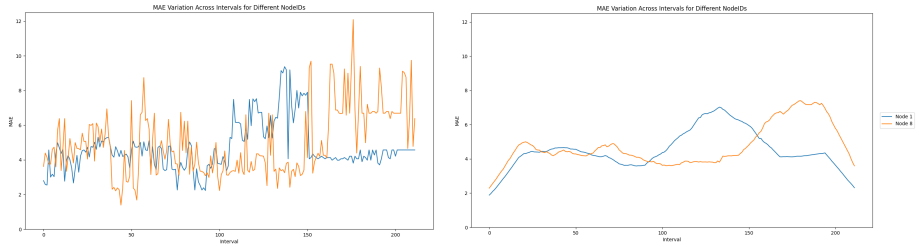


Figure 5.8: **Comparison of MAE variations across intervals for different Node IDs, illustrating both the original and smoothed versions filtered with a 35-interval window. The left figure depicts the normal data, while the right shows the smoothed versions.**

5.3 Comparison

Table 5.2 shows an overview of the current accuracies of various localisation techniques. It can be seen that the developed method does not have the same accuracy as the trilateration and triangulation techniques, due to noise and the lack of a noise estimate filter. Because of this, some of the nodes show better accuracy than others. Therefore, the federated spatial algorithm does not outperform the current best techniques, as some of the nodes have a significant difference in accuracy. However, it is important to note that the other methods are also not tested in our test setup, which can show a different comparison outcome.

Table 5.2: **Overview of relevant BLE indoor localisation techniques with accuracy**

Techniques/Methods	**Accuracy**	**Techniques/Methods**	**Accuracy**
Fingerprinting (FP.)(RSSI)	1.42m [9] [3]	*Time-of Arrival + RSSI*	2.0–2.7m [13]
Multi/Trilateration	0.45-1.85 m [33]	*Bloc: (CSI)*	0.86m [7]
Angle of Arrival	0.36m [8]	*ML (Classification, FP.)*	0.3-2.5m [21][36]
BLE landmark: DTW	0.42-0.75m [40]	*(New) Fed. Spatial Mapping*	4.62m

Chapter 6

Conclusions

Indoor localisation is a well-researched topic and it is challenging to improve the accuracy of existing techniques. However, utilising the federated approach in combination with local environment data and machine learning models has shown promising improvements in related studies. This thesis has focused specifically on a federated implementation for spatially mapping a network. The developed localisation algorithm allows for the creation of a 3D spatial reconstruction of the network in a federated way. This new approach decentralises indoor localisation and produces data from multiple perspectives, which can improve the aggregation of the spatial network structure. The structure coordinate map can be used as a reference for other localisation methods. The gathered data is also used to identify local noise factors.

The spatial mapping algorithm has first been tested with a simulator. Its results indicate that the maths behind the algorithm is correct for generating a perfect 3D structure in an ideal situation. However, the results of the real test show the real importance of a good RSSI to distance model and filter method. The outcome of the spatial mapping algorithm method indicates a significant variation in noise levels between nodes. During the investigation, it was discovered that factors like blockages and ambient noise contribute majorly to this variation. In addition, humidity was identified as another significant factor that affects noise levels. The results from the time series plot of RSSI measurement and other environmental data revealed that conducting measurements over multiple periods could provide new insights into noise levels. Although an RMSE ranging from 1 to 10.7 metres indicates high accuracy in some nodes. With an accuracy of 4.62 meters, it cannot be concluded that the method has a higher overall accuracy.

To conclude, the spatial mapping algorithm can create a 3D structure in a federated way and can serve as a localisation framework and location reference for further research with machine learning models. The developed framework also paves the way for running improved RSSI to distance conversion models at the Edge. Combining this method with other existing methods in a federated way could potentially improve accuracy by knowing the direction and relative locations of the nodes and other environmental factors. To address the research question, the federated approach can potentially enhance Bluetooth-based indoor localisation models. However, the spatial network mapping algorithm cannot exceed the localisation accuracy of current localisation techniques.

6.1 Limitations and Final Remarks

In this research, an algorithm was developed, but it has certain limitations. This section will discuss those limitations and provide some final remarks.

- As mentioned before, the federated spatial mapping Python simulator assumes all nodes are visible with fixed signal strength and no noise. However, the simulator may not account for certain edge cases, which can result in errors or unexpected behaviour. There is also no communication between the node threads.
- The federated spatial mapping implementation in C++ runs on designated BLE nodes and is integrated into the BlueNet firmware. It is currently limited to storing 15 surrounding nodes, 13 opposite edges and triangles, seven adjacent edges and five tetrahedrons. The number of triangles is fixed to 13 because this amount still fits in one hub sync message.
- As mentioned earlier, the noise level of the RSSI value is crucial for the success of the federated localisation algorithm. This means that the entire algorithm depends on a denoised RSSI value. If there is noise, the algorithms' accuracy will be affected and the usage of the system will be limited. A shorter sampling period, implementing new filter methods or using ML could improve the RSSI.
- Local environment data is also researched in this thesis. Some BLE nodes were equipped with sensors that measure humidity, temperature and pressure. This information can be shared across the network in a federated manner. Sensor data could have been used to improve the RSSI to distance model. However, due to memory constraints due to the usage of machine learning, this improvement has not been implemented in the firmware. The local environment data and machine learning research documentation can be found in Appendices C and D.

6.2 Future Work

The thesis presents a new approach to indoor localisation, which involves a federated framework for deploying models in a BLE mesh network. The algorithm and localisation techniques have been tested and the results were evaluated. However, the evaluation has revealed the potential for improvements and other future work, which will be addressed in this section.

- For this research, the BLE nodes used the SoC NRF52832. Upgrading the hardware of the BLE nodes would allow more resource-intensive machine-learning models to be run at the Edge. The new nRF series is a suitable option, based on its higher memory capacity and multicore.
- Additional information can be gathered to counteract external factors that may affect the accuracy of the localisation algorithm. Such data can be used as new features in ML models to improve their accuracy and robustness. Real-time data can be particularly useful in measuring factors such as humidity and temperature, which can influence the RSSI and create bias throughout the area. Other factors, such as walls and reflections, can be investigated more to improve accuracy.
- To improve the accuracy of indoor localisation, other localisation techniques could be combined with the spatial mapping algorithm. Information about the structure can be used to determine and adapt to noisy areas.
- Once the spatial network mapping algorithm has successfully run on a node, it has generated triangle and tetrahedron node structures, along with a coordinate map of the nodes from these structures for localisation purposes. For future work, a multilayer localisation mechanism can be developed. The first layer involves the node itself attempting to obtain a location estimate of the tracked unit per tetrahedron. The node can use trilateration, multilateration or a small neural net to localise the tag based on RSSI values between them and the tag. The second layer of the system combines the location predictions of each tetrahedron. This layer creates a final location prediction based on the output of the first layer and sends it to the local hub. To combine the nodes' results, an aggregation algorithm can be run on the hub.

Bibliography

[1] H. Gebreslasie Abreha, Mohammad Hayajneh, and M. Adel Serhani. Federated Learning in Edge Computing: A Systematic Survey. *Sensors*, 22(2), 2022.

[2] A.B. Adege, H.P. Lin, G.B. Tarekegn, Y.Y. Munaye, and L. Yen. An indoor and outdoor positioning using a hybrid of support vector machine and deep neural network algorithms. *Journal of Sensors*, 2018, 2018.

[3] H. Ai, S. Zhang, K. Tang, N. Li, W. Huang, and Y. Wang. Robust Low-Latency Indoor Localization Using Bluetooth Low Energy. In *ION 2019 Pacific PNT Meeting*, pages 58–72, 05 2019.

[4] L. Alsmadi, X. Kong, K. Sandrasegaran, and G. Fang. An Improved Indoor Positioning Accuracy Using Filtered RSSI and Beacon Weight. *IEEE Sensors Journal*, 21:18205–18213, 8 2021.

[5] A.V. Astafiev, D.V. Titov, A.L. Zhiznyakov, and A.A. Demidov. A method for mobile device positioning using a sensor network of BLE beacons, approximation of the RSSI value and artificial neural networks. *Computer Optics*, 45:277–285, 2021.

[6] D. Avellenada, D. Mendez, and G. lo Fortino. BLE-based Indoor Positioning Platform Utilizing Edge Tiny Machine Learning. In *2022 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 1–8. IEEE, 9 2022.

[7] R. Ayyalasomayajula, D. Vasisht, and D. Bharadia. BLoc: CSI-based accurate localization for BLE tags. *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pages 126–138, 12 2018.

[8] X. Qiu B. Wang, Y. Wang and Y. Shen. BLE Localization With Polarization Sensitive Array. *IEEE Wireless Communications Letters*, 10(5):1014–1017, 2021.

[9] B. Benaissa, F. Hendrichovsky, K. Yishida, M. Koppen, and P. Sincak. Phone Application for Indoor Localization Based on Ble Signal Fingerprint. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2018.

[10] Xin Cheng, Chuan Ma, Jun Li, Haiwei Song, Feng Shu, and Jiangzhou Wang. Federated Learning-Based Localization With Heterogeneous Fingerprint Database. *IEEE Wireless Communications Letters*, 11(7):1364–1368, 2022.

[11] Crownstone. Bluenet Firmware. `https://github.com/crownstone/bluenet`, 2023. Last accessed: Nov. 14, 2023.

[12] Crownstone. Crownstone Hardware. `https://crownstone.rocks/`, 2023. Last accessed: Nov. 14, 2023.

[13] D. Fontanelli D. Giovanelli, E. Farella and D. Macii. Bluetooth-Based Indoor Positioning Through ToF and RSSI Data Fusion. In *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, 2018.

[14] Eclipse. Eclipse Aidge platform. `https://gitlab.eclipse.org/eclipse/aidge/aidge`, 2023. Last accessed: Nov. 14, 2023.

[15] C. Ganguly, S. Nayak, S. Irene, A. Gupta, and K. Ch. UTILIZING MACHINE LEARNING ALGORITHMS FOR LOCALIZATION USING RSSI VALUES OF WIRELESS LAN. *ITU Journal on Future and Evolving Technologies*, 3(3), 6 2022.

[16] A. Garg and A. Gupta. Indoor tracking using BLE - brief survey of techniques. In *International Conference on Data Sciences & Machine Learning (ICDSML-2020)*, 03 2020.

[17] N.L. Giménez, M.M. Grau, R.P. Centelles, and F. Freitag. On-Device Training of Machine Learning Models on Microcontrollers with Federated Learning. *Electronics (Switzerland)*, 11, 2 2022.

[18] A. Guidara, G. Fersi, F. Derbel, and M.B. Jemaa. Impacts of Temperature and Humidity variations on RSSI in indoor Wireless Sensor Networks. In *Procedia Computer Science*, volume 126, pages 1072–1081. Elsevier B.V., 2018.

[19] A. Guidara, G. Fersi, M.B. Jemaa, and F. Derbel. A new deep learning-based distance and position estimation model for range-based indoor localization systems. *Ad Hoc Networks*, 114, 4 2021.

[20] S.J. Hayward, K. van Lopik, C. Hinde, and A.A. West. A Survey of Indoor Location Technologies, Techniques and Applications in Industry. *Internet of Things*, 20:100608, 2022.

[21] C. Jain, G.V.S. Sashank, N. Venkateswaran, and S. Markkandan. Low-cost BLE based Indoor Localization using RSSI Fingerprinting and Machine Learning. In *2021 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2021*, pages 363–367. Institute of Electrical and Electronics Engineers Inc., 3 2021.

[22] J.C. Jiang, B. Kantarci, S. Oktug, and T. Soyata. Federated learning in smart city sensing: Challenges and opportunities. *Sensors (Switzerland)*, 20:1–29, 9 2020.

[23] J.R. Jiang, H. Subakti, and H.S. Liang. Fingerprint feature extraction for indoor localization†. *Sensors*, 21, 8 2021.

[24] Z. Jianyong, L. Haiyong, C. Zili, and L. Zhaohui. RSSI based Bluetooth low energy indoor positioning. In *IPIN 2014 - 2014 International Conference on Indoor Positioning and Indoor Navigation*, pages 526–533. Institute of Electrical and Electronics Engineers Inc., 2014.

[25] A. H.M. Kamal, M.G.R. Alam, M.R. Hassan, T.S. Apon, and M.M. Hassan. Explainable indoor localization of BLE devices through RSSI using recursive continuous wavelet transformation and XGBoost classifier. *Future Generation Computer Systems*, 141:230–242, 4 2023.

[26] Keras. Keras API. `https://keras.io/`, 2023. Last accessed: Nov. 14, 2023.

[27] K. Kopparapu and E. Lin. TinyFedTL: Federated Transfer Learning on Tiny Devices. *CoRR*, abs/2110.01107, 2021.

[28] J. Lin, W.M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han. MCUNet: Tiny Deep Learning on IoT Devices. *CoRR*, abs/2007.10319, 2020.

[29] M. Masoodi, E.A. Sekehravani, and M. Maesoumi. RSSI-BASED MODIFIED K-NEAREST NEIGHBORS ALGORITHM FOR INDOOR TARGET TRACKING. *Far East Journal of Electronics and Communications*, 18:345–356, 3 2018.

[30] MathWorks. End-to-End Bluetooth LE PHY Simulation Using Path Loss Model, RF Impairments, and AWGN. `https://nl.mathworks.com/help/bluetooth/ug/end-to-end-bluetooth-le-phy-simulation-using-path-loss-model-rf-impairments-and-awgn.html`, 2023. Last accessed: Nov. 14, 2023.

[31] S. Naghdi and K. O'keefe. Combining Multichannel RSSI and Vision with Artificial Neural Networks to Improve BLE Trilateration. *Sensors*, 22, 6 2022.

[32] J. Paulavicius, S. Jardak, R. McConville, R. Piechocki, and R. Santos-Rodriguez. Temporal Self-Supervised Learning for RSSI-based Indoor Localization. In *IEEE International Conference on Communications*, volume 2022-May, pages 3046–3051. Institute of Electrical and Electronics Engineers Inc., 2022.

[33] S. Sadowski and P. Spachos. Optimization of BLE Beacon Density for RSSI-Based Indoor Localization. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2019.

[34] P. Sthapit, H.-Seon Gang, and J.-Young Pyun. Bluetooth Based Indoor Positioning Using Machine Learning Algorithms. In *2018 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 206–212. IEEE, 6 2018.

[35] H. Torii, S. Ibi, and S. Sampei. Indoor Positioning and Tracking by Multi-Point Observations of BLE Beacon Signal. In *2018 15th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–5. IEEE, 10 2018.

[36] K. Urano, K. Hiroi, T. Yonezawa, and N. Kawaguchi. Basic Study of BLE Indoor Localization using LSTM-based Neural Network (poster). In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 558–559. ACM, 6 2019.

[37] C. van Diepen. AI-chips Survey Report; Research and analysis of available AI accelerators chips; Edge AI. Technical report, Almende BV, 2023.

[38] J. Wang, J.G. Hwang, J. Peng, J. Park, and J.G. Park. Gaussian Filtered RSSI-based Indoor Localization in WLAN using Bootstrap Filter. In *2021 International Conference on Electronics, Information, and Communication, ICEIC 2021*. Institute of Electrical and Electronics Engineers Inc., 1 2021.

[39] P. Xiang, P. Ji, and D. Zhang. Enhance RSS-Based Indoor Localization Accuracy by Leveraging Environmental Physical Features. *Wireless Communications and Mobile Computing*, 2018, 2018.

[40] Y. Yu, R. Chen, L. Chen, X. Zheng, D. Wu, W. Li, and Y. Wu. A Novel 3-D Indoor Localization Algorithm Based on BLE and Multiple Sensors. *IEEE Internet of Things Journal*, 8(11):9359–9372, 2021.

Appendix A

Algorithm UML

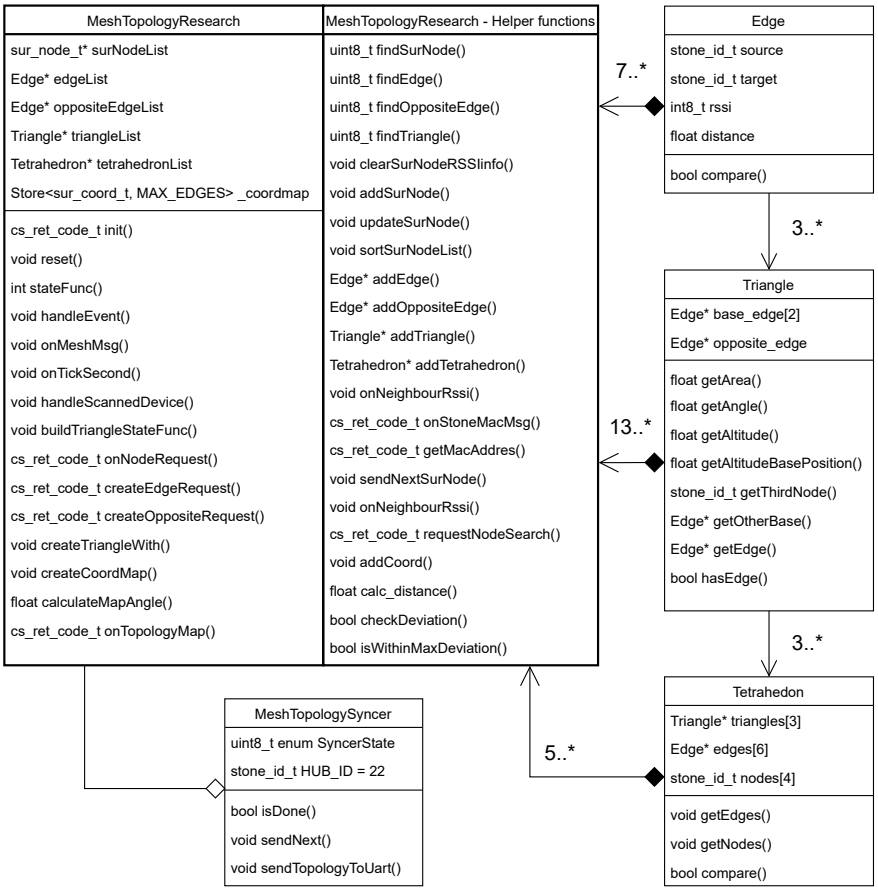


Figure A.1: **UML of the spatial network mapping algorithm (C++)**

Appendix B

Database

Coordinate map overview

Time ↓	sourceId	nodeId	x	y	z
2024-02-26 16:09:20	11	7	-0.137	0.296	1.52
2024-02-26 16:09:20	11	14	0.575	1.24	0
2024-02-26 16:09:20	11	11	0	0	0
2024-02-26 16:09:20	11	1	1.18	0	0
2024-02-26 16:08:28	1	6	-0.912	-0.130	-1.45
2024-02-26 16:08:28	1	22	1.12	0	0
2024-02-26 16:08:28	1	15	0.446	0.101	-1.30
2024-02-26 16:08:28	1	14	0.112	0.494	1.18
2024-02-26 16:08:28	1	11	-0.296	1.12	0
2024-02-26 16:08:28	1	1	0	0	0
2024-02-26 16:07:37	7	8	0.867	0.519	1.52
2024-02-26 16:07:37	7	7	0	0	0
2024-02-26 16:07:37	7	22	1.37	0	0
2024-02-26 16:07:37	7	11	0.402	1.52	0
2024-02-26 16:07:37	7	1	1.60	1.18	0
2024-02-26 16:07:22	10	9	1.60	0	0
2024-02-26 16:07:22	10	6	1.01	1.18	0
2024-02-26 16:07:22	10	15	0.671	0.637	1.45
2024-02-26 16:07:22	10	10	0	0	0
2024-02-26 16:07:20	6	9	1.30	0	0
2024-02-26 16:07:20	6	6	0	0	0
2024-02-26 16:07:20	6	22	1.69	-0.446	1.77
2024-02-26 16:07:20	6	15	0.296	0.423	1.45

Edges overview

Time ↓	sourceId	id	source	target	rssi	distance
2024-02-26 16:09:20	11	33118d7e-a58c-4e13-85...	11	7	-55	1.60
2024-02-26 16:09:20	11	8f35de9f-6a04-4549-a6...	22	15	-54	1.52
2024-02-26 16:09:20	11	d07b7336-8f93-47bc-b0...	8	1	-59	1.96
2024-02-26 16:09:20	11	4d6b659d-fd81-420b-91...	11	15	-58	1.87
2024-02-26 16:09:20	11	f6e03cf6-1d45-4026-86...	11	14	-52	1.37
2024-02-26 16:09:20	11	6980ba97-da52-48b2-9...	11	22	-57	1.77

Triangles overview

Time ↓	sourceId	id	base1	base2	opposite
2024-02-26 16:09:20	11	d07b7336-8f93-47bc-b036-d5...	1b51463d-74cd-4400-a5ae-c6d...	dd3c231e-eb5a-4270-afd7-083...	93c9b9de-c10e-448c-95d7-ea7...
2024-02-26 16:08:28	1	0a6ad07c-c473-4ea7-8f93-fa3f...	56864168-81f6-4562-bbae-37f...	87e35e49-0200-4b47-b8c7-819...	4ed60fe5-243c-4e7b-aa67-6e3...
2024-02-26 16:07:37	7	14903972-d258-4031-8c30-bae...	08616 4df-fb9c-46fe-9941-a50f...	a548827f-417d-435e-b502-229...	d6902ae9-2b80-4f43-81a7-3cd...
2024-02-26 16:07:22	10	dbd52a78-587a-4ff0-9214-a584...	a5e7c1a2-d4b4-43cf-a747-8f98...	8b70e3db-5377-43d8-9479-941...	3c6ca808-ae87-4bf4-ab30-4c7...
2024-02-26 16:07:20	6	daf9f299-c5ee-48ee-8231-937...	bcfe9722-6c20-4e0b-8a98-c6e...	dcb86d06-99bb-4585-bcbf-a58...	d6f7b9a1-1a12-49b4-936b-b95...
2024-02-26 16:07:00	22	b30e9b3c-4037-44f7-a678-ac3...	82c509f1-8210-4439-b072-54c...	a2386 2b0-c49c-40db-98b0-67b...	09ab0234-4a3e-49b6-ba1c-738...

Overview Tetrahedrons

Time ↓	sourceId	id	triangle1	triangle2	triangle3
2024-02-26 16:09:20	11	1bbc2b69-13ce-47e6-8dc4-118f...	ee1827b2-aa5e-441b-a509-298...	0795e22d-e3bf-49be-8b9e-9c7...	39252d1b-a65b-49ab-a9e9-bc...
2024-02-26 16:08:28	1	aeae3357-7c67-429d-9a45-80...	93d82e4b-e396-42c6-ab30-1d...	fa42fc34-be73-44b2-8ad9-372...	381b76e3-bbd9-4ea1-ac0b-7bb...
2024-02-26 16:08:28	1	49f757a3-726a-49ac-ad35-f57f...	93d82e4b-e396-42c6-ab30-1d...	a4c78387-2d45-4685-abcf-e59...	77ac6b7f-069a-44c3-8e7a-d57...
2024-02-26 16:08:28	1	336ac0ea-cff4-4eec-a977-1652...	93d82e4b-e396-42c6-ab30-1d...	60657b0c-c805-4594-9460-52...	e9885021-8cf2-493d-a65e-19a...
2024-02-26 16:07:37	7	b049a3db-d166-4c2e-8b52-f8...	aa3f4c79-4b42-417d-a8de-1113...	0be40208-d766-446d-ba71-50...	d32d7af2-2ab6-488a-beb8-df2...
2024-02-26 16:07:37	7	2beb67f1-5ced-4abb-b05d-d3a...	aa3f4c79-4b42-417d-a8de-1113...	e84ba590-7728-4e24-a660-40...	6107f220-f19d-4cc2-b590-1473...

Figure B.1: **Representation of the database with all the sent topology data**

Appendix C

Local Environment Data

In this research, the potential of using humidity, temperature, and pressure as additional features in machine learning models is being explored. To test this, the RSSI between multiple nodes is being measured and stored along with measurements of humidity, temperature, and pressure, see Figure C.1. It can be seen that the humidity shows sinusoidal behaviour.

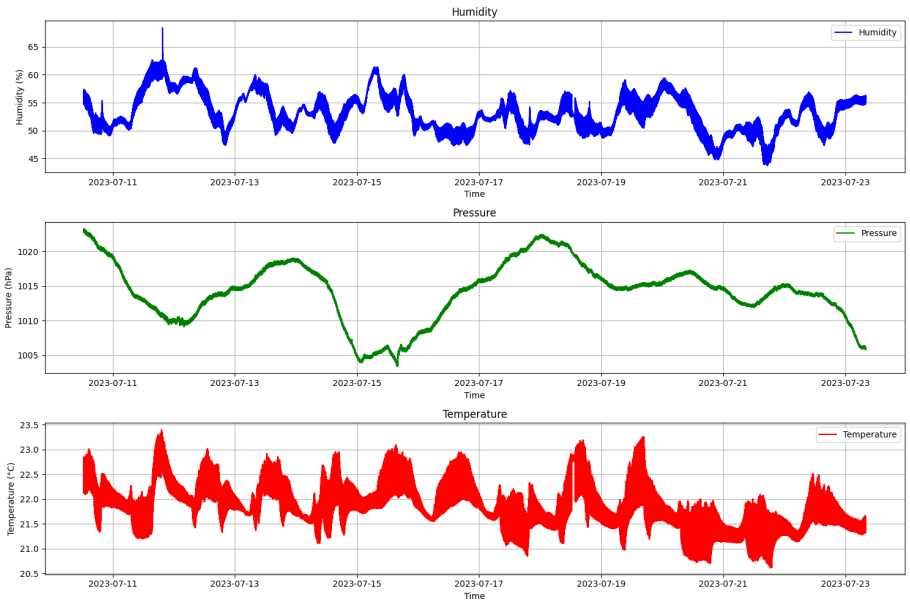


Figure C.1: **Time series plot of the humidity, pressure and temperature.**

An ML model has been trained and the feature importance result indicates that humidity has the greatest effect on the RSSI. Refer to Figure C.2. It would be beneficial to incorporate this feature into an RSSI to distance model for future work.

```
Attribute Evaluator (supervised, Class (nominal): 8  channel-0_last-50-records_label):
        Correlation Ranking Filter
Ranked attributes:
 0.7138    7 hum
 0.5152    3  channel-0_last-50-records_median_grouped
 0.4205    1  channel-0_last-50-records_mean
 0.3609    5 temp
 0.1636    6 press
 0.1574    4  channel-0_last-50-records_min_max_gap
 0.039     2  channel-0_last-50-records_stdev

Selected attributes: 7,3,1,5,6,4,2 : 7
```

Figure C.2: **Feature importance output.**

Appendix D

Machine Learning Models

There are two topics of interest for this research where an ML model can be applied: RSSI to distance conversion and RSSI-based localisation. These small neural networks can be developed on a local machine with Python using Keras[26] sequential model. In this sequential model, multiple different layers can be added and the input features and output can be defined. Two distinct datasets can be created which will be utilised for training the models. The first dataset will contain RSSI readings at multiple distances. In combination with environmental variables like temperature, humidity and pressure. The other dataset should also contain the previous variables, but now from multiple nodes. Before training, the datasets will be preprocessed to obtain the mean, variance and standard deviation of a specified time window. This will give extra features that can increase accuracy. The goal of these models is to achieve high accuracy with a minimally sized neural network that accommodates memory constraints on the hardware platform. The Python-generated model will be converted to C++ and optimised for microcontrollers to achieve this.

1. RSSI-to-distance conversion model: takes as input the current RSSI, the mean, the variance and standard deviation of RSSI, and the temperature, humidity and pressure over a to-be specified window.
2. RSSI-based Localisation model: takes as input the 7 previously mentioned variables multiplied by n the number of used nodes. The environmental data will probably be the same for groups of nodes, so this will later change the amount of inputs.

```
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(7 or 7*n,)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(3)
])
```

The Keras models can be converted to ONNX models after training and approval of accuracy. The ONNX models can be further compressed with the Eclipse Aidge platform [14] so it can run on a microcontroller. Aidge is particularly useful for neural network design and exploration, allowing simple and fast prototyping of neural networks with different topologies.

Appendix E

Determination of the rotation matrices with SVD

The SVD calculation uses two sets of points: base_edge (source) and other_edge (target) coordinates. The target coordinates, represented as matrix AD in Figure E.1, are used as the target coordinates, while the other original coordinates are used as the source coordinates. The method aims to determine the rotation matrix needed to align the source coordinates with the target coordinates. In short, the method centres both sets of points around their mean values and then computes the covariance matrix between the centred points. Subsequently, the function employs Singular Value Decomposition (SVD) on the covariance matrix to derive the rotation matrix that minimises the squared distance between the rotated source points and the target points. The results of the simulator and the proof that a rotation matrix can be found for each coordinate map output show that the mathematics is indeed correct and that a matching 3D spatial map can be created for each edge.

```
Before rotation:
Original coords: AD:
[[0.  0.  0. ]
 [1.5 3.5 0. ]
 [3.  2.5 1. ]
 [3.5 0.  0. ]
 [5.  1.  1. ]]

Original coords: AB:
[[ 0.     0.     0.   ]
 [ 3.808  0.     0.   ]
 [ 3.48   1.956 -0.562]
 [ 1.379  3.13   0.745]
 [ 2.889  4.319  0.   ]]

Original coords: AC:
[[ 0.     0.     0.   ]
 [ 3.287 -1.417  1.3  ]
 [ 4.031  0.     0.   ]
 [ 2.605  2.338  0.   ]
 [ 4.589  2.372 -0.557]]

Original coords: AE:
[[ 0.     0.     0.   ]
 [ 2.117  3.165  0.   ]
 [ 3.56   1.805  0.562]
 [ 3.368 -0.594 -0.745]
 [ 5.196  0.     0.   ]]

Calculated Matrices:
Rotation matrix: AD:
[[ 1. -0. -0.]
 [ 0.  1. -0.]
 [ 0.  0.  1.]]

Rotation matrix: AB:
[[ 0.394  0.919 -0.   ]
 [ 0.894 -0.383  0.232]
 [ 0.213 -0.091 -0.973]]

Rotation matrix: AC:
[[ 0.744  0.62   0.248]
 [ 0.668 -0.691 -0.276]
 [ 0.     0.371 -0.928]]

Rotation matrix: AE:
[[ 0.962  0.192  0.192]
 [-0.17   0.977 -0.129]
 [-0.213  0.091  0.973]]

After Rotation:
New coords: AD:
[[ 0.   0.   0. ]
 [ 1.5  3.5 -0. ]
 [ 3.   2.5  1. ]
 [ 3.5 -0.  -0. ]
 [ 5.   1.   1. ]]

New coords: AB:
[[ 0.   0.   0. ]
 [ 1.5  3.5 -0. ]
 [ 3.   2.5  1. ]
 [ 3.5 -0.   0. ]
 [ 5.   1.   1. ]]

New coords: AC:
[[ 0.   0.   0. ]
 [ 1.5  3.5  0. ]
 [ 3.   2.5  1. ]
 [ 3.5 -0.   0. ]
 [ 5.   1.   1. ]]

New coords: AE:
[[0.  0.  0. ]
 [1.5 3.5 0. ]
 [3.  2.5 1. ]
 [3.5 0.  0. ]
 [5.  1.  1. ]]
```

Figure E.1: **Validation of the output at node A involves aligning the original coordinates (A (top)- E (Bottom) with those of AD. The obtained matrix represents the rotation needed to transform the input coordinates into the coordinate space of AD.**

Appendix F

Coordmap Debug Logs

F.1 Single Tetrahedron

A log of the UART debug output of a successful coordmap procedure creating a single tetrahedron can be seen below.

Listing F.1: **UART debug log**

```
LOG: [2024-02-14 15:51:23.602676] I Max triangles reached -> Finished
LOG: [2024-02-14 15:51:25.665059] I Edges of [23]::(6):
LOG: [2024-02-14 15:51:25.671227] I o--o : target=22, rssi=-46, distance=1.049
LOG: [2024-02-14 15:51:25.676189] I o--o : target=8, rssi=-53, distance=1.504
LOG: [2024-02-14 15:51:25.679433] I o--o : target=21, rssi=-52, distance=1.383
LOG: [2024-02-14 15:51:25.681444] I o--o : target=15, rssi=-54, distance=1.460
LOG: [2024-02-14 15:51:25.683445] I o--o : target=12, rssi=-56, distance=1.642
LOG: [2024-02-14 15:51:25.685445] I o--o : target=9, rssi=-57, distance=1.847
LOG: [2024-02-14 15:51:25.687444] I Opposite edges of [23]::(10):
LOG: [2024-02-14 15:51:25.688674] I o==o : source=22, target=8, rssi=-55, distance=1.799
LOG: [2024-02-14 15:51:25.690673] I o==o : source=22, target=21, rssi=-55, distance=1.752
LOG: [2024-02-14 15:51:25.691672] I o==o : source=22, target=15, rssi=-52, distance=1.526
LOG: [2024-02-14 15:51:25.693679] I o==o : source=15, target=21, rssi=-54, distance=1.658
LOG: [2024-02-14 15:51:25.695747] I o==o : source=22, target=12, rssi=-53, distance=1.568
LOG: [2024-02-14 15:51:25.697193] I o==o : source=8, target=12, rssi=-60, distance=2.301
LOG: [2024-02-14 15:51:25.698193] I o==o : source=21, target=12, rssi=-56, distance=1.900
LOG: [2024-02-14 15:51:25.700334] I o==o : source=8, target=9, rssi=-61, distance=2.431
LOG: [2024-02-14 15:51:25.701334] I o==o : source=21, target=9, rssi=-58, distance=2.120
LOG: [2024-02-14 15:51:25.703338] I o==o : source=15, target=9, rssi=-57, distance=2.007
LOG: [2024-02-14 15:51:25.704338] I Triangles: (10)
LOG: [2024-02-14 15:51:25.706339] I o=o=o : id=0, (23:22:8): area=0.788, angle=87.709, altitude base1
    =1.503, altitude base2=1.048, altitude opposite=0.876
LOG: [2024-02-14 15:51:25.707338] I o=o=o : id=1, (23:22:21): area=0.725, angle=91.133, altitude base1
    =1.383, altitude base2=1.048, altitude opposite=0.828
LOG: [2024-02-14 15:51:25.709559] I o=o=o : id=2, (23:22:15): area=0.732, angle=72.829, altitude base1
    =1.395, altitude base2=1.002, altitude opposite=0.959
LOG: [2024-02-14 15:51:25.711562] I o=o=o : id=3, (23:15:21): area=0.956, angle=71.308, altitude base1
    =1.310, altitude base2=1.383, altitude opposite=1.153
LOG: [2024-02-14 15:51:25.713592] I o=o=o : id=4, (23:22:12): area=0.793, angle=67.144, altitude base1
    =1.513, altitude base2=0.966, altitude opposite=1.012
LOG: [2024-02-14 15:51:25.714597] I o=o=o : id=5, (23:8:12): area=1.231, angle=93.924, altitude base1
    =1.638, altitude base2=1.500, altitude opposite=1.070
LOG: [2024-02-14 15:51:25.715600] I o=o=o : id=6, (23:21:12): area=1.107, angle=77.307, altitude base1
    =1.602, altitude base2=1.349, altitude opposite=1.166
LOG: [2024-02-14 15:51:25.717604] I o=o=o : id=7, (23:8:9): area=1.388, angle=92.415, altitude base1
    =1.846, altitude base2=1.502, altitude opposite=1.142
LOG: [2024-02-14 15:51:25.718934] I o=o=o : id=8, (23:21:9): area=1.260, angle=80.652, altitude base1
    =1.823, altitude base2=1.364, altitude opposite=1.189
LOG: [2024-02-14 15:51:25.720921] I o=o=o : id=9, (23:15:9): area=1.294, angle=73.669, altitude base1
    =1.773, altitude base2=1.401, altitude opposite=1.290
LOG: [2024-02-14 15:51:25.721920] I Coord (23)(0.000, 0.000, 0.000) added to Map
LOG: [2024-02-14 15:51:25.723934] I CoordMap Base edge = 23:22
LOG: [2024-02-14 15:51:25.724922] I Coord (22)(1.049, 0.000, 0.000) added to Map
```

```
LOG: [2024-02-14 15:51:25.726935] I Coord (8)(0.060, 1.503, 0.000) added to Map
LOG: [2024-02-14 15:51:25.727920] I Base Triangle 23:22:8
LOG: [2024-02-14 15:51:25.729824] I CoordMap:
LOG: [2024-02-14 15:51:25.730835] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.732270] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.733278] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:25.734280] I Init Coord: -0.027, 1.383, 0.000
LOG: [2024-02-14 15:51:25.736281] I Find angle between 8 and 21
LOG: [2024-02-14 15:51:25.737281] I No opposite edge found
LOG: [2024-02-14 15:51:25.738414] I MapAngle: -1.000
LOG: [2024-02-14 15:51:25.739417] I Init Coord (rotated): -0.027, 1.382, -0.024
LOG: [2024-02-14 15:51:25.740416] I No triangle found with thirdNode 21 and other_base 8
LOG: [2024-02-14 15:51:25.741416] I o Node 21 will not be mapped with triangle 1
LOG: [2024-02-14 15:51:25.742416] I CoordMap:
LOG: [2024-02-14 15:51:25.742416] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.743416] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.744417] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:25.745416] I Init Coord: 0.431, 1.395, 0.000
LOG: [2024-02-14 15:51:25.746416] I Find angle between 8 and 15
LOG: [2024-02-14 15:51:25.747416] I No opposite edge found
LOG: [2024-02-14 15:51:25.748473] I MapAngle: -1.000
LOG: [2024-02-14 15:51:25.748473] I Init Coord (rotated): 0.431, 1.395, -0.024
LOG: [2024-02-14 15:51:25.749509] I No triangle found with thirdNode 15 and other_base 8
LOG: [2024-02-14 15:51:25.750498] I o Node 15 will not be mapped with triangle 2
LOG: [2024-02-14 15:51:25.751499] I Triangle 3 (23:15:21) has no baseEdge 23:22
LOG: [2024-02-14 15:51:25.752145] I CoordMap:
LOG: [2024-02-14 15:51:25.753156] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.755504] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:25.756501] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:25.758597] I Init Coord: 0.638, 1.513, 0.000
LOG: [2024-02-14 15:51:25.759603] I Find angle between 8 and 12
LOG: [2024-02-14 15:51:25.761165] I base posX A = 0.060, base posX B = 0.638
LOG: [2024-02-14 15:51:25.762811] I nodeDist = 2.301 -> deltaX = 0.578 -> TranslatedNodeDist = 2.227
LOG: [2024-02-14 15:51:25.763821] I AltitudeH A = 1.503, AltitudeH B = 1.513
LOG: [2024-02-14 15:51:25.764822] I MapAngle: 95.232
LOG: [2024-02-14 15:51:25.766823] I Init Coord (rotated): 0.638, -0.138, 1.506
LOG: [2024-02-14 15:51:25.767822] I Check Coord: -0.112, 1.638, 0.000
LOG: [2024-02-14 15:51:25.769975] I Find angle between 22 and 12
LOG: [2024-02-14 15:51:25.770976] I base posX A = 0.042, base posX B = -0.112
LOG: [2024-02-14 15:51:25.771975] I nodeDist = 1.568 -> deltaX = 0.154 -> TranslatedNodeDist = 1.560
LOG: [2024-02-14 15:51:25.773981] I AltitudeH A = 1.048, AltitudeH B = 1.638
LOG: [2024-02-14 15:51:25.774980] I MapAngle: 66.897
LOG: [2024-02-14 15:51:25.775981] I CheckCoord: -0.112, 0.643, 1.506
LOG: [2024-02-14 15:51:25.777011] I Distance between 8 and 12 is 2.301
LOG: [2024-02-14 15:51:25.779071] I compare -0.112, 0.643, 1.506 with 0.638, -0.138, 1.506
LOG: [2024-02-14 15:51:25.780979] I (YAW) checkCoord: -0.647, -0.087, 1.506
LOG: [2024-02-14 15:51:25.783005] I compare -0.647, -0.087, 1.506 with 0.638, -0.138, 1.506
LOG: [2024-02-14 15:51:25.785004] I (-YAW) checkCoord: -0.112, 0.643, 1.506
LOG: [2024-02-14 15:51:25.786006] I (Y) checkCoord: -0.112, -0.643, 1.506
LOG: [2024-02-14 15:51:25.787004] I (Y+YAW) checkCoord: 0.638, -0.138, 1.506
LOG: [2024-02-14 15:51:25.789587] I compare 0.638, -0.138, 1.506 with 0.638, -0.138, 1.506
LOG: [2024-02-14 15:51:25.791588] I Check deviation between 2.301 and 2.301
LOG: [2024-02-14 15:51:25.792592] I Coord (12)(0.638, -0.138, 1.506) added to Map
LOG: [2024-02-14 15:51:25.794588] I Triangle 5 (23:8:12) has no baseEdge 23:22
LOG: [2024-02-14 15:51:25.795588] I Triangle 6 (23:21:12) has no baseEdge 23:22
LOG: [2024-02-14 15:51:25.796589] I Triangle 7 (23:8:9) has no baseEdge 23:22
LOG: [2024-02-14 15:51:25.798749] I Triangle 8 (23:21:9) has no baseEdge 23:22
LOG: [2024-02-14 15:51:25.799390] I Triangle 9 (23:15:9) has no baseEdge 23:22
LOG: [2024-02-14 15:51:26.827665] I CoordMap:
LOG: [2024-02-14 15:51:26.835985] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:26.842396] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:26.849071] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:26.855804] I Node 12: x = 0.638, y = -0.138, z = 1.506
LOG: [2024-02-14 15:51:26.865581] I Tetrahedrons: (1)
LOG: [2024-02-14 15:51:26.869285] I o=o=o=o : id=0, triangles=0,4,5
LOG: [2024-02-14 15:51:26.874335] I isvalid? 1
LOG: [2024-02-14 15:51:26.877356] I --> Send coord: id=23, x=0.000, y=0.000, z=0.000
LOG: [2024-02-14 15:51:26.924428] I reliable msg success
LOG: [2024-02-14 15:51:29.405133] [rce/src/ble/cs_ServiceData.cpp: 533] I sendMeshState
LOG: [2024-02-14 15:51:36.384043] I CoordMap:
LOG: [2024-02-14 15:51:36.389597] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:36.395815] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:36.401534] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:36.407615] I Node 12: x = 0.638, y = -0.138, z = 1.506
```

```
LOG: [2024-02-14 15:51:36.413795] I isvalid? 1
LOG: [2024-02-14 15:51:36.419126] I --> Send coord: id=22, x=1.049, y=0.000, z=0.000
LOG: [2024-02-14 15:51:37.134787] I reliable msg success
LOG: [2024-02-14 15:51:44.451231] I CoordMap:
LOG: [2024-02-14 15:51:44.457336] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:44.464114] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:44.469496] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:44.475502] I Node 12: x = 0.638, y = -0.138, z = 1.506
LOG: [2024-02-14 15:51:44.481722] I isvalid? 1
LOG: [2024-02-14 15:51:44.488726] I --> Send coord: id=8, x=0.060, y=1.503, z=0.000
LOG: [2024-02-14 15:51:44.543945] I reliable msg success
LOG: [2024-02-14 15:51:53.100590] I CoordMap:
LOG: [2024-02-14 15:51:53.102600] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:53.105115] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:51:53.106109] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:51:53.108108] I Node 12: x = 0.638, y = -0.138, z = 1.506
LOG: [2024-02-14 15:51:53.109236] I isvalid? 1
LOG: [2024-02-14 15:51:53.110277] I --> Send coord: id=12, x=0.638, y=-0.138, z=1.506
LOG: [2024-02-14 15:51:53.791547] I reliable msg success
LOG: [2024-02-14 15:52:02.014158] I <- Incomming Node request received from 21
LOG: [2024-02-14 15:52:02.017158] I -> sent reply to 21: target 10, rssi -128
LOG: [2024-02-14 15:52:02.197094] I <- Incomming Node request received from 21
LOG: [2024-02-14 15:52:02.208280] I -> sent reply to 21: target 10, rssi -128
LOG: [2024-02-14 15:52:02.717449] I CoordMap:
LOG: [2024-02-14 15:52:02.721812] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-14 15:52:02.725811] I Node 22: x = 1.049, y = 0.000, z = 0.000
LOG: [2024-02-14 15:52:02.730193] I Node 8: x = 0.060, y = 1.503, z = 0.000
LOG: [2024-02-14 15:52:02.733538] I Node 12: x = 0.638, y = -0.138, z = 1.506
LOG: [2024-02-14 15:52:02.737553] I Finished sending topology
LOG: [2024-02-14 15:52:02.740797] I TOPOLOGY RESEARCH DONE!
```

F.2 Multiple Tetrahedron

A log of the UART debug output of a successful coordmap procedure creating multiple tetrahedrons can be seen below.

Listing F.2: **Coordmap Procedure Debug Log**

```
LOG: [2024-02-15 16:39:15.770069] I Max triangles reached -> Finished
LOG: [2024-02-15 16:39:17.846295] I Edges of [23]::(6):
LOG: [2024-02-15 16:39:17.848315] I o--o : target=21, rssi=-45, distance=0.935
LOG: [2024-02-15 16:39:17.849295] I o--o : target=12, rssi=-54, distance=1.560
LOG: [2024-02-15 16:39:17.851295] I o--o : target=15, rssi=-55, distance=1.572
LOG: [2024-02-15 16:39:17.852294] I o--o : target=11, rssi=-57, distance=1.824
LOG: [2024-02-15 16:39:17.854294] I o--o : target=8, rssi=-56, distance=1.670
LOG: [2024-02-15 16:39:17.855297] I o--o : target=14, rssi=-57, distance=1.866
LOG: [2024-02-15 16:39:17.856297] I Opposite edges of [23]::(12):
LOG: [2024-02-15 16:39:17.857297] I o==o : source=12, target=21, rssi=-59, distance=2.181
LOG: [2024-02-15 16:39:17.859297] I o==o : source=21, target=15, rssi=-53, distance=1.570
LOG: [2024-02-15 16:39:17.860514] I o==o : source=12, target=15, rssi=-56, distance=1.848
LOG: [2024-02-15 16:39:17.861525] I o==o : source=21, target=11, rssi=-63, distance=2.712
LOG: [2024-02-15 16:39:17.862524] I o==o : source=12, target=11, rssi=-49, distance=1.261
LOG: [2024-02-15 16:39:17.863524] I o==o : source=11, target=15, rssi=-58, distance=2.065
LOG: [2024-02-15 16:39:17.864525] I o==o : source=21, target=8, rssi=-59, distance=2.178
LOG: [2024-02-15 16:39:17.866524] I o==o : source=8, target=15, rssi=-58, distance=2.062
LOG: [2024-02-15 16:39:17.867524] I o==o : source=21, target=14, rssi=-63, distance=2.712
LOG: [2024-02-15 16:39:17.868523] I o==o : source=14, target=12, rssi=-59, distance=2.178
LOG: [2024-02-15 16:39:17.870759] I o==o : source=11, target=14, rssi=-48, distance=1.194
LOG: [2024-02-15 16:39:17.872162] I o==o : source=8, target=14, rssi=-57, distance=1.952
LOG: [2024-02-15 16:39:17.872162] I Triangles: (12)
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=0, (23:12:21): area=0.633, angle=119.811, altitude
    base1=0.811, altitude base2=1.354, altitude opposite=0.580
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=1, (23:21:15): area=0.701, angle=72.589, altitude base1
    =1.500, altitude base2=0.892, altitude opposite=0.893
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=2, (23:12:15): area=1.168, angle=72.325, altitude base1
    =1.497, altitude base2=1.486, altitude opposite=1.264
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=3, (23:21:11): area=0.323, angle=157.711, altitude
    base1=0.692, altitude base2=0.355, altitude opposite=0.238
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=4, (23:12:11): area=0.968, angle=42.877, altitude base1
    =1.241, altitude base2=1.061, altitude opposite=1.535
```

```
LOG: [2024-02-15 16:39:17.872162] I o=o=o : id=5, (23:11:15): area=1.381, angle=74.510, altitude base1
    =1.514, altitude base2=1.757, altitude opposite=1.337
LOG: [2024-02-15 16:39:17.887787] I o=o=o : id=6, (23:21:8): area=0.732, angle=110.269, altitude base1
    =1.566, altitude base2=0.877, altitude opposite=0.673
LOG: [2024-02-15 16:39:17.887787] I o=o=o : id=7, (23:8:15): area=1.288, angle=78.950, altitude base1
    =1.542, altitude base2=1.639, altitude opposite=1.249
LOG: [2024-02-15 16:39:17.887787] I o=o=o : id=8, (23:21:14): area=0.447, angle=149.205, altitude
    base1=0.955, altitude base2=0.479, altitude opposite=0.329
LOG: [2024-02-15 16:39:17.887787] I o=o=o : id=9, (23:14:12): area=1.426, angle=78.387, altitude base1
    =1.528, altitude base2=1.828, altitude opposite=1.309
LOG: [2024-02-15 16:39:17.887787] I o=o=o : id=10, (23:11:14): area=1.041, angle=37.723, altitude
    base1=1.142, altitude base2=1.116, altitude opposite=1.745
LOG: [2024-02-15 16:39:17.899293] I o=o=o : id=11, (23:8:14): area=1.431, angle=66.744, altitude base1
    =1.715, altitude base2=1.534, altitude opposite=1.467
LOG: [2024-02-15 16:39:17.899293] I Coord (23)(0.000, 0.000, 0.000) added to Map
LOG: [2024-02-15 16:39:17.903801] I CoordMap Base edge = 23:21
LOG: [2024-02-15 16:39:17.903801] I Coord (21)(0.935, 0.000, 0.000) added to Map
LOG: [2024-02-15 16:39:17.903801] I Coord (12)(-0.776, 1.354, 0.000) added to Map
LOG: [2024-02-15 16:39:17.903801] I Base Triangle 23:12:21
LOG: [2024-02-15 16:39:17.903801] I CoordMap:
LOG: [2024-02-15 16:39:17.903801] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.903801] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.903801] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:17.903801] I Try Triangle 23:21:15
LOG: [2024-02-15 16:39:17.903801] I Init Coord: 0.470, 1.500, 0.000
LOG: [2024-02-15 16:39:17.903801] I Find angle between 12 and 15
LOG: [2024-02-15 16:39:17.919432] I base posX A = -0.776, base posX B = 0.470
LOG: [2024-02-15 16:39:17.919432] I nodeDist = 1.848 -> deltaX = 1.246 -> TranslatedNodeDist = 1.365
LOG: [2024-02-15 16:39:17.919432] I AltitudeH A = 1.354, AltitudeH B = 1.500
LOG: [2024-02-15 16:39:17.919432] I MapAngle: 56.879
LOG: [2024-02-15 16:39:17.919432] I Init Coord (rotated): 0.470, 0.819, 1.256
LOG: [2024-02-15 16:39:17.919432] I Check Coord: 0.477, 1.497, 0.000
LOG: [2024-02-15 16:39:17.919432] I Find angle between 21 and 15
LOG: [2024-02-15 16:39:17.919432] I base posX A = -0.465, base posX B = 0.477
LOG: [2024-02-15 16:39:17.935095] I nodeDist = 1.570 -> deltaX = 0.942 -> TranslatedNodeDist = 1.256
LOG: [2024-02-15 16:39:17.935095] I AltitudeH A = 0.811, AltitudeH B = 1.497
LOG: [2024-02-15 16:39:17.935095] I MapAngle: 57.007
LOG: [2024-02-15 16:39:17.935095] I CheckCoord (rotated): 0.477, 0.815, 1.256
LOG: [2024-02-15 16:39:17.950682] I Distance between 12 and 15 is 1.848
LOG: [2024-02-15 16:39:17.950682] I compare 0.477, 0.815, 1.256 with 0.470, 0.819, 1.256
LOG: [2024-02-15 16:39:17.950682] I Check deviation between 1.848 and 1.854: 0.003
LOG: [2024-02-15 16:39:17.950682] I Coord (15)(0.477, 0.815, 1.256) added to Map
LOG: [2024-02-15 16:39:17.950682] I Triangle 2 (23:12:15) has no baseEdge 23:21
LOG: [2024-02-15 16:39:17.950682] I CoordMap:
LOG: [2024-02-15 16:39:17.950682] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.950682] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.950682] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:17.950682] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:17.950682] I Try Triangle 23:21:11
LOG: [2024-02-15 16:39:17.966309] I Init Coord: -1.687, 0.692, 0.000
LOG: [2024-02-15 16:39:17.966309] I Find angle between 12 and 11
LOG: [2024-02-15 16:39:17.966309] I base posX A = -0.776, base posX B = -1.687
LOG: [2024-02-15 16:39:17.966309] I nodeDist = 1.261 -> deltaX = 0.912 -> TranslatedNodeDist = 0.871
LOG: [2024-02-15 16:39:17.966309] I AltitudeH A = 1.354, AltitudeH B = 0.692
LOG: [2024-02-15 16:39:17.966309] I MapAngle: 33.999
LOG: [2024-02-15 16:39:17.966309] I Init Coord (rotated): -1.687, 0.573, 0.387
LOG: [2024-02-15 16:39:17.966309] I Check Coord: 1.336, 1.241, 0.000
LOG: [2024-02-15 16:39:17.966309] I Find angle between 21 and 11
LOG: [2024-02-15 16:39:17.966309] I base posX A = -0.465, base posX B = 1.336
LOG: [2024-02-15 16:39:17.966309] I nodeDist = 2.712 -> deltaX = 1.801 -> TranslatedNodeDist = 2.028
LOG: [2024-02-15 16:39:17.966309] I AltitudeH A = 0.811, AltitudeH B = 1.241
LOG: [2024-02-15 16:39:17.981932] I MapAngle: 161.839
LOG: [2024-02-15 16:39:17.981932] I CheckCoord (rotated): 1.336, -1.179, 0.387
LOG: [2024-02-15 16:39:17.981932] I Distance between 15 and 11 is 2.065
LOG: [2024-02-15 16:39:17.981932] I compare 1.336, -1.179, 0.387 with -1.687, 0.573, 0.387
LOG: [2024-02-15 16:39:17.981932] I (YAW) checkCoord: 0.359, 1.746, 0.387
LOG: [2024-02-15 16:39:17.981932] I compare 0.359, 1.746, 0.387 with -1.687, 0.573, 0.387
LOG: [2024-02-15 16:39:17.981932] I (-YAW) checkCoord: 1.336, -1.179, 0.387
LOG: [2024-02-15 16:39:17.981932] I (Y) checkCoord: 1.336, 1.179, 0.387
LOG: [2024-02-15 16:39:17.981932] I (Y+YAW) checkCoord: -1.687, 0.573, 0.387
LOG: [2024-02-15 16:39:17.997561] I compare -1.687, 0.573, 0.387 with -1.687, 0.573, 0.387
LOG: [2024-02-15 16:39:17.997561] I Check deviation between 2.065 and 2.345: 0.136
LOG: [2024-02-15 16:39:17.999566] I Coord (11)(-1.687, 0.573, 0.387) added to Map
LOG: [2024-02-15 16:39:17.999566] I Triangle 4 (23:12:11) has no baseEdge 23:21
```

```
LOG: [2024-02-15 16:39:17.999566] I Triangle 5 (23:11:15) has no baseEdge 23:21
LOG: [2024-02-15 16:39:17.999566] I CoordMap:
LOG: [2024-02-15 16:39:17.999566] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.999566] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:17.999566] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:17.999566] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:17.999566] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:18.013572] I Try Triangle 23:21:8
LOG: [2024-02-15 16:39:18.013572] I Init Coord: -0.578, 1.566, 0.000
LOG: [2024-02-15 16:39:18.013572] I Find angle between 12 and 8
LOG: [2024-02-15 16:39:18.013572] I No opposite edge found
LOG: [2024-02-15 16:39:18.013572] I MapAngle: -1.000
LOG: [2024-02-15 16:39:18.013572] I MapAngle initCoord failed
LOG: [2024-02-15 16:39:18.013572] I Triangle 7 (23:8:15) has no baseEdge 23:21
LOG: [2024-02-15 16:39:18.013572] I CoordMap:
LOG: [2024-02-15 16:39:18.013572] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:18.013572] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:18.013572] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:18.013572] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:18.029205] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:18.029205] I Try Triangle 23:21:14
LOG: [2024-02-15 16:39:18.029205] I Init Coord: -1.603, 0.955, 0.000
LOG: [2024-02-15 16:39:18.029205] I Find angle between 12 and 14
LOG: [2024-02-15 16:39:18.029205] I base posX A = -0.776, base posX B = -1.603
LOG: [2024-02-15 16:39:18.029205] I nodeDist = 2.178 -> deltaX = 0.828 -> TranslatedNodeDist = 2.015
LOG: [2024-02-15 16:39:18.029205] I AltitudeH A = 1.354, AltitudeH B = 0.955
LOG: [2024-02-15 16:39:18.029205] I MapAngle: 120.543
LOG: [2024-02-15 16:39:18.029205] I Init Coord (rotated): -1.603, -0.486, 0.823
LOG: [2024-02-15 16:39:18.029205] I Check Coord: 0.376, 1.828, 0.000
LOG: [2024-02-15 16:39:18.029205] I Find angle between 21 and 14
LOG: [2024-02-15 16:39:18.029205] I base posX A = -0.465, base posX B = 0.376
LOG: [2024-02-15 16:39:18.044829] I nodeDist = 2.712 -> deltaX = 0.841 -> TranslatedNodeDist = 2.579
LOG: [2024-02-15 16:39:18.044829] I AltitudeH A = 0.811, AltitudeH B = 1.828
LOG: [2024-02-15 16:39:18.044829] I MapAngle: 153.247
LOG: [2024-02-15 16:39:18.044829] I CheckCoord (rotated): 0.376, -1.632, 0.823
LOG: [2024-02-15 16:39:18.044829] I Distance between 11 and 14 is 1.194
LOG: [2024-02-15 16:39:18.044829] I compare 0.376, -1.632, 0.823 with -1.603, -0.486, 0.823
LOG: [2024-02-15 16:39:18.044829] I (YAW) checkCoord: 1.230, 1.138, 0.823
LOG: [2024-02-15 16:39:18.044829] I compare 1.230, 1.138, 0.823 with -1.603, -0.486, 0.823
LOG: [2024-02-15 16:39:18.044829] I (-YAW) checkCoord: 0.376, -1.632, 0.823
LOG: [2024-02-15 16:39:18.044829] I (Y) checkCoord: 0.376, 1.632, 0.823
LOG: [2024-02-15 16:39:18.060455] I (Y+YAW) checkCoord: -1.603, -0.486, 0.823
LOG: [2024-02-15 16:39:18.060455] I compare -1.603, -0.486, 0.823 with -1.603, -0.486, 0.823
LOG: [2024-02-15 16:39:18.060455] I Check deviation between 1.194 and 1.148: 0.038
LOG: [2024-02-15 16:39:18.060455] I Coord (14)(-1.603, -0.486, 0.823) added to Map
LOG: [2024-02-15 16:39:18.060455] I Triangle 9 (23:14:12) has no baseEdge 23:21
LOG: [2024-02-15 16:39:18.060455] I Triangle 10 (23:11:14) has no baseEdge 23:21
LOG: [2024-02-15 16:39:18.060455] I Triangle 11 (23:8:14) has no baseEdge 23:21
LOG: [2024-02-15 16:39:19.095939] I CoordMap:
LOG: [2024-02-15 16:39:19.095939] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:19.106515] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:19.112024] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:19.112024] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:19.112024] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:19.112024] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:39:19.112024] I Tetrahedrons: (3)
LOG: [2024-02-15 16:39:19.112024] I o=o=o=o : id=0, triangles=0,1,2
LOG: [2024-02-15 16:39:19.112024] I o=o=o=o : id=1, triangles=0,3,4
LOG: [2024-02-15 16:39:19.112024] I o=o=o=o : id=2, triangles=0,8,9
LOG: [2024-02-15 16:39:19.112024] I isvalid? 1
LOG: [2024-02-15 16:39:19.112024] I --> Send coord: id=23, x=0.000, y=0.000, z=0.000
LOG: [2024-02-15 16:39:19.253499] I reliable msg success
LOG: [2024-02-15 16:39:28.302089] I CoordMap:
LOG: [2024-02-15 16:39:28.302089] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:28.302089] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:28.317239] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:28.317239] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:28.317239] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:28.317239] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:39:28.317239] I isvalid? 1
LOG: [2024-02-15 16:39:28.317239] I --> Send coord: id=21, x=0.935, y=0.000, z=0.000
LOG: [2024-02-15 16:39:28.460418] I reliable msg success
LOG: [2024-02-15 16:39:37.670487] I CoordMap:
LOG: [2024-02-15 16:39:37.673922] I Node 23: x = 0.000, y = 0.000, z = 0.000
```

```
LOG: [2024-02-15 16:39:37.676918] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:37.678920] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:37.680918] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:37.681920] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:37.682920] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:39:37.683920] I isvalid? 1
LOG: [2024-02-15 16:39:37.685918] I --> Send coord: id=12, x=-0.776, y=1.354, z=0.000
LOG: [2024-02-15 16:39:37.775551] I reliable msg success
LOG: [2024-02-15 16:39:44.887834] I sendMeshState
LOG: [2024-02-15 16:39:46.737841] I CoordMap:
LOG: [2024-02-15 16:39:46.737841] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:46.737841] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:46.753517] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:46.753517] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:46.753517] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:46.753517] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:39:46.753517] I isvalid? 1
LOG: [2024-02-15 16:39:46.753517] I --> Send coord: id=15, x=0.477, y=0.815, z=1.256
LOG: [2024-02-15 16:39:47.413663] I reliable msg success
LOG: [2024-02-15 16:39:54.323864] I CoordMap:
LOG: [2024-02-15 16:39:54.326878] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:54.326878] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:39:54.326878] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:39:54.326878] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:39:54.326878] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:39:54.342514] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:39:54.342514] I isvalid? 1
LOG: [2024-02-15 16:39:54.342514] I --> Send coord: id=11, x=-1.687, y=0.573, z=0.387
LOG: [2024-02-15 16:39:54.467095] I reliable msg success
LOG: [2024-02-15 16:40:01.571905] I CoordMap:
LOG: [2024-02-15 16:40:01.586486] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:40:01.586486] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:40:01.586486] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:40:01.586486] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:40:01.586486] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:40:01.586486] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:40:01.602124] I isvalid? 1
LOG: [2024-02-15 16:40:01.602124] I --> Send coord: id=14, x=-1.603, y=-0.486, z=0.823
LOG: [2024-02-15 16:40:01.697388] I reliable msg success
LOG: [2024-02-15 16:40:09.487751] I CoordMap:
LOG: [2024-02-15 16:40:09.487751] I Node 23: x = 0.000, y = 0.000, z = 0.000
LOG: [2024-02-15 16:40:09.487751] I Node 21: x = 0.935, y = 0.000, z = 0.000
LOG: [2024-02-15 16:40:09.487751] I Node 12: x = -0.776, y = 1.354, z = 0.000
LOG: [2024-02-15 16:40:09.487751] I Node 15: x = 0.477, y = 0.815, z = 1.256
LOG: [2024-02-15 16:40:09.503351] I Node 11: x = -1.687, y = 0.573, z = 0.387
LOG: [2024-02-15 16:40:09.503351] I Node 14: x = -1.603, y = -0.486, z = 0.823
LOG: [2024-02-15 16:40:09.503351] I Finished sending topology
LOG: [2024-02-15 16:40:09.503351] I TOPOLOGY RESEARCH DONE!
```

Appendix G

RSSI and edge distance distribution Boxplots

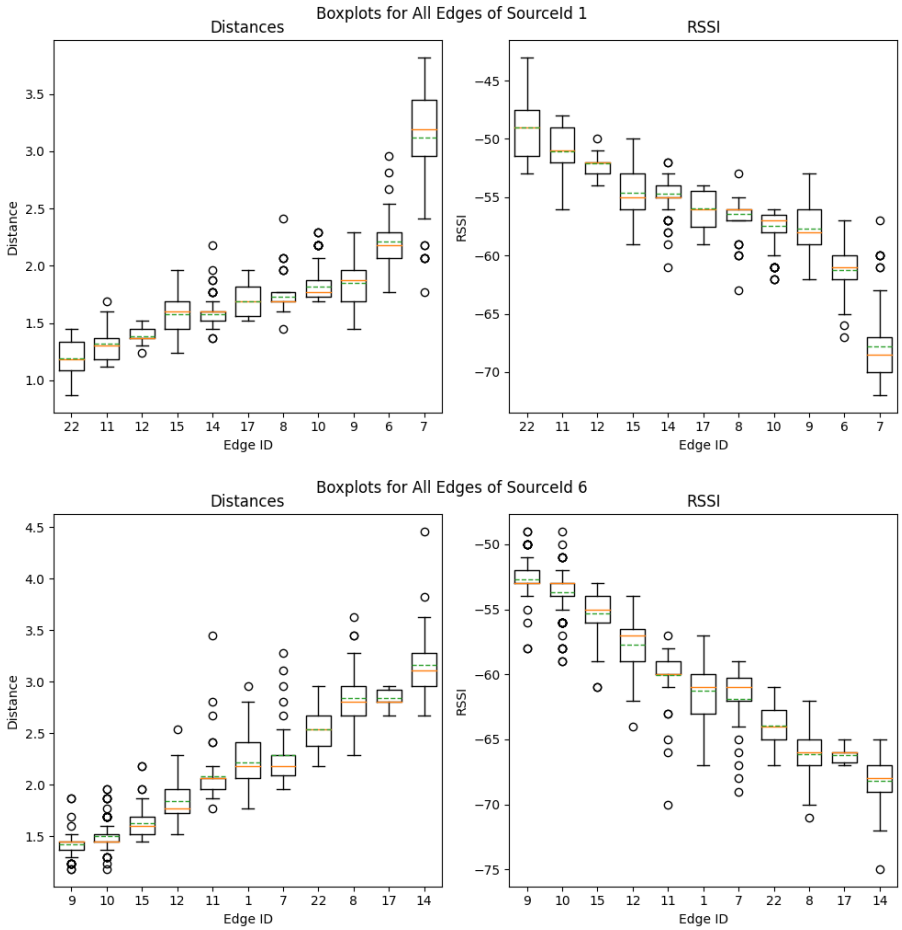


Figure G.1: **Box-plots of edge statistics for distance and RSSI, nodes sorted on RSSI.**

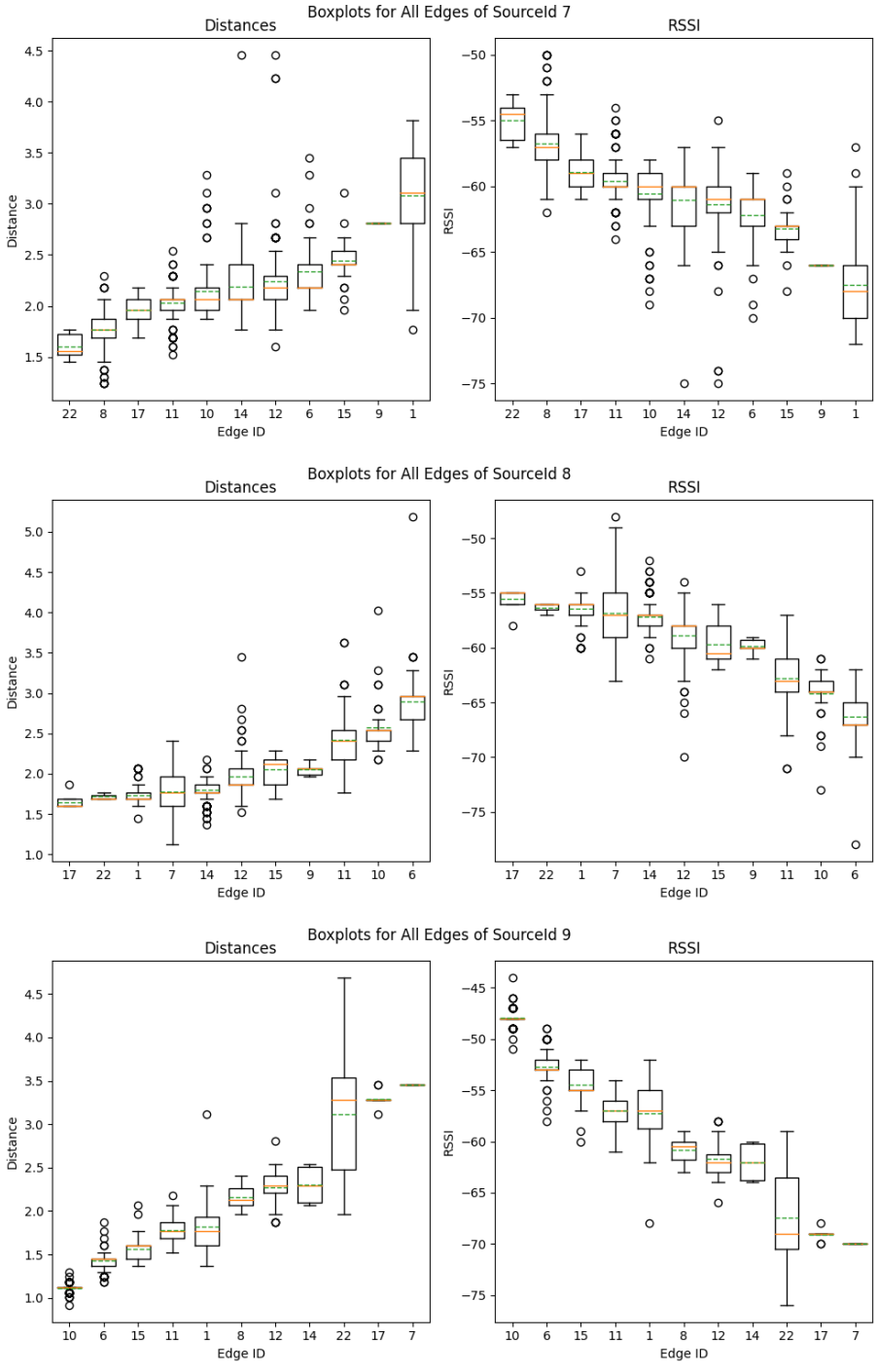


Figure G.2: **Box-plots of edge statistics for distance and RSSI, nodes sorted on RSSI.**

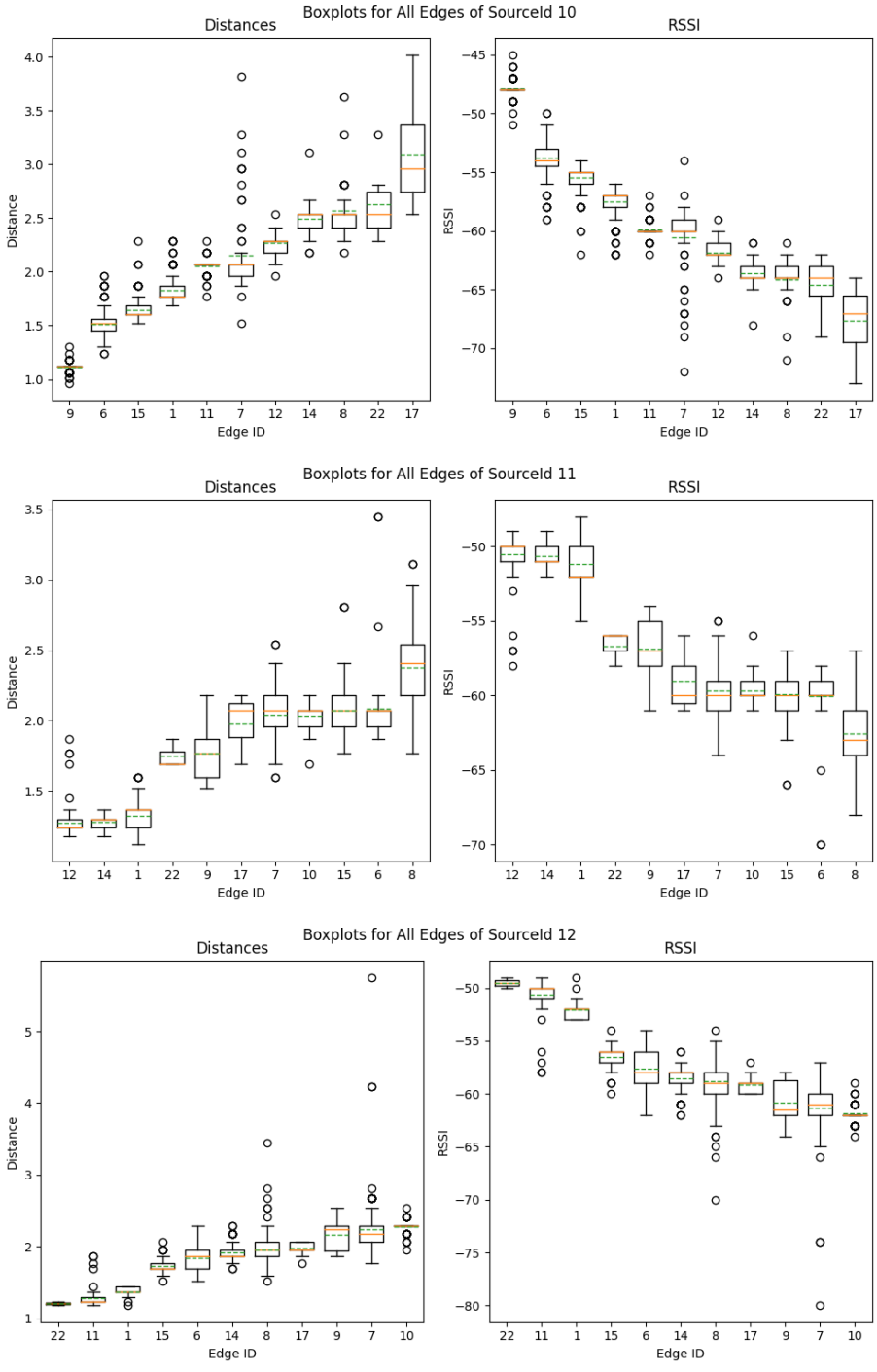


Figure G.3: **Box-plots of edge statistics for distance and RSSI, nodes sorted on RSSI.**

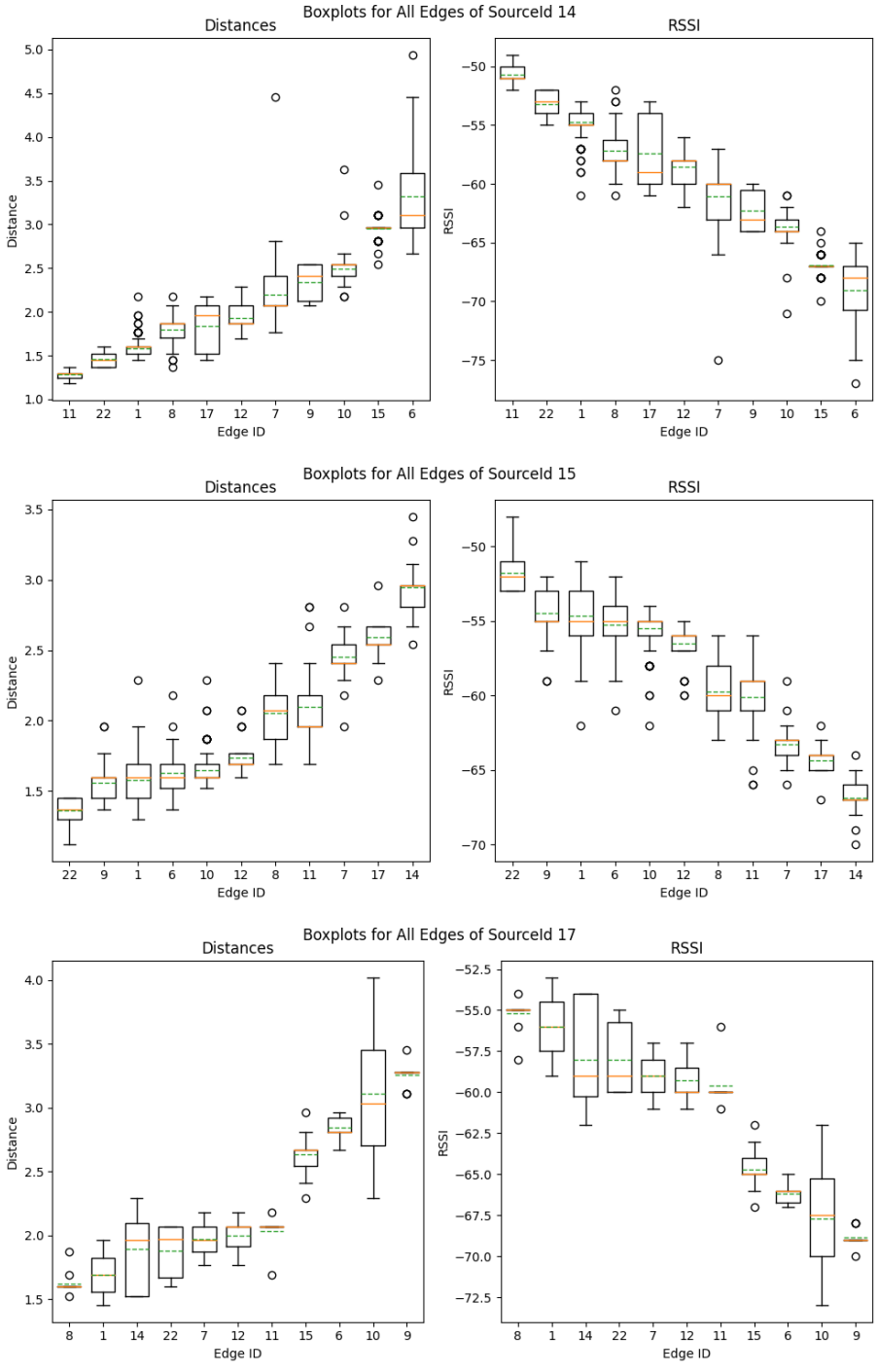
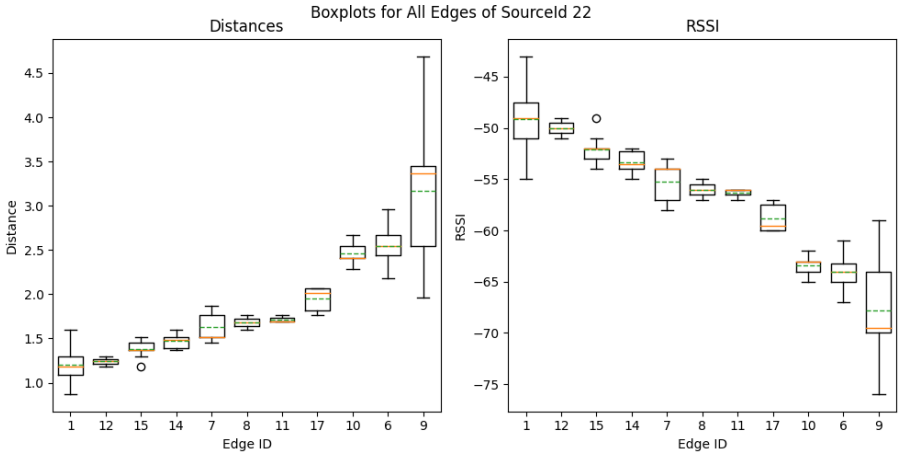


Figure G.4: **Box-plots of edge statistics for distance and RSSI, nodes sorted on RSSI.**


Boxplots for All Edges of SourceId 22
Distances
RSSI
Distance
RSSI
Edge ID
Edge ID
1
12
15
14
7
8
11
17
10
6
9

Appendix H

Edge statistics for distance and RSSI

Table H.1: **Edge statistics for distance and RSSI for Node 1 (left) and Node 6 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
9	1.426	0.100	-52.7	1.324
10	1.505	0.122	-53.7	1.53
15	1.632	0.125	-55.3	1.404
12	1.847	0.181	-57.7	1.853
11	2.083	0.251	-60.0	1.981
1	2.220	0.211	-61.3	1.814
7	2.291	0.287	-61.9	2.231
22	2.536	0.207	-63.9	1.605
8	2.846	0.294	-66.1	2.007
17	2.841	0.088	-66.2	0.6
14	3.161	0.340	-68.2	1.991

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
9	1.426	0.010	-52.7	1.752
10	1.505	0.015	-53.7	2.341
15	1.632	0.016	-55.3	1.971
12	1.847	0.033	-57.7	3.434
11	2.083	0.063	-60.0	3.926
1	2.220	0.045	-61.3	3.29
7	2.291	0.082	-61.9	4.98
22	2.536	0.043	-63.9	2.576
8	2.846	0.086	-66.1	4.027
17	2.841	0.008	-66.2	0.36
14	3.161	0.116	-68.2	3.965

Table H.2: **Edge statistics for distance and RSSI for Node 7 (left) and Node 8 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
22	1.605	0.124	-55.0	1.528
8	1.766	0.209	-56.8	2.411
17	1.962	0.142	-58.9	1.443
11	2.030	0.152	-59.6	1.498
10	2.145	0.304	-60.6	2.431
14	2.193	0.263	-61.0	2.068
12	2.246	0.404	-61.4	2.911
6	2.335	0.319	-62.2	2.437
15	2.446	0.211	-63.2	1.669
9	2.810	0.000	-66.0	0.0
1	3.086	0.514	-67.5	3.59

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
17	1.645	0.078	-55.5	0.866
22	1.717	0.038	-56.3	0.471
1	1.731	0.112	-56.5	1.22
7	1.773	0.228	-56.8	2.61
14	1.796	0.138	-57.2	1.542
12	1.964	0.213	-58.9	1.906
15	2.051	0.199	-59.7	1.967
9	2.052	0.076	-59.8	0.687
11	2.415	0.349	-62.8	2.752
10	2.578	0.288	-64.2	1.939
6	2.896	0.501	-66.3	2.927

Table H.3: **Edge statistics for distance and RSSI for Node 9 (left) and Node 10 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
10	1.116	0.049	-47.9	0.842
6	1.426	0.094	-52.7	1.258
15	1.565	0.101	-54.5	1.236
11	1.783	0.163	-57.0	1.769
1	1.818	0.313	-57.2	3.049
8	2.163	0.151	-60.8	1.344
12	2.269	0.257	-61.7	2.256
14	2.302	0.203	-62.0	1.732
22	3.113	0.711	-67.5	4.631
17	3.292	0.078	-69.1	0.457
7	3.450	0.000	-70.0	0.0

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
9	1.110	0.044	-47.8	0.753
6	1.513	0.126	-53.8	1.569
15	1.643	0.132	-55.4	1.45
1	1.827	0.142	-57.5	1.438
11	2.054	0.080	-59.8	0.761
7	2.150	0.352	-60.5	2.756
12	2.273	0.104	-61.8	0.913
14	2.491	0.115	-63.6	0.88
8	2.567	0.244	-64.1	1.698
22	2.630	0.311	-64.6	2.195
17	3.095	0.476	-67.7	2.891

Table H.4: **Edge statistics for distance and RSSI for Node 11 (left) and Node 12 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
12	1.275	0.089	-50.5	1.206
14	1.280	0.045	-50.6	0.705
1	1.322	0.121	-51.2	1.797
22	1.750	0.085	-56.7	0.943
9	1.772	0.160	-56.9	1.729
17	1.980	0.210	-59.0	2.16
7	2.039	0.151	-59.7	1.467
10	2.036	0.080	-59.7	0.788
15	2.074	0.269	-59.9	2.348
6	2.087	0.300	-60.0	2.254
8	2.377	0.301	-62.6	2.529

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
22	1.210	0.030	-49.5	0.5
11	1.281	0.091	-50.6	1.222
1	1.377	0.057	-52.1	0.778
15	1.737	0.119	-56.5	1.317
6	1.841	0.176	-57.6	1.835
14	1.921	0.134	-58.5	1.323
8	1.959	0.212	-58.8	1.895
17	1.978	0.100	-59.1	0.994
9	2.170	0.217	-60.8	1.993
7	2.247	0.470	-61.3	3.076
10	2.279	0.098	-61.9	0.858

Table H.5: **Edge statistics for distance and RSSI for Node 14 (left) and Node 15 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
11	1.284	0.047	-50.7	0.733
22	1.462	0.089	-53.2	1.166
1	1.584	0.122	-54.7	1.447
8	1.797	0.139	-57.2	1.561
17	1.837	0.293	-57.4	3.178
12	1.927	0.130	-58.6	1.296
7	2.197	0.268	-61.1	2.105
9	2.336	0.206	-62.3	1.75
10	2.494	0.165	-63.6	1.177
15	2.949	0.136	-66.9	0.9
6	3.317	0.513	-69.0	2.781

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
22	1.361	0.082	-51.8	1.208
9	1.563	0.097	-54.5	1.205
1	1.579	0.152	-54.6	1.828
6	1.627	0.116	-55.3	1.317
10	1.647	0.134	-55.5	1.486
12	1.735	0.123	-56.5	1.339
8	2.053	0.193	-59.8	1.894
11	2.096	0.289	-60.1	2.548
7	2.450	0.161	-63.3	1.317
17	2.590	0.131	-64.4	0.985
14	2.946	0.138	-66.9	0.912

Table H.6: **Edge statistics for distance and RSSI for Node 17 (left) and Node 22 (right).**

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
8	1.620	0.076	-55.2	0.86
1	1.693	0.151	-56.0	1.751
14	1.891	0.306	-58.0	3.24
22	1.880	0.210	-58.0	2.236
7	1.969	0.129	-59.0	1.279
12	1.996	0.115	-59.3	1.135
11	2.031	0.144	-59.6	1.498
15	2.634	0.134	-64.7	1.005
6	2.841	0.088	-66.2	0.6
10	3.106	0.489	-67.7	3.081
9	3.256	0.088	-68.9	0.515

Node ID	Mean Distance	Std Distance	Mean RSSI	Std RSSI
1	1.199	0.200	-49.1	3.343
12	1.240	0.060	-50.0	1.0
15	1.381	0.079	-52.1	1.109
14	1.472	0.084	-53.3	1.106
7	1.626	0.164	-55.2	1.939
8	1.685	0.085	-56.0	1.0
11	1.717	0.038	-56.3	0.471
17	1.952	0.134	-58.8	1.344
10	2.464	0.130	-63.4	1.02
6	2.550	0.236	-64.0	1.844
9	3.164	0.690	-67.8	4.503

Appendix I

MAE and RMSE plots

Figure I.1: **Comparison of RMSE and MAE variations across intervals for different Node IDs, illustrating both the original and smoothed versions filtered with a 35-interval window. The top row depicts the normal data, while the bottom shows the smoothed versions.**