POLITECNICO DI TORINO

Master Degree course in ICT For Smart Societies

Master Degree Thesis

Patient Indoor Positioning: Leveraging Wearable Technology for Location Awareness

Supervisors
Prof. Michela Meo
Prof. Guido Pagana
Dott. Rafael Fontana

Candidate
Simone Terranova

Academic Year 2024-2025

Acknowledgements

I would like to express my sincere gratitude to all those who have contributed to the completion of this Master's thesis. Their support, guidance, and encouragement have been invaluable throughout this academic journey.
First and foremost, I extend my deepest appreciation to my supervisors, for their insightful guidance, unwavering support, and profound knowledge. Their expertise and meticulous feedback were instrumental in shaping this research and overcoming numerous challenges. This thesis would not have been possible without their dedication and mentorship.
My heartfelt thanks go to my family and friends for their endless encouragement, patience, and understanding. Their belief in me has been a constant source of motivation.
Finally, I would like to acknowledge LINKS Foundation for their support and assistance during the research process. Their contributions, whether direct or indirect, have been greatly appreciated.

Abstract

The accurate determination of the position of individuals and objects in indoor environments represents a significant challenge, given the ineffectiveness of traditional Global Navigation Satellite Systems (GNSS) in such contexts. This thesis addresses this problem by designing, implementing, and experimentally evaluating an indoor positioning system (IPS) based on Bluetooth Low Energy (BLE) technology, ESP32 microcontrollers, and wearable tags. The main objective is to provide a reliable, precise, and energy-efficient solution for real-time tracking, with particular attention to applications in healthcare.
The developed system adopts a hierarchical architecture that integrates wearable BLE tags (specifically Global Tag Disk Beacons) as mobile transmitters and a network of ESP32 microcontrollers as intelligent scanning/receiving stations. Communication is managed via MQTT protocols for data exchange between ESP32s and the backend, WebSocket for real-time updates to the frontend, and RESTful APIs for resource management. Positioning algorithms are based on RSSI-weighted multilateration techniques, with the integration of advanced filtering techniques such as Exponential Moving Average (EMA) and Kalman filter to improve the accuracy and robustness of the system in dynamic environments.
The research methodology included a thorough literature review, system design, firmware development, software development and a rigorous experimental validation phase conducted in a controlled 90 m^2 domestic environment. The results demonstrated that the system achieved room-level positioning accuracy with an average error of 1.32 meters. The system showed exceptional energy efficiency, allowing BLE tags a battery life of over six months with a 1000 ms advertising interval and remarkable operational stability, with zero crashes recorded during two weeks of continuous testing. The average zone transition latency was 25 seconds, with an 8% false alarm rate.
This research contributes a practical and low-cost indoor positioning solution that effectively balances the requirements of accuracy, energy efficiency, and implementation complexity. Although the system has some limitations in highly mobile scenarios and a sensitivity to electromagnetic interference, its robustness and energy efficiency make it particularly suitable for continuous monitoring applications. Future directions include the integration of inertial sensors to further reduce positioning error, the development of adaptive propagation models based on machine learning, and the exploration of hybrid architectures for greater scalability and precision.

Contents

List of Figures

List of Tables

List of Acronyms

AFH Adaptive Frequency Hopping.

AoA Angle of Arrival.

AoD Angle of Departure.

API Application Programming Interface.

BLE Bluetooth Low Energy.

BR/EDR Basic Rate/Enhanced Data Rate.

CAN Controller Area Network.

CNN Convolutional Neural Network.

CORS Cross-Origin Resource Sharing.

CPU Central Processing Unit.

CR2032 Lithium Coin Cell Battery type.

CRC Cyclic Redundancy Check.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

DoS Denial of Service.

EID Ephemeral ID.

EKF Extended Kalman Filter.

EMA Exponential Moving Average.

ERD Entity-Relationship Diagram.

ESP-IDF Espressif IoT Development Framework.

ESP32 A specific System-on-Chip microcontroller.

GDPR General Data Protection Regulation.

GNSS Global Navigation Satellite System.

GPIO General-Purpose Input/Output.

GPS Global Positioning System.

HIPAA Health Insurance Portability and Accountability Act.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

Hz Hertz.

I2C Inter-Integrated Circuit.

I2S Inter-IC Sound.

IDE Integrated Development Environment.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IMU Inertial Measurement Unit.

IoT Internet of Things.

IP Ingress Protection.

IPS Indoor Positioning System.

ISM Industrial, Scientific, and Medical radio band.

JS JavaScript.

JSON JavaScript Object Notation.

JWT JSON Web Token.

k-NN k-Nearest Neighbors.

KB Kilobyte.

LBS Location-Based Services.

LED	Light Emitting Diode.
LiFi	Light Fidelity.
MAC	Media Access Control address.
MB	Megabyte.
Mbps	Megabits per second.
MCU	Microcontroller Unit.
MPE	Mean Positioning Error.
MQTT	Message Queuing Telemetry Transport.
MQTTS	MQTT Secure.
MTBF	Mean Time Between Failures.
NFC	Near Field Communication.
NLOS	Non-Line-of-Sight.
NoSQL	Not only SQL.
ORM	Object-Relational Mapper.
OTA	Over-The-Air.
PCB	Printed Circuit Board.
QoS	Quality of Service.
RAM	Random Access Memory.
RBAC	Role-Based Access Control.
RDBMS	Relational Database Management System.
REST	Representational State Transfer.
RF	Radio Frequency.
RFID	Radio-Frequency Identification.
RMSE	Root Mean Square Error.
RNN	Recurrent Neural Network.
ROI	Return on Investment.

ROM	Read-Only Memory.
RSSI	Received Signal Strength Indicator.
RTC	Real-Time Clock.
RTT	Round-Trip Time.
SDIO	Secure Digital Input Output.
SDK	Software Development Kit.
SLAM	Simultaneous Localization and Mapping.
SoC	System-on-Chip.
SPA	Single Page Application.
SPI	Serial Peripheral Interface.
SQL	Structured Query Language.
SQLite	A specific SQL database engine.
SRAM	Static Random-Access Memory.
SSRN	Social Science Research Network.
SVG	Scalable Vector Graphics.
SVM	Support Vector Machine.
TCO	Total Cost of Ownership.
TCP	Transmission Control Protocol.
TDoA	Time Difference of Arrival.
TLM	Telemetry.
TLS	Transport Layer Security.
ToA	Time of Arrival.
TX	Transmit/Transmission.
UART	Universal Asynchronous Receiver-Transmitter.
UI	User Interface.
UID	Unique Identifier.
ULP	Ultra-Low Power.

UPS	Uninterruptible Power Supply.
URL	Uniform Resource Locator.
USB	Universal Serial Bus.
UUID	Universally Unique Identifier.
UWB	Ultra-Wideband.
UX	User Experience.
VLC	Visible Light Communication.
Wi-Fi	Wireless Fidelity.
WSS	WebSockets Secure.

Chapter 1

Introduction

1.1 Context and Motivation

In the contemporary digital era, the ability to accurately determine the position of individuals and objects has assumed fundamental importance in numerous application contexts. While Global Navigation Satellite Systems (GNSS), such as the Global Positioning System (GPS), have revolutionized positioning in outdoor environments (Figure 1.1), their effectiveness is significantly compromised in indoor settings (Figure 1.2) due to signal attenuation through building structures and the presence of electromagnetic interference [1]. This inherent limitation has motivated the development of alternative technologies specifically designed for indoor positioning, where individuals spend an average of 80-90% of their daily time.

Figure 1.1: Outdoor positioning

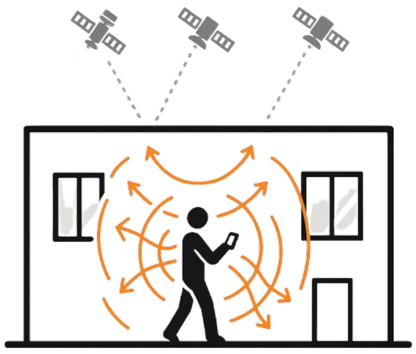


Figure 1.2: Indoor positioning

The rapid expansion of Location-Based Services (LBS) has underscored the critical need for reliable and accurate indoor positioning technologies. These services, which include indoor navigation, emergency response, resource tracking, and industrial process optimization, depend heavily on precise localization data to deliver context-aware functionalities to users [2]. In industrial environments shaped by the principles of *Industry 4.0*, the

ability to monitor the real-time location of assets, equipment, and personnel is fundamental. Such spatial awareness enables the coordination of complex operations, enhancing overall efficiency, operational flexibility, and inter-system collaboration—key factors in achieving the objectives of mass customization.
The recent proliferation of Internet of Things (IoT) devices has created a ubiquitous connectivity fabric, enabling scalable Indoor Positioning Systems (IPS). Among the wireless radios available, Bluetooth Low Energy (BLE) stands out for its low power draw, adequate indoor range, and native support in virtually all smartphones. These characteristics make BLE the technology of choice for continuous user and asset tracking without frequent battery replacements.
At the heart of many IoT endpoints lies the ESP32 microcontroller, which integrates both BLE and Wi-Fi radios, offers sufficient processing power for real-time localization algorithms, and remains cost-effective. Combined with compact, wearable BLE tags (for example, bracelet-style beacons), the ESP32 device can deliver a seamless user experience: devices stay with the user at all times, report position continuously, and survive for days on a single charge. [3].
This thesis addresses that gap by designing, implementing, and experimentally evaluating a fully integrated BLE-based IPS focusing on:

- **Precision and Robustness:** Achieving sub-meter accuracy in challenging indoor layouts with obstacles and multipath.
- **Energy Efficiency:** Optimizing firmware and hardware choices to extend battery life to multiple days under continuous use.
- **Real-World Validation:** Deploying the system in an apartment setting to measure performance under realistic conditions.

Through a rigorous methodology that encompasses literature review, system design, firmware development, and field trials, this thesis sets the stage for broader adoption of wearable BLE-based tracking in domains from healthcare to smart manufacturing and emergency management.

1.2 Research Objectives

This research aims to design, implement, and evaluate a robust indoor positioning system using Bluetooth Low Energy (BLE), ESP32 microcontrollers, and wearable tags. Specifically, the objectives are:

1. **Choose wearable BLE tags:** Bracelet-style BLE devices must ensure comfort, signal stability, and long battery life. These wearables must be robust and capable of transmitting consistent signals suitable for precise localization. Studies such as [4] highlight the value of integrating BLE into wearable form factors for continuous user tracking and acceptance.
2. **Develop ESP32-based receiver infrastructure:** Build a network of ESP32 nodes capable of detecting and processing BLE signals. The ESP32 is selected for

its built-in BLE and Wi-Fi support, and cost-effectiveness—qualities demonstrated in recent works like [5].

3. **Implement and optimize positioning algorithms:** Design localization algorithms based on RSSI-weighted multilateration, with integration of adaptive filters and potential application of machine learning techniques. This will enhance positioning accuracy and robustness in dynamic indoor conditions, as also advocated by [1].

4. **Evaluate system accuracy and robustness:** Conduct tests in indoor environments, analyzing performance using standardized metrics such as mean error, precision, and latency. These tests will define the operational boundaries and highlight areas for refinement.

5. **Benchmark against alternative technologies:** Compare the BLE-based system with existing solutions including Wi-Fi, UWB, RFID, and inertial tracking. This comparison will provide a comprehensive perspective on trade-offs, helping to position the developed system within the broader technological landscape [1].

6. **Assess real-world applicability:** Explore implementation in scenarios like patient monitoring, considering practical aspects such as system scalability, integration with existing infrastructure, data privacy and security concerns.

Through these objectives, this thesis contributes to the advancement of user-centric, wearable indoor positioning systems, offering a viable solution for real-time tracking with high precision, minimal energy consumption, and practical deployability.

1.3 Research Methodology

To achieve the stated research objectives, a structured and iterative methodology has been adopted. This approach combines theoretical analysis, engineering design, prototyping, and experimental validation, ensuring scientific rigor and practical relevance in the development of an indoor positioning system based on BLE technology, ESP32 microcontrollers, and wearable devices.
The methodological framework is organized into four interdependent macro-phases, each defined by specific objectives, activities, and expected outcomes. This structure supports a systematic approach to the inherent complexity of indoor positioning systems and guarantees the reproducibility and traceability of the research process.

Phase 1: Literature Review and Technology Mapping

The first phase involves a comprehensive review of scientific literature, technical documentation, and industrial solutions related to indoor positioning. Particular focus is placed on recent works (2018–2024) concerning BLE communication, ESP32 devices, and wearable IoT devices. This review is organized along three thematic axes: (i) technologies and algorithms for indoor positioning, (ii) applications of BLE and ESP32 in smart environments, and (iii) wearable system design for location tracking.

This phase results in a structured mapping of existing approaches, identification of research gaps, and the definition of benchmarking criteria for evaluating the proposed system. As noted by [1], understanding the variety of positioning techniques and their suitability in complex indoor environments is essential for developing robust and innovative systems.

Phase 2: System Design and Architectural Definition

Building on the findings of the literature review, this phase focuses on the conceptual and detailed design of the indoor positioning system. A user-centered design approach guides the definition of functional and non-functional requirements. The overall system architecture is developed, detailing both the wearable tags and the ESP32-based receiver infrastructure, considering factors such as spatial coverage, connectivity, and energy efficiency.
Particular attention is given to the selection and adaptation of positioning algorithms, balancing accuracy, computational complexity, and robustness. Inspired by recent studies on wearable BLE integration [4], user acceptability is also incorporated into design considerations. Outputs of this phase include technical specifications, system architecture diagrams, and algorithm flowcharts.

Phase 3: Prototype Development and Integration

This phase involves the incremental implementation of the system, allowing for early validation and iterative refinement of design choices. Development covers both hardware (ESP32 receivers) and software components, including communication modules, data processing pipelines, and user interfaces.
Following best practices in software engineering, the implementation emphasizes modularity, scalability, and maintainability. As highlighted by [2], integrating hardware and software components is a critical challenge in indoor positioning and requires a multidisciplinary approach. The phase culminates in functional prototypes capable of real-time indoor positioning and visualization.

Phase 4: Experimental Evaluation and Performance Assessment

The final phase consists of a structured evaluation of the developed system. Experiments are designed using rigorous protocols and focus on key performance indicators such as accuracy, precision, latency, robustness, energy efficiency, and scalability. Comparative analysis with existing systems further contextualizes the results.
Following guidelines from recent work on ESP32-based positioning [5], evaluation employs statistical methods and includes both quantitative metrics and qualitative user feedback. This phase validates the effectiveness and applicability of the proposed solution.

Cross-cutting Considerations

Throughout all phases, ethical research principles are applied, with particular attention to user privacy, data protection, and transparency in result dissemination. As emphasized by [1], these aspects are especially critical in systems involving wearable devices and location tracking.

1.4 Thesis Structure

This thesis comprises nine chapters and accompanying appendices, organized into three parts:

- **Part I: Foundations**
  - Chapter 1: Introduction (context, objectives, methodology)
  - Chapter 2: State-of-the-Art (survey of IPS technologies)
- **Part II: System Development**
  - Chapter 3: BLE Technologies (protocols, positioning techniques)
  - Chapter 4: ESP32 Platform (hardware, software frameworks)
  - Chapter 5: Wearable BLE Tags (design, ergonomics, firmware)
  - Chapter 6: System Architecture & Algorithms (integration, optimizations)
- **Part III: Validation and Conclusions**
  - Chapter 7: Implementation & Testing (prototyping, test protocols)
  - Chapter 8: Results & Discussion (performance analysis, comparative evaluation)
  - Chapter 9: Conclusions & Future Work (contributions, limitations, outlook)

Appendix A provides the pseudocode of this thesis.

Chapter 2

State of the Art in Indoor Positioning Systems

2.1 Overview of Indoor Positioning Technologies

Over the past decade, IPS have undergone significant evolution, catalyzed by the growing demand for location-based services in environments where traditional satellite navigation technologies are ineffective. As highlighted in Chapter 1, the need to accurately determine the position of people and objects in indoor contexts represents a complex technological challenge, which has stimulated the development of multiple approaches and solutions. This chapter aims to critically analyze the state of the art in the field of indoor positioning systems, examining the main available technologies, the most common positioning techniques, commonly adopted evaluation metrics, emerging applications, and still-open challenges.

Research in the field of indoor positioning systems is characterized by a notable heterogeneity of approaches, each with specific strengths and limitations. This diversity reflects the intrinsic complexity of indoor environments, characterized by obstacles, electromagnetic interference, variations in architectural structure, and dynamics of space utilization that significantly influence signal propagation and, consequently, the accuracy of positioning systems [1]. Unlike outdoor positioning, where GPS represents a universally accepted standard, the indoor context requires more articulated and often customized solutions based on specific application needs.

Indoor positioning technologies can be classified according to various criteria, including the type of signal used, the required infrastructure, the achievable accuracy, power consumption, and implementation costs. A fundamental distinction concerns the use of radio frequency (RF) based technologies versus non-RF ones. The former include Wi-Fi, Bluetooth Low Energy (BLE), Ultra-Wideband (UWB), Radio-Frequency Identification (RFID), and cellular network-based systems, while the latter comprise optical, ultrasonic, magnetic, and inertial systems [2]. Each of these technologies has distinctive characteristics that make it more or less suitable for specific application contexts. A comparison of common RF-based indoor positioning technologies is presented in Table 2.1.

Table 2.1: Comparison of Common RF-Based Indoor Positioning Technologies

Technology	**Accuracy (m)**	**HW Cost**	**Range (m)**	**Power Consumption**	**Infrastructure**	**NLOS Robustness**
Wi-Fi	5-15	Medium	50-100	Medium	Existing	Good
BLE	1-8	Low	10-70	Very Low	Dedicated	Good
UWB	0.1-0.5	High	10-50	High	Dedicated	Excellent
RFID (Active)	3-30	Medium (Tags High)	Up to 100	Medium (Tags)	Dedicated	Good
RFID (Passive)	0.5-5	Very Low (Tags)	Up to 10	N/A (Tags)	Dedicated	Poor/Fair

In recent years, the attention of the scientific and industrial community has particularly focused on RF technologies, thanks to their ability to operate in Non-Line-of-Sight (NLOS) conditions and the possibility of leveraging pre-existing communication infrastructures. Among these, Bluetooth Low Energy has gained particular relevance due to its low power consumption, compatibility with most modern mobile devices, and the low cost of the necessary hardware. Concurrently, technologies like Ultra-Wideband have opened new perspectives in terms of localization accuracy, achieving accuracies in the order of centimeters, albeit at higher implementation costs [6].
The choice of the most appropriate technology for an indoor positioning system depends on multiple factors, including accuracy requirements, the operating environment, the number of users or objects to be tracked, energy constraints, and the available budget. In many cases, hybrid approaches that combine different technologies represent the optimal solution, allowing for a balance between accuracy, cost, and implementation complexity. As emphasized by [7], each technology presents specific advantages and limitations that must be carefully evaluated in relation to the application context.
In addition to the choice of technology, a crucial aspect in the development of indoor positioning systems concerns the algorithmic techniques used to determine the position. These include approaches based on trilateration, triangulation, fingerprinting, and proximity, each with specific characteristics in terms of accuracy, robustness, and computational complexity.

2.2 Indoor Positioning Techniques

Indoor positioning systems utilize various algorithmic techniques to determine the position of objects or people within enclosed environments. These techniques, which form the methodological core of IPS systems, can be classified into different categories based on the mathematical and physical principles they rely on. The choice of the most appropriate technique depends on multiple factors, including the hardware technology used, accuracy requirements, environmental characteristics, and computational constraints. This section will analyze the main indoor positioning techniques, with particular attention to their operating principles, advantages, limitations, and application areas. A summary comparison of these techniques is provided in Table 2.2.

Table 2.2: Comparison of Indoor Positioning Techniques

Technique	**Principle**	**Typical Accuracy**	**Complexity**	**Infrastructure**	**Calibration**	**NLOS Robustness**
Trilateration	Distance	Variable (e.g., 1-10m for RSSI)	Medium	Reference points	Minimal	Fair
Triangulation	Angles	Variable (e.g., <1m for AoA with UWB/BLE 5.1)	High	Directional Antennas	Moderate	Fair
Fingerprinting	Signal Fingerprints	1-5m	High (offline), Medium (online)	Reference points	Extensive	Good
Proximity	Vicinity	Zone-level	Low	Beacons/ Readers	Minimal	Variable

2.2.1 Multilateration

Multilateration is one of the most common positioning techniques and is based on measuring the distance between the device to be localized and at least three reference points with known coordinates [8]. Knowing these distances, the device's position is determined as the intersection point of at least three spheres (in three dimensions) or at least three circles (in two dimensions), each centered at the reference point with a radius equal to the measured distance.

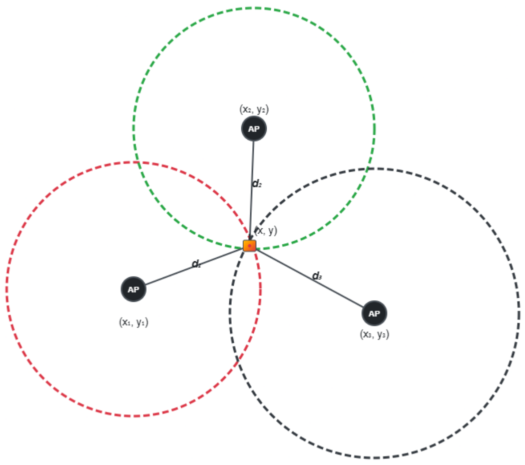


Figure 2.1: Multilateration technique with 3 Stations

In indoor positioning contexts based on wireless technologies, distance is typically estimated through signal parameters such as the Received Signal Strength Indicator (RSSI), Time of Arrival (ToA), or Time Difference of Arrival (TDoA). RSSI, in particular, is widely used in systems based on Bluetooth Low Energy and Wi-Fi, where the received signal strength is converted into a distance estimate through signal propagation models. The mathematical foundation of trilateration, a multilateration technique utilizing three positioning stations, can be formalized through a system of equations. Considering a two-dimensional environment, if (x_i, y_i) represent the coordinates of reference point i and d_i the estimated distance between this point and the device to be localized with coordinates (x, y), the following system is obtained:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = d_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = d_2^2 \\ (x - x_3)^2 + (y - y_3)^2 = d_3^2 \end{cases} \tag{2.1}$$

Solving this system provides the device's coordinates. In three-dimensional environments, a fourth equation and a fourth reference point are needed to determine the z coordinate as well.
Despite its apparent conceptual simplicity, the practical implementation of trilateration in indoor environments presents several challenges. The main one concerns the accuracy

of distance estimation, which can be compromised by phenomena such as signal reflections, diffractions, and attenuations due to obstacles, electromagnetic interference, and environmental variations. As highlighted by [9], these factors can introduce significant errors in position estimation, especially when using technologies like BLE, whose signal is particularly sensitive to environmental conditions.

To mitigate these problems, several variations and optimizations of basic trilateration have been proposed. These include the use of filtering algorithms such as the Kalman filter or particle filter, which allow for the integration of successive measurements and reduce the impact of noise, and the adoption of adaptive signal propagation models that account for the specific characteristics of the environment [10]. Furthermore, integrating data from inertial sensors or other sources can help improve the overall accuracy of the system.

Trilateration finds application in numerous indoor positioning contexts, especially in combination with technologies such as BLE, UWB, and Wi-Fi. Its relative implementation simplicity and ability to provide continuous position estimates make it particularly suitable for applications requiring real-time tracking of moving people or objects.

2.2.2 Triangulation

Triangulation is a positioning technique that, unlike trilateration, is based on measuring angles rather than distances. Specifically, the position of a device is determined by measuring the Angle of Arrival (AoA) of the signal emitted by the device itself with respect to two or more receivers with known positions [11].

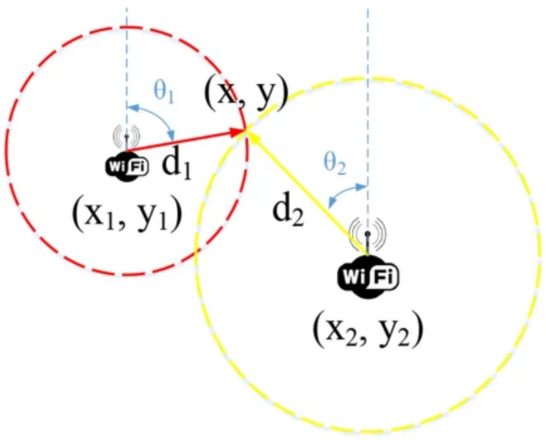


Figure 2.2: Triangulation technique

The mathematical principle behind triangulation can be illustrated by considering a two-dimensional environment with two receivers positioned at points (x_1, y_1) and (x_2, y_2). If θ_1 and θ_2 represent the angles of arrival of the signal with respect to a reference direction, the position (x, y) of the device can be determined as the intersection point of the two lines originating from the receivers at the measured angles.

In mathematical terms, this translates to solving the following system of equations:

$$\begin{cases} y - y_1 = \tan(\theta_1) \cdot (x - x_1) \\ y - y_2 = \tan(\theta_2) \cdot (x - x_2) \end{cases} \tag{2.2}$$

In three-dimensional environments, triangulation requires measuring both the azimuth and elevation of the signal, and at least three receivers to uniquely determine the position. The measurement of the angle of arrival can be achieved through various techniques, including the use of directional antenna arrays or measuring the phase difference of the signal received by multiple antennas. These techniques require specialized hardware and are generally more complex than measuring the signal strength used in RSSI-based trilateration.

Triangulation potentially offers higher accuracy than trilateration under ideal conditions, as angle measurement is less affected by signal attenuation due to distance. However, in complex indoor environments, characterized by multiple reflections and multipath phenomena, accurately determining the angle of arrival can be problematic. As highlighted by [12], the presence of obstacles and reflective surfaces can cause significant distortions in the apparent direction of signal arrival, compromising positioning accuracy.

Recent technological developments have led to the introduction of solutions that improve the applicability of triangulation in indoor contexts. For example, Bluetooth 5.1 introduced native support for angle of arrival determination, enabling more efficient and accurate implementations of triangulation in BLE-based systems [6]. Similarly, technologies like Ultra-Wideband offer advanced angular resolution capabilities due to their high bandwidth.

Triangulation is applied in various indoor positioning contexts, especially in scenarios where high accuracy is required and where the infrastructure can support the specialized hardware needed for angle measurement. It is particularly effective in environments with a direct Line-of-Sight (LoS) between transmitter and receiver, while its effectiveness decreases in the presence of obstacles and multiple reflections.

2.2.3 Fingerprinting

Fingerprinting represents a radically different approach compared to geometric techniques like trilateration and triangulation. Instead of relying on mathematical models to calculate position, fingerprinting adopts an empirical approach based on recognizing characteristic signal patterns at different locations in the environment [13].

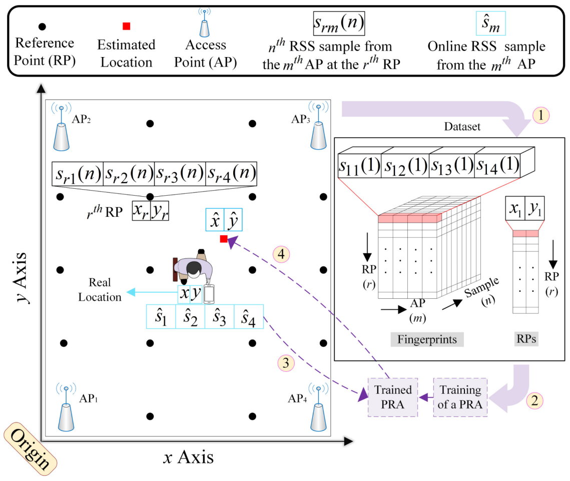


Figure 2.3: Fingerprinting technique

This technique is divided into two main phases: an offline calibration phase and an online positioning phase. During the offline phase, a database of signal "fingerprints" is created by collecting and recording signal characteristics (typically RSSI from multiple base stations) at known points in the environment. This process, known as "site survey" or "radio mapping," requires a considerable initial investment in terms of time and resources but forms the basis for subsequent positioning.

In the online phase, real-time measured signal characteristics are compared with those stored in the database, and the position is estimated by identifying the most similar fingerprint or by interpolating between multiple nearby fingerprints. This comparison can be performed using various techniques, from the simplest ones based on Euclidean distance in the feature space, to more sophisticated methods using machine learning algorithms such as k-nearest neighbors (k-NN), support vector machines (SVM), artificial neural networks, or probabilistic algorithms like the Bayesian filter [14].

Fingerprinting offers several advantages over geometric techniques. Firstly, it does not require explicit modeling of signal propagation, thus being more robust in complex environments where phenomena like reflections, diffractions, and attenuations would render theoretical models inaccurate. Secondly, it can exploit the peculiarities of the environment, such as interference and reflections, as distinctive elements that enrich the uniqueness of the fingerprints, rather than considering them as disturbing factors.

On the other hand, fingerprinting also has significant limitations. The need for an extensive calibration phase represents a significant challenge for its implementation in large-scale environments or those subject to frequent changes. Furthermore, changes in the

environment (such as moving furniture, the variable presence of people, or structural modifications) can significantly alter signal characteristics, rendering the fingerprint database obsolete and requiring periodic recalibrations.
To address these limitations, recent research has focused on approaches that reduce the burden of calibration, such as crowdsourcing techniques that distribute the data collection process among multiple users, or automatic calibration methods that leverage inertial sensors and other information sources to dynamically update the fingerprint database. Moreover, the application of advanced deep learning techniques has shown promising results in improving the robustness of fingerprinting to environmental variations [15].
Fingerprinting is particularly suitable for technologies like Wi-Fi and BLE, which are widely available in indoor environments and provide easily accessible RSSI measurements. It has been successfully applied in various contexts, from shopping malls to hospitals, from airports to university buildings, demonstrating an accuracy typically in the order of a few meters.
An interesting variant of traditional fingerprinting is magnetic fingerprinting, which exploits distortions of the Earth's magnetic field caused by metal structures and electrical installations in buildings. This technique does not require dedicated communication infrastructures, relying solely on the magnetic sensors present in most modern smartphones, but generally offers lower accuracy than solutions based on RF technologies [13].

2.2.4 Proximity

The proximity-based positioning technique is the simplest approach among those discussed, but it offers significant advantages in terms of implementation simplicity and low costs. Unlike previous techniques, which aim to determine the precise coordinates of an object or person, proximity positioning limits itself to detecting the vicinity of the device to be localized to one or more known reference points [10].

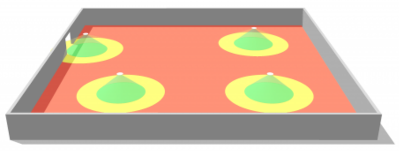


Figure 2.4: Proximity technique

The operating principle is extremely intuitive: when a mobile device enters the coverage radius of a fixed sensor or transmitter (such as a BLE beacon, an RFID reader, or a Wi-Fi access point), its position is associated with that of the reference point. The granularity of positioning depends on the density of reference points and their coverage radius: with a high density of short-range sensors, relatively precise localization can be

achieved, while with long-range sensors, a coarser localization is obtained but with more extensive coverage.
In formal terms, if $P = \{p_1, p_2, ..., p_n\}$ represents the set of reference points with known positions, and $R = \{r_1, r_2, ..., r_n\}$ their respective coverage radii, the estimated position $\hat{p}$ of a device can be determined as:

- $\hat{p} = p_i$ if the device is detected by one reference point.
- $\hat{p} = f(p_{i1}, p_{i2}, ..., p_{ik})$ if the device is detected by multiple reference points.

where f represents an aggregation function that can be simply the selection of the reference point with the strongest signal, or a more complex operation such as calculating the weighted centroid based on signal strength.
The proximity technique finds natural application with technologies such as RFID, NFC (Near Field Communication), and BLE beacons, which are intrinsically designed to operate over limited distances. In particular, BLE beacons configured in iBeacon or Eddystone mode represent a particularly popular solution for implementing proximity-based positioning systems, thanks to their low cost, ease of installation, and compatibility with most modern mobile devices.

Table 2.3: Advantages and Limitations of Proximity Positioning

Advantages	**Limitations**
Does not require complex algorithms or extensive calibrations	Provides discrete rather than continuous position information
Can be realized with inexpensive and widely available hardware	To achieve fine localization, a large number of sensors is necessary
The system can be easily expanded by adding new reference points	Proximity detection can be compromised by interferences

Proximity positioning is adequate for numerous applications that do not require precise localization but rather the identification of the zone or area where the user or object is located. Typical examples include access control systems, proximity marketing applications in retail, presence monitoring in specific areas of a building, and contextual notification systems.
An interesting evolution of the proximity technique is represented by systems that combine multiple technologies to improve positioning accuracy and robustness. For example, integrating BLE beacons with inertial sensors present in smartphones allows for the implementation of "proximity-enhanced dead reckoning" solutions, where the position estimate based on inertial data integration is periodically corrected when the user approaches a known reference point [1].

2.2.5 Hybrid and Advanced Techniques

The positioning techniques discussed so far each have specific strengths and limitations, making them more or less suitable for particular application contexts. In practice, many indoor positioning systems adopt hybrid approaches that combine different techniques to leverage their respective advantages and compensate for their weaknesses. Furthermore, recent developments in artificial intelligence and machine learning have led to the emergence of advanced techniques that overcome the limitations of traditional approaches.

Data Fusion and Multi-Technology Approaches

Data fusion is a fundamental paradigm in modern indoor positioning systems, enabling the integration of information from different sources to improve positioning accuracy, robustness, and continuity. This integration can occur at different levels:

- **Fusion of positioning techniques**: Combination of different algorithmic techniques, such as the joint use of trilateration and fingerprinting, where the former provides an initial estimate that is then refined by the latter.
- **Fusion of hardware technologies**: Integration of data from different communication technologies, such as BLE, Wi-Fi, and UWB, to exploit the complementary characteristics of each.
- **Fusion with inertial sensors**: Combination of radio frequency-based positioning techniques with data from accelerometers, gyroscopes, and magnetometers, implementing "pedestrian dead reckoning" (PDR) solutions that allow tracking movement even in the absence of RF signals.
- **Fusion with contextual information**: Integration of positioning data with information about the environment, such as building maps, movement constraints (e.g., inability to pass through walls), and user behavior models.

The implementation of data fusion requires specific algorithms that determine how to combine the different sources of information, taking into account their reliability, precision, and availability. Among the most common approaches are the Kalman filter and its variants (such as the extended or unscented Kalman filter), the particle filter, and methods based on fuzzy logic or Bayesian networks [10].

Techniques based on Machine Learning and Deep Learning

The application of machine learning and deep learning techniques for indoor positioning has opened new perspectives, allowing for the resolution of problems that were difficult to address with traditional approaches. These techniques can be applied in different phases of the positioning process:

- **Improvement of fingerprinting**: Use of deep neural networks for the recognition of complex patterns in signal fingerprints, improving robustness to temporal variations and reducing the need for frequent recalibrations.
- **Adaptive modeling of signal propagation**: Employment of learning algorithms to develop propagation models that automatically adapt to the specific characteristics of the environment, improving the accuracy of trilateration.

- **Activity and context recognition**: Use of classification techniques to identify user activity (walking, running, climbing stairs, etc.) and the environmental context, information that can be used to improve position estimation.
- **End-to-end positioning**: Implementation of solutions that directly map raw signal characteristics to position, bypassing intermediate feature extraction phases and the application of geometric algorithms.

A particularly promising example is the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) for fingerprinting, which have demonstrated superior capabilities in handling the temporal and spatial variability of signal characteristics compared to traditional approaches [14].

Techniques based on Computer Vision

An emerging research area involves the integration of computer vision techniques into indoor positioning systems. These solutions use cameras (mounted in the environment or on the mobile device) and image processing algorithms to determine position through the recognition of visual landmarks, motion estimation based on visual features (visual odometry), or augmented reality techniques like SLAM (Simultaneous Localization and Mapping).
Vision-based techniques potentially offer high accuracy and do not require dedicated communication infrastructures, but they present significant challenges in terms of computational costs, user privacy, and dependence on lighting conditions. Despite these limitations, the integration of visual information with other positioning techniques represents a promising research area, especially in contexts where high accuracy is required or where traditional RF technologies are ineffective [6].

2.3 Evaluation Metrics for Indoor Positioning Systems

The performance evaluation of indoor positioning systems is a fundamental aspect both in research and in real-world application contexts. The adoption of standardized and well-defined metrics allows for objective comparison of different technological solutions, guides design choices, and verifies the fulfillment of application requirements. This section will analyze the main metrics used to evaluate the performance of indoor positioning systems, with particular attention to their meanings, measurement methods, and relevance in different application contexts. An overview of these metrics is presented in Table 2.4.

Table 2.4: Main Evaluation Metrics for Indoor Positioning Systems (IPS)

Metric	**Description**	**Typical Units / Considerations**
Accuracy	Average positioning error	m, cm
Precision	Repeatability of estimates	std. dev. of error (m, cm)
Latency	Delay in position estimation	ms, s
Update Rate	Number of estimates per second	Hz
Availability	Percentage of time/area with active service	%
Reliability	Consistency of performance over time	MTBF, Robustness to interference
Scalability	Ability to handle increased users/area	No. of users, Area (m^2, km^2)
Energy Efficiency	Energy consumption per estimate/device	mW, Joule/estimate, Battery life (hours)
Costs	HW, SW, installation, maintenance costs	EUR, USD, TCO
Privacy and Security	Data protection, consent, attack resilience	Policies, GDPR compliance, Security protocols

2.3.1 Accuracy and Precision

Accuracy and precision are the most commonly used metrics to evaluate the performance of an indoor positioning system, although the two terms are often confused or used interchangeably in non-specialist literature.
Accuracy refers to the closeness between the position estimated by the system and the actual position of the object or person. In formal terms, accuracy can be defined as the average positioning error, calculated as the Euclidean distance between the estimated and actual position:

$$\text{Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (z_i - \hat{z}_i)^2}$$

where (x_i, y_i, z_i) represents the actual position and $(\hat{x}_i, \hat{y}_i, \hat{z}_i)$ the estimated position for the i-th test point, and n is the total number of test points.

Precision, on the other hand, refers to the repeatability or consistency of measurements, i.e., the dispersion of position estimates when the measurement is repeated under the same conditions. A common metric to quantify precision is the standard deviation of the positioning error:

$$\text{Precision} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i - \bar{e})^2}$$

where e_i represents the positioning error for the i-th test point, and $\bar{e}$ is the average error. As highlighted by [16], a localization system is characterized by a precision appropriate to the scale the system itself aims for: in the case of satellite geo-positioning, a precision of the order of meters can be considered excellent, while for indoor positioning applications, a precision in the order of sub-meters might be required.
It is important to note that accuracy and precision are not necessarily correlated: a system can be precise (providing consistent results in repeated measurements) but not accurate (with a systematic error with respect to the actual position), or vice versa. Ideally, a positioning system should be both accurate and precise.
In addition to the average error and standard deviation, other commonly used metrics to characterize accuracy include:

- **Median error**: Less sensitive to outliers than the mean, it provides an indication of the system's "typical" performance.
- **Percentile error** (e.g., 75th or 90th percentile): Indicates the error value below which a certain percentage of measurements fall, providing information about the tail of the error distribution.
- **Maximum error**: Represents the worst-case scenario and can be critical in applications where even a single significant error could have serious consequences.
- **Cumulative Distribution Function (CDF) of the error**: Provides a complete representation of the error distribution, allowing visualization of the probability that the error is below a certain value.

2.3.2 Latency and Update Rate

The temporal dimension of an indoor positioning system's performance is primarily characterized by two metrics: latency and update rate.
Latency refers to the time delay between the moment an object or person is at a certain position and the moment the system provides the estimate of that position. In real-time positioning systems, high latency can significantly compromise usability, especially in applications requiring responsiveness, such as indoor navigation or tracking of rapidly moving objects.
The overall latency of a positioning system can be broken down into several components:

- **Acquisition latency**: The time required to acquire data from sensors or receive signals from transmitters.

- **Transmission latency**: The time required to transmit the acquired data to the processing unit.
- **Processing latency**: The time required to process the data and calculate the position.
- **Communication latency**: The time required to communicate the calculated position to the user or the system that will use it.

The update rate indicates the number of position estimates provided by the system per unit of time (typically expressed in Hz). A high update rate is particularly important for tracking rapidly moving objects or for applications requiring continuous and smooth position monitoring.
As emphasized by [17], the choice of the optimal update rate depends on the application context and often represents a trade-off with other system parameters, such as power consumption or computational complexity. For example, in applications tracking slowly moving assets, an update rate of 0.1-1 Hz might be sufficient, while for tracking moving people or for augmented reality applications, frequencies of 10 Hz or higher might be necessary.

2.3.3 Availability and Reliability

The availability of an indoor positioning system refers to the percentage of time or area in which the system is capable of providing position estimates with an acceptable level of accuracy. This metric is particularly relevant in real-world application contexts, where system coverage might not be uniform due to infrastructure limitations or environmental characteristics affecting signal propagation.
Availability can be quantified in several ways:

- **Temporal availability**: Percentage of time the system provides valid position estimates.
- **Spatial availability**: Percentage of the target area where the system can operate with acceptable performance.
- **Functional availability**: Percentage of positioning requests the system can satisfy within certain accuracy and latency constraints.

Reliability, closely related to availability, refers to the system's ability to maintain consistent performance over time and under different operating conditions. A reliable system should provide position estimates with a predictable and stable level of accuracy, even in the presence of environmental variations, interference, or partial infrastructure failures.
Specific metrics for evaluating reliability include:

- **Rate of false positives and false negatives**: In zone-based or region-based systems, the percentage of times the system erroneously indicates the presence or absence of an object in a given area.

- **Mean Time Between Failures (MTBF)**: The average time of correct operation between successive system failures.

- **Robustness to interference**: The system's ability to maintain acceptable performance in the presence of electromagnetic interference or other sources of disturbance.

As highlighted by [18], reliability is a critical factor for the adoption of indoor positioning systems in real-world application contexts, especially in fields like safety or healthcare, where a system malfunction could have significant consequences.

2.3.4 Scalability

Scalability refers to the positioning system's ability to maintain adequate performance as the number of users, covered area, or device density increases. This metric is particularly relevant for large-scale implementations, such as shopping malls, airports, or university campuses, where the system might need to simultaneously manage hundreds or thousands of devices.
Scalability can be assessed along several dimensions:

- **Scalability with respect to the number of users**: Ability to handle an increasing number of devices to be located without significant degradation in performance in terms of accuracy, latency, or availability.

- **Spatial scalability**: Ability to extend system coverage to larger areas while maintaining consistent performance.

- **Infrastructure scalability**: Ease with which the system's infrastructure can be expanded to support more users or wider coverage.

As emphasized by [17], scalability is a significant challenge for many indoor positioning systems, especially those based on technologies requiring dedicated infrastructure or using shared communication channels. For example, systems based on Wi-Fi fingerprinting might experience performance degradation in densely populated environments due to network congestion, while UWB-based systems might require a large number of anchors to cover extensive areas, resulting in significant implementation costs.

2.3.5 Energy Efficiency

Energy efficiency is a crucial metric for indoor positioning systems, especially for those involving mobile devices or battery-powered sensors. Low power consumption is essential to ensure adequate battery life and reduce the need for frequent battery replacements or recharges.
Energy consumption can be assessed at different levels:

- **Energy consumption of the device to be located**: Energy required by the tag, smartphone, or other device that needs to be located.

- **Energy consumption of the infrastructure**: Energy needed to power the positioning infrastructure, such as anchors, access points, or environmental sensors.
- **Overall system energy consumption**: Sum of the energy consumed by all system components.

Energy efficiency is particularly critical for applications involving wearable devices or IoT sensors, where small size and battery life constraints impose severe limitations on power consumption. As highlighted by [13], technologies like Bluetooth Low Energy were specifically designed to offer low power consumption, making them particularly suitable for indoor positioning applications based on battery-powered devices.
Specific metrics for evaluating energy efficiency include:

- **Energy consumption per position estimate**: Energy required to obtain a single position estimate.
- **Operational autonomy**: Continuous operating time of the system with a given battery capacity.
- **Energy efficiency as a function of accuracy**: Relationship between energy consumption and positioning accuracy, which allows evaluating trade-offs between these two dimensions.

2.3.6 Costs

Costs represent a fundamental metric for evaluating the practical feasibility of an indoor positioning system, especially in commercial or industrial contexts where return on investment is a critical decision factor. The costs of a positioning system can be divided into several categories:

- **Implementation costs**: Include the purchase of hardware (tags, anchors, sensors, servers), software development or purchase, and system installation and configuration.
- **Operational costs**: Comprise energy, maintenance, software and hardware updates, and technical staff for system management.
- **Scalability costs**: Incremental costs associated with expanding the system to cover larger areas or support more users.
- **Indirect costs**: Represent potential operational costs associated with disruptions during installation or maintenance, as well as the ones associated with staff training.

As highlighted by [13], the cost of a system, regardless of its precision, can determine its large-scale application. A limited cost combined with a sufficiently accurate system can be a decisive factor for adopting the technology in commercial or industrial contexts.
Cost assessment should consider not only the initial investment but also the Total Cost of Ownership (TCO) over the system's expected lifecycle. Furthermore, it is important to evaluate costs in relation to the expected benefits and the added value that the positioning system can provide in terms of operational efficiency, improved user experience, or new business opportunities.

2.3.7 Privacy and Security

Although privacy and security are not performance metrics in the strict sense, they represent critical aspects in the evaluation of an indoor positioning system, especially in contexts involving the tracking of people. A system's ability to protect location data and prevent unauthorized access or manipulation can be assessed through several dimensions:

- **Data protection**: Measures implemented to protect location data during transmission and storage, such as encryption, anonymization, or pseudonymization.
- **Access control**: Mechanisms to ensure that only authorized users can access location data or configure the system.
- **Transparency and consent**: Clarity in data collection and usage policies for location data, and mechanisms for obtaining users' informed consent.
- **Resilience to attacks**: The system's ability to withstand attempts at manipulation or disruption, such as signal spoofing or denial-of-service attacks.

As highlighted by [15], privacy concerns represent a significant obstacle to the adoption of these technologies in some contexts, especially those involving continuous tracking of people in environments like offices, hospitals, or shopping malls.
The evaluation of privacy and security measures should consider not only technical aspects but also legal and ethical ones, such as compliance with data protection regulations (e.g., GDPR in Europe) and the potential impact on user perception and social acceptance of the technology.

2.4 Applications of Indoor Positioning Systems

Indoor positioning systems have found application in a wide range of sectors, thanks to their ability to provide contextual information based on localization in environments where traditional satellite navigation technologies are ineffective. The growing maturity of indoor positioning technologies, coupled with reduced implementation costs and greater awareness of potential benefits, has led to an increasingly widespread adoption of these solutions in commercial, industrial, healthcare, and public service contexts (Table 2.5).

Table 2.5: Summary of Applications of Indoor Positioning Systems

Application Sector	**Key Use Cases**	**Typical Technologies Required**
Healthcare	Navigation, asset/patient tracking, workflow optimization, infection control	BLE, Wi-Fi, UWB (for high accuracy), RFID
Retail and Large-Scale Distribution	Proximity marketing, customer flow analysis, inventory management, personalized offers	BLE, Wi-Fi, NFC, Vision
Logistics and Industry	Asset/vehicle tracking, process automation, worker safety, warehouse management	UWB, RFID (Active/Passive), BLE, Wi-Fi
Public Spaces and Smart Cities	Navigation (airports, stations), crowd management, personalized experiences (museums), accessibility	Wi-Fi, BLE, LiFi, Computer Vision, Cellular
Emergency and Security	Emergency call localization, evacuation guidance, first responder coordination, access control	Wi-Fi, BLE, UWB, Cellular Networks

2.4.1 Applications in the Healthcare Sector

The healthcare sector represents one of the most promising application areas for indoor positioning systems, due to the complexity of hospital environments and the criticality of the operations performed therein. Key applications in this sector include:
Navigation and orientation within healthcare facilities: As highlighted by [19], large hospitals and healthcare complexes are complex environments where patients, visitors, and staff can easily become disoriented. Indoor positioning systems enable the implementation of wayfinding solutions that guide users to specific destinations such as wards, outpatient clinics, diagnostic services, or emergency exits. These solutions can be integrated into mobile apps or information kiosks, significantly improving user experience and reducing appointment delays due to orientation difficulties.
Tracking of medical equipment: Healthcare facilities possess numerous expensive and critical pieces of equipment, such as defibrillators, infusion pumps, ventilators, and monitors, which are frequently moved between different areas. Indoor positioning systems allow for the rapid localization of this equipment when needed, improving operational efficiency, reducing search times, and optimizing resource utilization. As reported by [20],

this application can lead to significant cost savings and improvements in the quality of care, especially in emergency situations where timeliness is crucial.
Monitoring of at-risk patients: Patients with certain conditions, such as dementia, cognitive disorders, or a risk of falls, can benefit from monitoring systems based on indoor positioning technologies. These systems, typically implemented through bracelets or other wearable devices, allow healthcare staff to receive alerts when a patient enters unsafe areas or exhibits anomalous movement patterns that could indicate a fall or health issue. This application significantly contributes to patient safety and reduces the workload for nursing staff.
Optimization of workflows and operational analysis: Positioning data of healthcare staff and patients can be analyzed to identify inefficiencies in workflows, areas of congestion, and opportunities for improvement in space layout. This analysis can guide targeted interventions to optimize routes, reduce waiting times, and improve the allocation of human resources. Furthermore, in clinical research contexts, positioning data can provide valuable information on patient movement patterns in relation to their health status and response to treatments.
Infection control and contact tracing: As emerged during the COVID-19 pandemic, indoor positioning systems can play a crucial role in controlling hospital-acquired infections through contact tracing and the identification of potential exposures. These systems allow for the retrospective reconstruction of movements and interactions of patients and staff, facilitating the implementation of targeted preventive measures and outbreak management.
The specific needs of the healthcare sector in terms of positioning technologies include high accuracy (especially for applications like patient monitoring), robust reliability, ease of integration with existing hospital information systems, and strict adherence to privacy and data security, in compliance with regulations such as GDPR and HIPAA. Technologies like BLE, thanks to their low cost, ease of implementation, and compatibility with mobile devices, have found particular diffusion in this sector, especially for wayfinding and asset tracking applications.

2.5 Current Challenges and Limitations

Despite significant progress in recent years in the field of indoor positioning systems, numerous technical, methodological, and economic challenges persist (Table 2.6), limiting the widespread adoption and effectiveness of these technologies in real-world contexts. Understanding these challenges is fundamental not only for guiding future research but also for directing implementation choices in concrete projects, allowing for the adoption of appropriate strategies to mitigate existing limitations.

Table 2.6: Main Challenges and Limitations of Indoor Positioning Systems (IPS)

Challenge / Limitation	**Brief Description**
Accuracy and Reliability in Complex Environments	Impact of multipath, NLOS, interference, environmental variability. Difficulty in achieving consistent high precision.
Energy Consumption and Battery Life Constraints	Balancing performance (accuracy, update rate) with battery life of mobile devices and infrastructure components (e.g., beacons).
Implementation Costs and Scalability	High costs of hardware, installation, site surveys, and ongoing maintenance. Difficulties in expanding coverage to large or multiple areas cost-effectively.
Privacy, Security, and Ethical Considerations	Protection of sensitive location data, ensuring informed consent, mitigating surveillance risks, and complying with regulations like GDPR.
Standardization and Interoperability	Lack of universal standards leading to technological fragmentation, proprietary solutions, and difficulties in integrating systems from different vendors.

Indoor positioning systems represent a dynamic and continuously evolving research field, characterized by the convergence of different disciplines such as telecommunications, electronics, computer science, and artificial intelligence. The in-depth understanding of the different technologies, techniques, metrics, applications, and challenges presented in this chapter provides a solid foundation for the design and implementation choices that will be discussed in the subsequent chapters.

Chapter 3

Bluetooth Low Energy Technologies for Indoor Positioning

3.1 Fundamentals of Bluetooth Low Energy

3.1.1 Introduction to Bluetooth Low Energy

Bluetooth Low Energy (BLE), introduced with the Bluetooth 4.0 specification in 2010, is a wireless communication technology designed for applications requiring extremely low energy consumption while maintaining adequate communication range. Unlike classic Bluetooth (BR/EDR), BLE was designed from the ground up to support sporadic, small data transmissions, making it particularly suitable for Internet of Things (IoT) applications and indoor positioning systems [21].
BLE has emerged as one of the most promising solutions for indoor positioning, offering an optimal balance between performance, energy efficiency, and device ubiquity. According to [22], the market for BLE-based indoor positioning solutions has experienced significant growth, with applications spanning retail, healthcare, logistics, and smart buildings.
This chapter analyzes BLE's technical characteristics that make it suitable for indoor positioning applications, examining beacon protocols, distance estimation methods, positioning algorithms, and comparing advantages and limitations with other technologies.

3.1.2 BLE Architecture and Technical Specifications

Bluetooth Low Energy differs from classic Bluetooth through architectural and technical features specifically designed to minimize energy consumption. While both operate in the 2.4 GHz ISM band, BLE presents substantial differences in network topology, channel management, packet format, and connection modes.

Network Topology

BLE supports two main network topologies:

1. **Point-to-point connection**: Establishes bidirectional communication between two devices with master (central) and slave (peripheral) roles, enabling reliable and secure data exchange through a preliminary connection phase.

2. **Broadcasting**: Allows unidirectional data transmission to any receiver within range without establishing connections, forming the basis for BLE beacon operation in indoor positioning applications.

The Bluetooth standard evolution has introduced more complex topologies. Version 4.1 added simultaneous master-slave capabilities across different connections, while version 5.0 introduced mesh networking support for many-to-many communications [23].

Channel Management and Frequency Hopping

BLE operates on 40 channels in the 2.4 GHz band, each with 2 MHz bandwidth, divided into:

- 3 advertising channels (37, 38, 39) for device discovery, connection establishment, and broadcast transmissions

- 37 data channels for bidirectional communication after connection establishment

To mitigate interference from other 2.4 GHz technologies (Wi-Fi, classic Bluetooth), BLE implements Adaptive Frequency Hopping (AFH). During connections, devices change channels at each interval following a pseudo-random pattern, avoiding high-interference channels [24].

Packet Format and Transmission Rate

BLE packets are optimized for efficient small-data transmission with the following structure:

- Preamble (1 byte)

- Access Address (4 bytes)

- Packet Header (2 bytes)

- Length (1 byte)

- Payload (0-27 bytes in v4.0/4.1, up to 255 bytes in later versions)

- CRC for error control (3 bytes)

Transmission rates are 1 Mbps in versions 4.0-4.1, with version 5.0 introducing 2 Mbps modes and long-range options (125 kbps and 500 kbps) for extended-distance communications.

Power Saving Mechanisms

BLE achieves extremely low energy consumption through several strategies:

1. **Duty cycling**: Devices spend most time in ultra-low power sleep mode, briefly waking for data transmission/reception
2. **Variable connection intervals**: Configurable synchronization intervals (7.5 ms to 4 s) with sleep mode between communications
3. **Low-frequency advertising**: Configurable advertising intervals up to several seconds, drastically reducing average consumption
4. **Rapid transmissions**: Minimized active transmission duration through optimized packet format and protocols

These features enable devices with small button-cell batteries (CR2032) to operate for months to years depending on configuration. According to [25], typical BLE beacons operate up to 24 months on a single battery, making this technology ideal for wearable tags in indoor positioning applications.

3.1.3 Evolution of Bluetooth Standard for Positioning

The Bluetooth standard has undergone significant evolution, with progressive improvements expanding indoor positioning capabilities. Table 3.1 summarizes the evolution of positioning capabilities across different Bluetooth versions.

Table 3.1: Evolution of positioning capabilities in different Bluetooth standard versions

Version	Year	Relevant Features for Positioning	Accuracy
4.0	2010	Introduction of BLE, RSSI	5-10 meters
4.1	2013	Improvements in coexistence	5-10 meters
4.2	2014	Improved security, extended payload	3-8 meters
5.0	2016	Increased range, speed, and broadcasting capacity	2-5 meters
5.1	2019	Direction Finding (AoA/AoD)	0.5-2 meters
5.2-5.4	2020-2023	Incremental optimizations	0.5-2 meters

3.1.4 Key Features for Indoor Positioning

BLE possesses several characteristics making it particularly suitable for indoor positioning applications with wearable tags:

- **Low Energy Consumption**: According to [26], typical BLE tags operate 1-2 years on a single button-cell battery through optimized packet structures, adjustable transmission power (-20 dBm to +10 dBm), and efficient sleep modes with nanoampere-level currents.

- **Mobile Device Compatibility**: BLE's ubiquity in smartphones and tablets enables positioning solutions without dedicated hardware, allowing devices to function as receivers or scanners with native iOS and Android APIs.
- **Adequate Indoor Range**: Provides 10-30 meter range in typical environments (up to 100+ meters with Bluetooth 5.0), enabling coverage of medium-sized spaces with reasonable infrastructure density.
- **Cost-Effective Implementation**: Low costs compared to Ultra-Wideband or camera-based systems. As noted by [27], BLE beacons cost €10-15 per unit while ESP32-based receivers cost under €15 each.
- **Interference Robustness**: Despite operating in the shared 2.4 GHz band, BLE implements Adaptive Frequency Hopping, strategically distributed advertising channels, and redundant transmissions to mitigate Wi-Fi and Bluetooth interference.

These features make BLE particularly suitable for indoor positioning applications requiring low-power wearable tags. The following sections examine beacon protocols, distance estimation methods, and BLE-specific positioning algorithms.

3.2 BLE Beacon Protocols

Bluetooth Low Energy (BLE) beacons are widely adopted transmitters that periodically broadcast data packets enabling receiver devices to detect presence and estimate distance. Since Apple's introduction of iBeacon in 2013, BLE beacons have gained popularity across retail, healthcare, logistics, and smart buildings due to their simplicity, low cost, and broad compatibility.
BLE beacons operate in two primary configurations: fixed beacons with mobile receivers (navigation/proximity marketing) and fixed receivers with mobile beacons (asset tracking). The latter configuration, relevant for wearable tag-based positioning systems, offers advantages in privacy management and energy efficiency.

3.2.1 iBeacon Protocol

Apple's iBeacon [28], introduced in 2013, was the first commercially successful BLE beacon standard. An iBeacon packet contains a standard BLE prefix (9 bytes) and a 21-byte payload including:

- UUID (16 bytes): Identifies beacon owner/application
- Major (2 bytes): Identifies beacon group
- Minor (2 bytes): Identifies individual beacons
- Measured Power (1 byte): Calibrated signal strength at 1-meter distance

iBeacon provides simplicity, native iOS support, and energy efficiency, but offers limited functionality and restricted Android support due to its proprietary nature.

3.2.2 Eddystone Protocol

Google's Eddystone (2015) provides an open-source alternative with enhanced flexibility through multiple frame types:

- Eddystone-UID: Transmits unique identifier (Namespace + Instance ID)
- Eddystone-URL: Broadcasts compressed URLs
- Eddystone-TLM: Conveys telemetry data (temperature, battery)
- Eddystone-EID: Transmits periodically changing identifiers for security

Eddystone offers open-source flexibility, cross-platform support, and advanced features like URL broadcasting and telemetry, but with increased implementation complexity and potentially higher energy consumption.

3.2.3 AltBeacon Protocol

Radius Networks' AltBeacon (2014) serves as an open-source, vendor-independent alternative. Its 24-byte payload contains a flexible 20-byte ID structure, beacon code, reference RSSI, and reserved field. While offering openness and flexibility, AltBeacon has lower commercial adoption and lacks native platform support.

3.2.4 Protocol Comparison

Table 3.2 compares the three protocols across key features.

Table 3.2: Comparison of BLE beacon protocols

Feature	**iBeacon**	**Eddystone**	**AltBeacon**
Developer	Apple	Google	Radius Networks
License	Proprietary	Open-source	Open-source
Year introduced	2013	2015	2014
ID structure	UUID+Major+Minor	Namespace+Instance	Flexible (20 bytes)
Frame types	Single	Multiple	Single
iOS support	Native	Libraries	Libraries
Android support	Libraries	Native	Libraries
Special features	Regional monitoring	URLs, telemetry, EIDs	Flexible ID
Security	Basic	Advanced	Basic
Energy usage	Very low	Low-medium	Very low
Adoption	High	Medium	Low

Protocol selection depends on target platform requirements, functional needs, security considerations, energy constraints, and required flexibility. For wearable BLE tag-based positioning systems, iBeacon represents an appropriate choice due to its simplicity, energy efficiency, and broad hardware compatibility including ESP32 platforms. The distance estimation techniques and positioning algorithms discussed in subsequent chapters remain applicable across all protocols with appropriate adaptations.

3.3 BLE-based Distance Estimation Methods

3.3.1 Introduction

Accurate distance estimation between BLE devices is fundamental to indoor positioning systems. Distance estimation in BLE primarily relies on Received Signal Strength Indicator (RSSI) analysis, supported by all BLE devices. Recent Bluetooth 5.1+ standards introduced Direction Finding features (AoA/AoD) for enhanced precision with specialized hardware.

3.3.2 RSSI-based Distance Estimation

RSSI measures received radio signal power (typically in dBm). In real environments, the log-distance path loss model is commonly used:

$$\text{RSSI}(d) = \text{RSSI}(d_0) - 10n \log_{10}(d/d_0) + X \tag{3.1}$$

where $\text{RSSI}(d)$ is signal strength at distance d, $\text{RSSI}(d_0)$ is reference strength at d_0 (typically 1m), n is the path loss exponent (2-4), and X represents shadowing effects. Distance is estimated by inverting the equation:

$$d = d_0 \times 10^{(\text{RSSI}(d_0) - \text{RSSI}(d))/(10n)} \tag{3.2}$$

RSSI-based estimation faces several challenges including signal variability, multipath fading, material absorption, antenna directionality, and hardware heterogeneity. These factors lead to estimation errors of 20-50% in complex indoor environments [25].

3.3.3 Accuracy Improvement Techniques

Three main approaches improve RSSI-based distance estimation:

1. **Calibration:** Adapts theoretical models to specific environments and devices through reference RSSI calibration, path loss exponent adjustment, environment-specific modeling, and cross-device compensation. Accurate calibration can significantly reduce estimation errors, especially in stable environments [29].

2. **Filtering:** Reduces RSSI variability using moving averages, low-pass filters, Kalman filters, or particle filters. Proper filtering can reduce RSSI variability by up to 70% [26].

3. **Data Fusion:** Combines information from multiple beacons, sensors (accelerometers, gyroscopes), temporal estimates, and contextual information (floor plans, obstacles). Hybrid approaches can reduce positioning error by 30-40% compared to RSSI-only methods [30].

3.3.4 Direction Finding Methods

Bluetooth 5.1+ introduced Direction Finding through AoA and AoD techniques. These methods measure phase differences across antenna arrays to determine signal direction.

- **Angle of Arrival (AoA):** Transmitter uses single antenna; receiver uses antenna array to determine signal origin direction.

- **Angle of Departure (AoD):** Transmitter uses antenna array; receiver measures phase differences with single antenna.

Direction Finding offers higher accuracy (angular precision of few degrees), reduced sensitivity to power variations, and positioning with fewer receivers. However, it requires specialized hardware, is sensitive to multipath effects, has limited consumer adoption, and consumes more energy [31].

3.3.5 Method Comparison

Table 3.3 compares BLE distance estimation methods.

Table 3.3: Comparison of BLE-based distance estimation methods

Feature	**Basic RSSI**	**Enhanced RSSI**	**Direction Finding**
Typical Accuracy	3–10 m	1–3 m	0.1–1 m
Hardware Requirements	Standard	Standard	Specialized
Implementation Complexity	Low	Medium	High
Robustness	Low	Medium	Medium–High
Energy Consumption	Very Low	Low	Medium
Cost	Very Low	Low	Medium–High
Technology Maturity	High	High	Medium
Consumer Device Support	Universal	Universal	Limited

Method selection depends on accuracy requirements, energy constraints, cost considerations, environmental characteristics, and compatibility needs. For wearable BLE applications, enhanced RSSI techniques with calibration and filtering often provide optimal balance between accuracy, energy consumption, and implementation complexity.

3.4 Positioning Algorithms for BLE Systems

Building upon the theoretical foundations presented in the State of the Art, this section evaluates the positioning algorithms most suitable for BLE-based indoor localization systems, focusing on their practical implementation and performance characteristics in real deployment scenarios [29].

3.4.1 Algorithm Selection and Performance Analysis

For BLE-based positioning systems, four primary algorithmic approaches demonstrate practical viability: trilateration using RSSI measurements, triangulation leveraging Bluetooth 5.1 Direction Finding capabilities, fingerprinting techniques, and probabilistic methods. Each approach presents distinct trade-offs between accuracy, computational complexity, and implementation requirements.

RSSI-based Trilateration remains the most widely adopted approach due to its compatibility with standard BLE hardware. Using distance estimates derived from signal strength measurements, trilateration solves the system of equations:

$$(x - x_i)^2 + (y - y_i)^2 = d_i^2, \quad i \in \{1,2,3\} \tag{3.3}$$

where (x_i, y_i) represent beacon coordinates and d_i the estimated distances. While computationally efficient, this method typically achieves 2-5 meter accuracy in indoor environments due to RSSI variability and multipath interference [16].

Direction Finding Triangulation exploits Angle of Arrival (AoA) or Angle of Departure (AoD) measurements available in Bluetooth 5.1. By determining angles θ_i from reference points, the position is computed as:

$$y - y_i = \tan(\theta_i) \cdot (x - x_i), \quad i \in \{1,2\} \tag{3.4}$$

This approach can achieve sub-meter accuracy (0.5-1 m) but requires specialized antenna arrays and increases system complexity and power consumption [31].
Fingerprinting Methods create location-specific radio signatures during an offline calibration phase, then match real-time measurements against this database. Advanced implementations using machine learning techniques can achieve 1-3 meter accuracy with robust performance in complex multipath environments, though they require extensive site-specific calibration [32].
Probabilistic Approaches model positioning as a Bayesian inference problem, applying:

$$P(\text{position}|\text{measurements}) \propto P(\text{measurements}|\text{position}) \times P(\text{position}) \tag{3.5}$$

Methods such as Kalman filtering and particle filtering excel in dynamic tracking scenarios, typically achieving 1-2 meter accuracy with superior temporal consistency [30].

3.4.2 Implementation Considerations

Table 3.4 summarizes the key performance characteristics relevant to BLE system design.

Table 3.4: BLE positioning algorithm comparison

Algorithm	**Accuracy**	**Hardware Req.**	**Calibration**	**Tracking**
RSSI Trilateration	2-5 m	Standard BLE	Moderate	Limited
AoA/AoD Triangulation	0.5-1 m	Antenna arrays	Low	Limited
Fingerprinting	1-3 m	Standard BLE	Extensive	Limited
Probabilistic	1-2 m	Standard BLE	Moderate	Excellent

For wearable BLE applications requiring balance between accuracy and power efficiency, a hybrid approach combining RSSI-based trilateration with probabilistic filtering offers optimal performance. This theoretical foundation guides the design of the BLE-based positioning system with wearable tags and ESP32 presented in Chapter 6, which implements these principles in a practical, cost-effective solution.

Chapter 4

ESP32 as a Station for Positioning Systems

4.1 ESP32 Architecture and Capabilities

4.1.1 Platform Overview

The ESP32, developed by Espressif Systems, represents a versatile and powerful micro-controller particularly suitable for IoT applications and indoor positioning systems. As the successor to the ESP8266, the ESP32 combines high performance, low power consumption, rich communication interfaces, and low cost, making it ideal for implementing RSSI-based positioning techniques discussed in Chapter 3 [33].

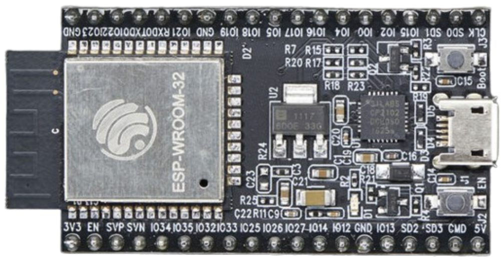


Figure 4.1: ESP32 microcontroller

In this research, ESP32 units serve as receiving/scanner stations that detect signals from wearable BLE tags, process RSSI data, and transmit information via MQTT to a centralized system for position determination. This configuration leverages the ESP32's native BLE scanning capabilities, processing power, and connectivity features.

4.1.2 Hardware and Software Architecture

The ESP32 features a 32-bit dual-core Xtensa LX6 processor operating at up to 240 MHz, significantly higher than other IoT microcontrollers. This computing power enables effective management of complex signal processing and data filtering algorithms crucial for BLE-based indoor positioning accuracy [34].

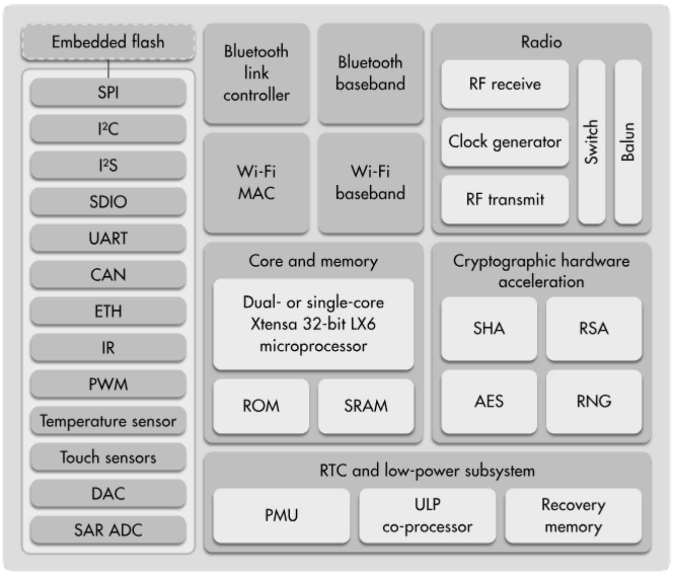


Figure 4.2: ESP32 Function Block Diagram

Memory specifications include 520 KB SRAM and up to 16 MB external flash, providing enough space for sophisticated applications and temporary data storage such as BLE device information during scanning operations. This generous capacity allows implementation of larger data processing buffers and complex filtering algorithms [35].
The ESP32 integrates Wi-Fi (802.11 b/g/n) and Bluetooth (Classic and BLE) modules managed by a dedicated coprocessor, enabling simultaneous operation of both technologies. This architecture allows background BLE scanning while the main processor handles data processing, improving overall system efficiency. The ESP32's ability to operate in Wi-Fi and Bluetooth modes simultaneously provides significant advantages for indoor positioning applications [34].
Additional features include comprehensive peripheral interfaces (SPI, I²C, UART, CAN, Ethernet MAC, SDIO) and multiple power-saving modes from active operation to deep-sleep with only ULP coprocessor active. These power management capabilities enable optimization based on application requirements, particularly relevant for battery-powered positioning systems [36].
Development Frameworks: The ESP32 supports various development approaches including ESP-IDF (official C framework with FreeRTOS), Arduino Core (simplified development), MicroPython (high-level scripting), and Mongoose OS (IoT-focused). For high-performance positioning applications requiring energy optimization, ESP-IDF or Arduino Core represent the most appropriate choices [37, 38].

4.2 ESP32 BLE Scanner Implementation

BLE scanning forms the foundation of the positioning system, enabling detection of advertising packets from wearable tags and extraction of MAC addresses and RSSI values for distance estimation.
Key implementation considerations include resource management for memory and power optimization, scan mode selection (passive vs. active scanning), and filtering mechanisms for relevant devices. Active scanning provides more complete data but increases power consumption and radio traffic [33].
Effective BLE scanning implementations require device filtering to eliminate irrelevant broadcasts, RSSI aggregation techniques for multiple measurements, and adaptive scanning parameters based on device density and interference levels. Advanced optimizations include density-aware and battery-conscious scanning algorithms, temporal RSSI filtering using exponential moving averages or Kalman filters, environmental calibration through empirical measurements, and strategic station placement for optimal coverage [35].

4.2.1 Software Implementation

The ESP32 scanner software integrates several key functionalities through a modular architecture:

- **Wi-Fi Management:** Multi-network support with automatic connection and fallback mechanisms.
- **MQTT Communication:** Efficient data transmission with retry mechanisms and broker fallback.
- **Tag Management:** Maintenance of authorized BLE tag lists with dynamic MQTT updates.
- **BLE Scanning:** Periodic 3-second active scans every 4 seconds, providing an optimal balance between comprehensive device detection, frequent data updates, and power consumption efficiency.
- **Remote Configuration:** Enables dynamic parameter updates (including connection settings, tag lists, and operational parameters) via MQTT protocol without requiring device reprogramming or firmware updates.

The BLE scan implementation uses Arduino Core libraries with custom callbacks to process detected devices. The callback extracts MAC addresses and RSSI values, filters for known tags, constructs JSON messages containing ESP32 ID, RSSI, timestamp, and tag ID, then publishes via MQTT with retry mechanisms.
Communication Protocol: MQTT provides reduced overhead, publish/subscribe decoupling, and configurable quality of service suitable for distributed IoT applications. Messages use structured JSON format on topics like `hospital/ble/scan` for data and `hospital/ble/config` for control [35].

Chapter 5

Wearable BLE Tags: Technology, Integration, and Considerations

5.1 Overview of Wearable BLE Tags

Wearable BLE tags are key to indoor positioning, using low-power signals detected by fixed receivers to balance range, battery life, and compatibility [7].

5.1.1 Classification and Applications

Wearable BLE tags can be classified by form factor and functionality as shown in Table 5.1.

Table 5.1: Classification of wearable BLE tags

Category	**Type**	**Characteristics**
Form Factor	Disk/Coin	Circular, 20–35mm diameter, easily integrated into accessories [39]
	Credit Card	85.6×54mm, larger battery capacity, corporate badges [40]
	Miniaturized	<20mm dimensions, limited battery life [41]
	Integrated	Smartwatches, fitness trackers with secondary BLE functionality [42]
Functionality	Passive	Basic advertising transmission, most economical [43]
	Interactive	Buttons, LEDs for user interaction [39]
	Sensorized	Additional sensors (accelerometer, temperature, humidity) [7]
	Programmable	Customizable firmware, maximum flexibility [37]

5.2 Design and Components

5.2.1 Hardware Architecture

Wearable BLE tags integrate several components in minimal space: a low-power ARM Cortex-M microcontroller managing firmware and protocols, an integrated BLE radio module (typically Bluetooth 5.0+), PCB or ceramic antenna, lithium coin cell battery (CR2032), power management circuitry, optional sensors, basic user interface elements, and protective enclosure [23, 43].

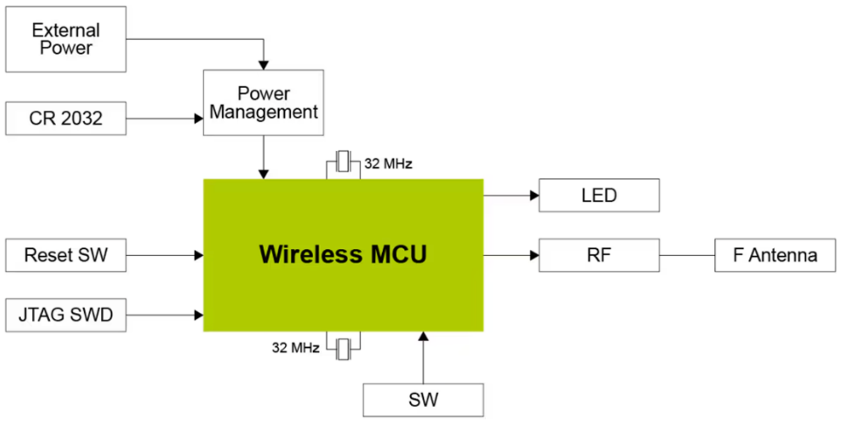


Figure 5.1: BLE Tag Function Block Diagram

5.3 Global Tag Disk Beacon Analysis

The Global Tag Disk Beacon exemplifies modern wearable BLE tag design with key specifications: 31mm diameter × 10mm thickness (8.2g weight), IP65-rated polycarbonate enclosure, Bluetooth 4.2 with configurable transmission power (up to 200m range), default broadcast interval (1s), CR2032 battery (1-year life at 1s interval), multifunction button with bi-color LED, and optional accelerometer/environmental sensors [39].

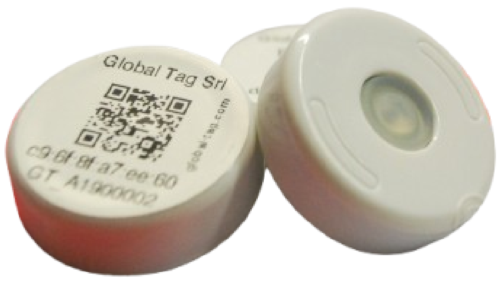


Figure 5.2: Global Tag Disk Beacon

5.3.1 Commercial Comparison

Table 5.2 compares leading commercial solutions, highlighting the Global Tag Disk Beacon's optimal balance for wearable applications.

Table 5.2: Comparison of commercial wearable BLE tags

Model	**Global Tag**	**Estimote Prox.**	**Kontakt Card**	**BlueCats AA**	**Minew i7**
Size/Weight	31×10mm 8.2g	55×55×15mm 15g	85.6×54×2.5mm 10g	42×42×11mm 30g	40×12mm 12g
Battery	1yr CR2032	2yrs CR2477	1-2yrs integrated	4yrs 2×AA	2yrs CR2450
Range/BLE	200m/5.0	70m/5.0	50m/4.2	300m/5.0	80m/5.0
Interface	Button+LED	None	None	Button+LED	Button+LED
Use Case	Wearable bracelets	Fixed install.	ID badges	Industrial	Personal badges

The Global Tag Disk Beacon's circular form factor, replaceable battery, complete user interface, and balanced specifications make it ideal for wearable bracelet integration.

5.4 Privacy and Security Considerations

5.4.1 Identified Challenges

Continuous indoor tracking presents distinct privacy and security vulnerabilities. Privacy risks encompass behavioral profiling, sensitive information inference, personal autonomy erosion, and unauthorized surveillance potential. Security vulnerabilities include unencrypted BLE transmissions, static identifiers enabling tracking, spoofing/cloning susceptibility, replay attacks, signal jamming, and device fingerprinting.

5.4.2 Comprehensive Mitigation Framework

A multi-layered approach addresses these challenges through three complementary strategies:

- **Privacy Protection** employs transparent informed consent processes, user control mechanisms, data minimization principles, and anonymization techniques with limited retention policies.
- **Security Enhancement** implements identifier rotation (MAC address changes, ephemeral identifiers), payload-level encryption, authentication requirements, role-based access control, RF anomaly monitoring, and spatio-temporal validation [23].
- **Privacy-by-Design Integration** establishes configurable positioning granularity, data segregation protocols, comprehensive audit trails, and automated data minimization processes.

5.4.3 Regulatory Compliance and Implementation Standards

GDPR compliance necessitates valid legal basis establishment, comprehensive privacy notices, data subject rights protection, Data Protection Impact Assessments, and breach notification procedures. Implementation standards include identity-positioning data separation, differentiated retention policies, right to disconnect provisions, regular impact assessments, ethical review processes, staff training programs, and robust incident response procedures.

These integrated practices demonstrate that effective indoor positioning systems can achieve high privacy protection and security standards while building essential user trust for successful technology adoption.

Chapter 6

Architecture and Implementation of the IoT System for Indoor Positioning

6.1 Chapter Objectives

This chapter presents the IoT-based indoor positioning system for real-time user tracking within buildings. The system integrates Bluetooth Low Energy (BLE) technology [44,45], distributed edge computing, and modern web technologies to address the complex challenges of accurate indoor localization [46, 47]. The architecture employs a multi-layered approach [48]: BLE beacons for signal acquisition, ESP32 microcontrollers as intelligent scanning stations, a Python backend for centralized processing, and a React frontend for visualization. By combining established positioning methodologies with modern IoT practices and containerized deployment, the implementation creates a robust and scalable solution tailored to meet the stringent requirements of healthcare environments. The chapter examines the technical decisions, algorithmic implementations, and architectural patterns that enable this comprehensive Indoor Positioning System (IPS) to deliver accurate, real-time location services.

6.1.1 System Requirements and Design Principles

The development of this indoor positioning system was guided by comprehensive functional and non-functional requirements that reflect the operational demands of the deployment scenario [49, 50]. The system must provide room-level positional accuracy sufficient for patient tracking applications. Real-time performance characteristics demand position updates with minimal latency to support emergency response scenarios and routine patient monitoring activities.

Reliability represents a cornerstone requirement [51], necessitating continuous operation with minimal downtime across all system components, from edge devices to backend services. The architecture must demonstrate scalability to accommodate growing numbers of BLE tags, scanning stations, and concurrent users without performance degradation.

Security considerations demand robust protection of sensitive patient location data through encrypted communication channels, secure authentication mechanisms, and comprehensive access controls. The system incorporates role-based authorization to ensure appropriate data access while maintaining compliance with healthcare privacy regulations.

Cost-effectiveness influences technology selection, favoring open-source frameworks and commercially available hardware components to enable broader adoption across different institutions.

Usability requirements emphasize intuitive interfaces that accommodate medical staff with varying technical expertise levels. The system provides clear visualization capabilities for floor maps, real-time tag locations, and system status information. Maintainability considerations drive the modular architecture design, facilitating straightforward debugging, updates, and component replacement through containerized deployment strategies.

6.2 System Architecture Overview

The indoor positioning system implements a hierarchical architecture that separates concerns across distinct functional layers while maintaining efficient data flow and communication pathways [48, 52]. This design approach enables independent scaling of system components while ensuring robust integration across the entire solution stack.

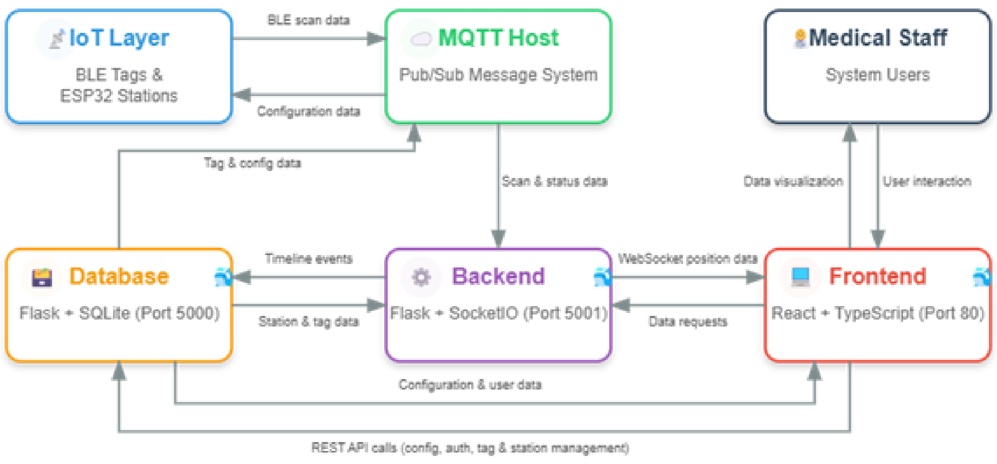


Figure 6.1: Comprehensive architectural diagram of the IoT indoor positioning system showing data flow between sensing layer, communication infrastructure, backend processing, and presentation components

The sensing layer forms the foundation of the system through strategically deployed BLE beacons and ESP32 scanning stations. BLE Disk Beacons manufactured by Global Tag serve as mobile transmitters attached to users, broadcasting unique identifiers through standardized advertising packets. ESP32 microcontrollers function as intelligent scanning stations that capture BLE signals, extract relevant data including Received Signal Strength Indicator (RSSI) values, and perform initial data processing before transmission to upstream systems.
The communication infrastructure facilitates reliable data transfer between distributed edge devices and centralized processing systems. Message Queuing Telemetry Transport (MQTT) protocol [53] serves as the primary communication backbone, providing lightweight publish-subscribe messaging capabilities optimized for IoT deployments [54]. The Wi-Fi network provides connectivity for ESP32 stations, enabling seamless integration with existing network infrastructure while maintaining appropriate security boundaries.
Backend processing systems implement the core positioning algorithms and data management functions. A Python-based processing engine subscribes to MQTT data streams, applies sophisticated filtering and positioning algorithms, and maintains persistent storage of historical location data. Flask framework provides RESTful API services for system

configuration and data retrieval, while integrated WebSocket capabilities [55, 56] enable real-time data distribution to frontend applications.

The presentation layer delivers comprehensive user interfaces through a React-based single-page application that provides real-time visualization of tag locations, system status monitoring, and administrative configuration tools. The frontend communicates with backend systems through standard HTTP APIs and maintains real-time connectivity through WebSocket connections for immediate position updates.

Deployment orchestration utilizes Docker containerization [57] to ensure consistent runtime environments across development and production deployments. Docker Compose manages multi-container applications, defining service dependencies, network configurations, and persistent storage volumes that support system reliability and maintainability.

6.3 Core Technologies and Communication Protocols

The system architecture implements a multi-layered communication framework that addresses distinct operational requirements: lightweight IoT messaging, real-time frontend updates, and comprehensive API services. Each protocol serves specific functions while maintaining seamless integration across the entire positioning system.

6.3.1 MQTT Implementation for IoT Communication

Message Queuing Telemetry Transport (MQTT) [53, 58] serves as the primary communication backbone between ESP32 scanning stations and the backend infrastructure. This lightweight publish-subscribe protocol is specifically optimized for resource-constrained IoT environments, providing reliable message delivery with minimal bandwidth overhead [51].

ESP32 devices publish BLE detection data to structured topic hierarchies using JSON formatting, enabling efficient data organization and selective subscription patterns. The backend system leverages the Python Paho-MQTT client to subscribe to these data streams, facilitating continuous ingestion of positioning measurements from distributed scanning stations.

Beyond data collection, MQTT enables bidirectional device management capabilities. Status reporting mechanisms allow ESP32 stations to communicate operational health and connectivity status through dedicated topics. Remote configuration management utilizes MQTT's publish mechanism to distribute Wi-Fi credentials, broker settings, and tag assignment parameters to deployed devices, eliminating the need for physical access during system updates.

The current development implementation utilizes public MQTT brokers, though production deployments require private broker infrastructure with comprehensive security measures including TLS encryption, client authentication, and role-based access control.

6.3.2 WebSocket Implementation for Real-Time Updates

WebSocket technology addresses the frontend's requirement for immediate position updates and system event notifications [59, 60]. The Flask-SocketIO integration provides

server-side WebSocket capabilities, while React frontend components utilize Socket.IO client libraries for connection management and real-time event handling [61].
The implementation broadcasts calculated tag coordinates through `new_position` events, ensuring all connected clients receive immediate location updates without polling overhead. System status events, including station connectivity changes and room entry/exit notifications, utilize dedicated WebSocket channels to maintain comprehensive operational awareness.
Client-side connection management is handled through the `WebSocketContext.tsx` component, which implements automatic reconnection logic and distributes events to React components. This architecture ensures that map visualization components receive instantaneous updates while maintaining connection resilience during network interruptions.

6.3.3 RESTful API Architecture

Representational State Transfer (REST) is an architectural style for designing networked applications that relies on stateless, client-server communication using standard HTTP methods (GET, POST, PUT, DELETE) and uniform resource identifiers (URIs) to perform operations on system resources.
The RESTful API layer [62] provides structured access to system resources and historical data through well-defined endpoints organized into functional categories. The Flask framework in `server2.py` implements this architecture with clear separation between authentication, resource management, and analytics services.
Authentication endpoints handle user lifecycle operations including registration, login, logout, and profile management. Security is enforced through bcrypt password hashing and session management. Resource management APIs implement complete CRUD operations for system entities such as tags, scanning stations, rooms, floors, and user accounts, following standard HTTP conventions with appropriate status codes and JSON formatting.
Analytics endpoints provide access to historical position data, timeline events, and system performance metrics. These services support both operational monitoring and retrospective analysis, enabling comprehensive system evaluation and optimization.
Data ingestion operates through a separate service (`app_w_db.py`) that maintains MQTT subscriptions and implements thread-safe queue architecture for message processing. This design decouples real-time data collection from API response handling, ensuring system responsiveness under varying load conditions.
Timeline event detection implements geofencing capabilities through point-in-polygon calculations, identifying room entry and exit events. These events generate database records and trigger WebSocket notifications, providing real-time situational awareness. Cross-Origin Resource Sharing (CORS) configuration ensures secure frontend-backend communication while maintaining appropriate security boundaries for web-based client access.

6.4 ESP32 Devices: The Intelligent Scanner Stations

ESP32 microcontrollers serve as intelligent scanning stations in our Indoor Positioning system [63]. These devices were selected for their integrated Wi-Fi, BLE capabilities, and sufficient processing power. The complete firmware implementation is available in `ESP32_SCRIPT.ino`, written in C++ for the Arduino/PlatformIO environment.

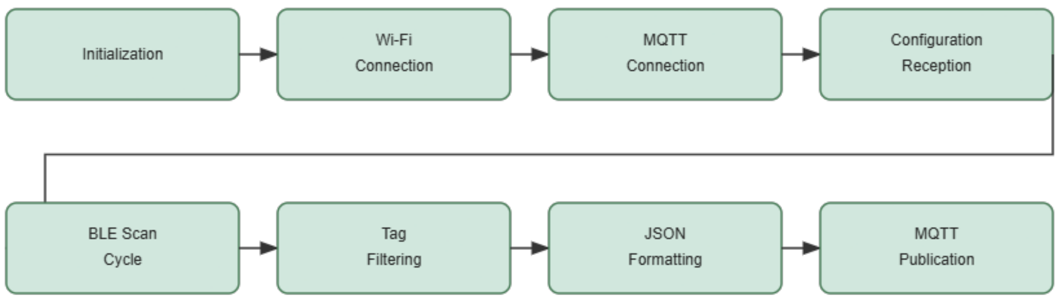


Figure 6.2: Flowchart of the firmware logic of the ESP32

6.4.1 Device Initialization and Network Connectivity

Upon startup, the ESP32 follows a systematic initialization process:

1. **Wi-Fi Connection**: The device initializes and connects to the Wi-Fi network using preconfigured credentials, implementing reconnection logic to maintain persistent connectivity.
2. **MQTT Broker Connection**: After establishing network connectivity, the device connects to the MQTT broker (e.g., `mqtt.eclipseprojects.io`) using the `PubSubClient` library.
3. **Topic Subscription**: The ESP32 subscribes to specific topics for receiving commands and configurations (e.g., `hospital/ble/config/<BOARD_ID>`) and implements a callback function (`callback_mqtt`) for processing incoming messages.

6.4.2 MQTT Communication Framework

MQTT serves as the primary communication backbone between ESP32 stations and the backend system [53,54]. This protocol was selected for its lightweight nature and publish-subscribe architecture, making it ideal for IoT applications in different environments [51].

ESP32 to Backend Communication occurs through two main topic channels:

- **RSSI Data** (Topic: `hospital/ble/scan`): ESP32s publish BLE tag detection data with JSON payloads containing `rssi` (signal strength), `mac` (tag identifier), `station_id` (ESP32 identifier), and `timestamp`. For example:

```
{
  "rssi": -75,
  "mac": "AA:BB:CC:DD:EE:FF",
  "station_id": "ESP32_01",
  "timestamp": 1678886400
}
```

- **Status Updates** (Topic: `hospital/ble/status/<BOARD_ID>`): ESP32 stations report their connection status and device information:

```
{
  "esp32_id": "50e22db5cd12",
  "status": "connected",
  "ssid": "iPhone di Simone",
  "ip": "192.168.1.105"
}
```

Status messages are published during initial connection, after reconnections, and periodically as heartbeats. This mechanism allows the backend service (`app_w_db.py`) to track active base stations, monitor connection status, and update the database accordingly.

Backend to ESP32 Communication enables remote management and configuration through:

- **Configuration Messages** (Topic: `hospital/ble/config`): The backend sends Wi-Fi and MQTT broker configuration data:

```
{
  "esp32_id": "all",
  "wifi_ssid": "AP1234",
  "wifi_password": "1234",
  "mqtt_server": "mqtt.eclipseprojects.io",
  "mqtt_port": 1883,
  "timestamp": 1745857658
}
```

- **Tag Configuration Messages** (Topic: `hospital/ble/tags`): The backend sends a list of BLE tags that ESP32 devices should scan for:

```
[
  {
    "mac": "c4:42:06:a5:66:13",
    "tag_id": "GT_AY02839",
    "patient_names": "John Doe"
  },
  {
    "mac": "f8:f2:c5:76:db:92",
    "tag_id": "GT_AY02840",
    "patient_name": "Jane Smith"
  }
]
```

By publishing to `hospital/ble/config_req/<BOARD_ID>`, ESP32 stations can request configuration, prompting the backend to send current settings. This on-demand retrieval mechanism ensures operational continuity after power cycles or network disruptions. Upon receiving tag configuration data, base stations store this information locally to filter BLE detections, processing only relevant tags. This approach optimizes both processing resources and network bandwidth while maintaining system efficiency.

6.4.3 BLE Scanning and Tag Detection

For BLE scanning and tag detection, the ESP32 utilizes native BLE APIs (`BLEDevice`, `BLEScan`). A custom callback class `MyAdvertisedDeviceCallbacks` is implemented with an `onResult` method that is invoked upon device detection. This callback performs several critical functions:

- Checks if the detected device matches a known tag from the configured list
- Extracts the device's MAC address and RSSI value
- Performs preliminary filtering to focus on relevant detections
- Prepares data for MQTT transmission to the backend

6.4.4 Future Implementation Considerations

Several critical technical considerations could be addressed in future iterations of the system:

- **Power Consumption**: Developing power management strategies to optimize consumption, particularly important for battery-operated deployments
- **Security**: Implementing MQTTS and proper ESP32 authentication mechanisms on the MQTT broker to ensure secure communication

6.5 Backend Architecture: The Central Nervous System

The backend serves as the core of the system, handling data processing, position calculations, and communications. Built primarily with Python and Flask, it consists of several integrated components working together to deliver real-time positioning capabilities.

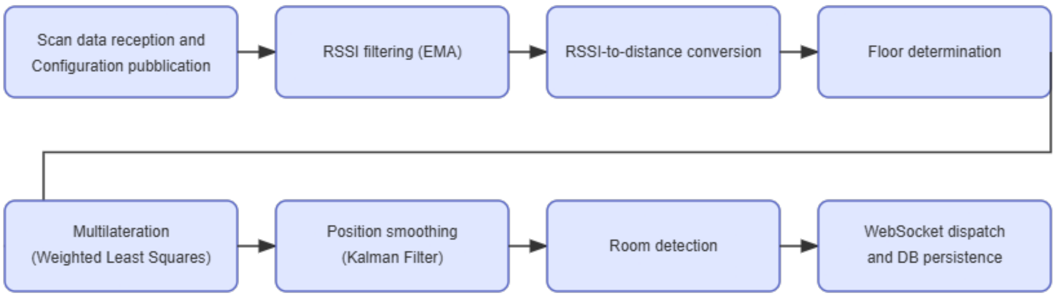

Scan data reception and Configuration pubblication
RSSI filtering (EMA)
RSSI-to-distance conversion
Floor determination
Multilateration (Weighted Least Squares)
Position smoothing (Kalman Filter)
Room detection
WebSocket dispatch and DB persistence

Figure 6.3: Flowchart of the data processing pipeline in the backend

6.5.1 Typical Data Workflow

1. ESP32 devices publish RSSI data via MQTT
2. Backend ingests the data into the thread-safe queue
3. Positioning algorithm processes the data through its pipeline
4. Calculated positions are:
    - Emitted to frontend clients via WebSocket
    - Checked against geofencing rules for potential events and then each event is saved into the databse

The backend effectively serves as the central nervous system, coordinating data flow between edge devices and the frontend while performing the complex calculations necessary for accurate indoor positioning.

6.5.2 Positioning Algorithm Pipeline

After MQTT's JSON messages are correctly ingested by the backend, the pipeline processes data through sequential stages in a dedicated thread:

- **RSSI Filter:** Applies Exponential Moving Average to smooth raw RSSI values [64, 65] using the formula:

$$RSSI_{filtered} = \alpha \cdot RSSI_{raw} + (1 - \alpha) \cdot RSSI_{filtered\_previous}$$

where α is the smoothing factor (default 0.2), giving more weight to historical data to smooth out rapid fluctuations.

- **RSSI-to-Distance Conversion:** Utilizes the log-distance path loss model [66,67] with configurable parameters (`A` and `N_PARAM`), present in the `rssi_to_distance` function:
$$d = 10^{\frac{RSSI_0 - RSSI_{filtered}}{10 \cdot n}}$$
where $RSSI_0$ is the reference RSSI at 1 meter (calibrated to -59 dBm in this system) corresponding to parameter `A`, and n is the path loss exponent (set to 2.2 for the hospital environment) corresponding to parameter `N_PARAM`. These parameters are crucial and should be empirically calibrated for the specific environment.

- **Floor Determination:** Identifies the most probable floor based on signal strengths. The `process_ble_scan` function groups stations by floor. For stations detecting the tag, a selection criterion might involve choosing stations whose RSSI is above a threshold (e.g., $RSSI_{threshold} = \max(RSSI) - 10\text{dBm}$). The most probable floor is then selected using:
$$F_{most\_probable} = \arg\max_{f} \sum_{s \in S_{top}} ⊮_{floor(s)=f} \cdot RSSI_s$$
where S_{top} is the set of selected strong-signal stations and $⊮_{floor(s)=f}$ is an indicator function that equals 1 when the station s is on floor f and 0 otherwise.

- **Multilateration:** When sufficient data is available (at least 3 non-collinear stations), calculates tag coordinates using Weighted Least-Squares Multilateration [68]. Each station with known coordinates (x_i, y_i) and estimated distance d_i provides an equation:
$$(x - x_i)^2 + (y - y_i)^2 = d_i^2$$
The system employs a weighted least-squares approach that minimizes:
$$\min_{x,y} \sum_{i=1}^{N} w_i \left(d_i - \sqrt{(x - x_i)^2 + (y - y_i)^2} \right)^2$$
with weights inversely proportional to the square of the estimated distance:
$$w_i = \frac{1}{\max(d_i, 0.1)^2}$$
giving higher importance to closer stations (stronger signals). This non-linear optimization problem is solved iteratively using SciPy's `least_squares`, starting from an initial guess. The residual function for each station i is:
$$r_i(x, y) = \sqrt{w_i} \cdot \left(d_i - \sqrt{(x - x_i)^2 + (y - y_i)^2} \right)$$

- **Kalman Filtering:** Applies a 2D Kalman Filter [69–71] to smooth positions, reduce noise, and predict future states. The filter tracks the tag's state vector, which includes position (x_k, y_k) and velocity $(v_{x,k}, v_{y,k})$ at time step k:

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ v_{x,k} \\ v_{y,k} \end{bmatrix}$$

  The filter operates in two recursive steps:

  1. **Prediction (Time Update):**

$$\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1|k-1}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k$$

  2. **Update (Measurement Update):** The innovation (measurement residual) $\mathbf{y}_k$ and innovation covariance $\mathbf{S}_k$ are:

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k \mathbf{x}_{k|k-1}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$$

     The optimal Kalman gain $\mathbf{K}_k$ is:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

     The updated state estimate $\mathbf{x}_{k|k}$ and updated covariance $\mathbf{P}_{k|k}$ are:

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}$$

  Implementation details include:

  – Initialization with $\mathbf{x}_{0|0} = \mathbf{0}$ and $\mathbf{P}_{0|0} = \mathbf{I}_4$

  – State transition matrix $\mathbf{F}_k$ (with $dt = 0.1$ default time step):

$$\mathbf{F}_k = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  – Process noise covariance $\mathbf{Q}_k$ (with default process variance $q = 1e-3$):

$$\mathbf{Q}_k = q \cdot \begin{bmatrix} \frac{dt^4}{4} & 0 & \frac{dt^3}{2} & 0 \\ 0 & \frac{dt^4}{4} & 0 & \frac{dt^3}{2} \\ \frac{dt^3}{2} & 0 & dt^2 & 0 \\ 0 & \frac{dt^3}{2} & 0 & dt^2 \end{bmatrix}$$

– Measurement matrix $\mathbf{H}_k$ (as only position is measured):

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

– Measurement noise covariance $\mathbf{R}_k$ (with default measurement variance $r = 1e - 1$):

$$\mathbf{R}_k = r \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r & 0 \\ 0 & r \end{bmatrix}$$

Each tracked tag has its own Kalman filter instance.

- **Room Entry/Exit Detection (Simple Geofencing):** Once a reliable position (x, y, floor) for a tag is obtained, the system determines if the tag is within a predefined room using a point-in-polygon check. This generates `TimelineEvent` entries for room entry and exit events.
- **Emission of the position to the frontend** The smoothened position is sent via WebSocket.

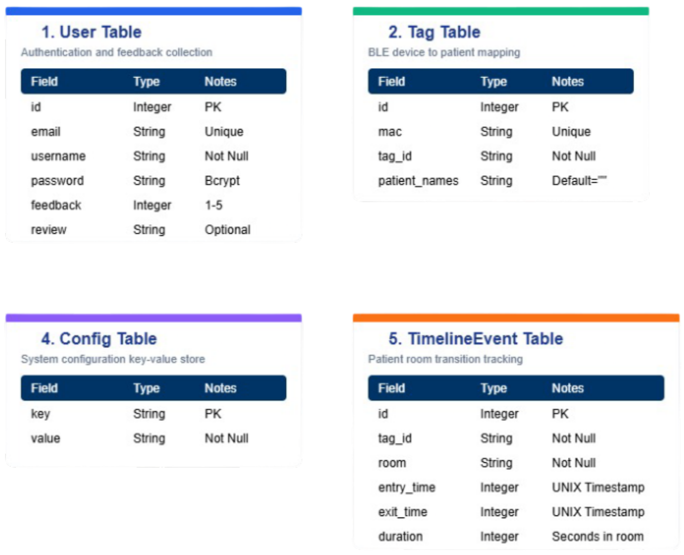
6.6 Data Persistence and Management

The system's data persistence layer is built upon SQLite [72], a lightweight yet robust database solution that provides an optimal balance between simplicity and functionality for the current implementation scale.

The database architecture leverages **SQLAlchemy** as the Object-Relational Mapping (ORM) framework [73], implemented through `db_w_add_remove2.py` and `create_db.py`. This abstraction layer enables seamless interaction between Python objects and database entities through well-defined model classes that map directly to database tables. The ORM approach enhances code maintainability while providing type safety and automatic relationship management between related entities.

6.6.1 Data Model and Schema Design

The database schema encompasses seven primary entities, each serving distinct functional requirements within the IoT ecosystem.



1. User Table
Authentication and feedback collection

Field	Type	Notes
id	Integer	PK
email	String	Unique
username	String	Not Null
password	String	Bcrypt
feedback	Integer	1-5
review	String	Optional

2. Tag Table
BLE device to patient mapping

Field	Type	Notes
id	Integer	PK
mac	String	Unique
tag_id	String	Not Null
patient_names	String	Default=""

3. Station Table
Base station location mapping

Field	Type	Notes
id	Integer	PK
station_id	String	Unique
x	Float	X coordinate
y	Float	Y coordinate
room_name	String	Nullable
floor	String	Default="Ground"

4. Config Table
System configuration key-value store

Field	Type	Notes
key	String	PK
value	String	Not Null

5. TimelineEvent Table
Patient room transition tracking

Field	Type	Notes
id	Integer	PK
tag_id	String	Not Null
room	String	Not Null
entry_time	Integer	UNIX Timestamp
exit_time	Integer	UNIX Timestamp
duration	Integer	Seconds in room

Figure 6.4: SQLite database schema

The `User` model manages authentication credentials and profile information, providing the foundation for system access control. The `Tag` model, which stores essential identification data including MAC addresses, descriptive names, assignment status, and operational state.

Infrastructure components are represented by the `Station` model, capturing positioning

coordinates with floor-level granularity and operational status indicators. Spatial organization is achieved through `Room` and `Floor` models, where rooms define boundary coordinates for location-based services, and floors maintain map imagery with corresponding dimensional data for accurate spatial representation.
Historical data persistence is managed through `TimelineEvent` that captures significant system events like room transitions associated to a timestamp. This approach ensures both granular position tracking and event-driven monitoring capabilities.

6.6.2 Database Operations and Transaction Management

Database initialization is orchestrated by `create_db.py`, which establishes the complete table structure and provisions essential system data, including default administrative user accounts. The backend API layer, implemented in `server2.py`, executes **CRUD Operations** through dedicated database functions or direct SQLAlchemy session management, ensuring transactional integrity across all data operations.
The system manages four distinct data categories: user authentication and authorization data, system configuration parameters including device registrations and spatial definitions, real-time and historical positioning data, and MQTT broker configuration settings. This categorization enables efficient data organization and targeted optimization strategies for different access patterns.
While SQLite provides substantial advantages for the current implementation, including zero-configuration deployment and excellent reliability, it presents certain architectural constraints. The system's concurrency capabilities are limited under high write-load scenarios, and scalability remains constrained compared to client-server database solutions. Additionally, some advanced database features available in enterprise solutions are not present in SQLite's feature set.
These limitations represent acceptable trade-offs for the thesis project's scope, though production deployments at scale would benefit from migration to more robust database solutions such as PostgreSQL or MySQL. The current architecture's modular design through SQLAlchemy facilitates such future migrations with minimal code modifications, preserving the investment in application logic while enabling infrastructure evolution.

6.7 Frontend Architecture and User Experience

6.7.1 Frontend Architecture and Implementation

The frontend architecture implements a modern Single Page Application (SPA) using React as the foundational framework [52], providing an intuitive and responsive interface for system interaction. The application leverages contemporary web technologies including React Router for client-side navigation, TypeScript for enhanced code reliability and maintainability, and Socket.IO Client for real-time communication. The frontend code is organized according to the Flux architecture pattern with Redux for global application state management, complemented by React's Context API for component-level state distribution. This hybrid architectural approach facilitates management of complex state that includes real-time positioning data, system configurations, and user authentication information.

The component architecture follows a modular design pattern with clear separation of concerns across different functional layers. Application initialization begins with `main.tsx` as the entry point, which orchestrates context providers and initializes `App.tsx` to define the routing structure and URL-to-component mappings. The core component hierarchy encompasses view-level components including `Login.tsx`, `Register.tsx`, `Dashboard.tsx`, `Tracking.tsx`, `Analytics.tsx`, and `Config.tsx`, each serving distinct functional requirements within the application workflow.

Layout consistency is maintained through `HospitalLayout.tsx`, which provides a unified structural foundation incorporating `HospitalHeader.tsx` and `HospitalSidebar.tsx` components. Interactive mapping capabilities are delivered through specialized components `IndoorPositioningMap.tsx` and `FloorMap.tsx`, which render real-time tag locations and provide spatial visualization of the tracking environment using optimized SVG rendering for floor plans combined with React components for interactive elements.

6.7.2 Real-time Communication and Performance Optimization

To ensure smooth real-time updates, the application implements efficient WebSocket connection management through the `WebSocketContext.tsx` component, which provides a consistent API for subscribing to positioning and system events including `new_position` events and `station_status_update` notifications. This centralized approach to real-time connection management simplifies the implementation of reactive features throughout the application while maintaining optimal performance even with numerous tags displayed simultaneously at high update frequencies.

The frontend bundle is optimized through advanced performance strategies including code splitting and lazy loading techniques, reducing initial loading times and improving user experience. Static resources are served through Content Delivery Network (CDN) infrastructure to ensure fast response times even in distributed deployments. Map visualization performance is further enhanced through an optimized point-in-polygon algorithm used to determine room occupancy, specifically designed to minimize computational load on the browser while maintaining accuracy.

Data flow follows a structured pattern beginning with user authentication through the login interface and authentication context managed by `AuthContext.tsx`. Following successful authentication, components initialize with comprehensive data fetching operations retrieving floors, stations, and tag configurations. The indoor positioning map component renders using this initial dataset while continuously integrating WebSocket updates for seamless real-time functionality.

6.7.3 User Interface Design and Accessibility

The user interface has been developed following design system principles with reusable components that maintain visual and behavioral consistency throughout the application. Tailwind CSS implements utility-first styling alongside the `use-mobile.tsx` hook for adaptive layouts, ensuring responsive design that adapts to different screen sizes and maintains usability on both desktop workstations and mobile devices used by medical personnel.

Analytical dashboards utilize Recharts data visualization library, specifically chosen for its React compatibility and optimized performance characteristics. Charts are designed to be fully interactive, allowing users to explore data through hover interactions, zoom functionality, and dynamic selections, providing comprehensive data analysis capabilities within the hospital environment.

To ensure accessibility and inclusive design, the interface implements WCAG 2.1 guidelines including comprehensive support for keyboard navigation, adequate color contrast ratios for optimal readability, and full compatibility with assistive technologies. This inclusive approach ensures that the system remains usable by operators with diverse abilities and accessibility requirements.

Visual feedback mechanisms including Framer Motion animations and real-time status indicators provide immediate confirmation of user actions, while the interface emphasizes intuitiveness through clear navigation patterns and consistent UI elements. Additional contexts manage theme preferences and sidebar state, ensuring consistent user interface behavior across all application views.

This comprehensive frontend implementation represents a careful balance between advanced functionality, optimal performance, and exceptional usability, creating an interface that effectively meets the complex operational requirements of an indoor positioning system within a demanding hospital environment.

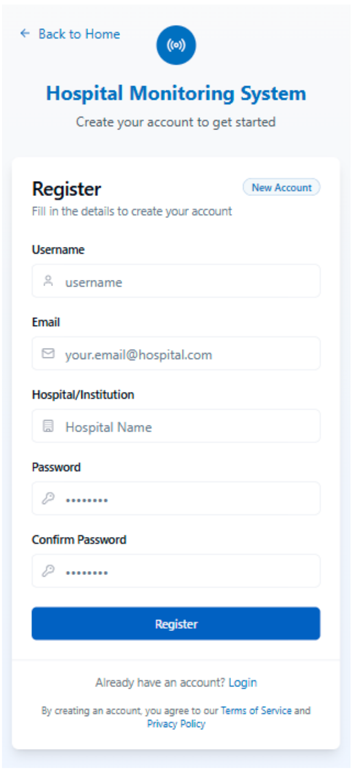
6.7.4 Frontend Feature Showcase

Although the system is not implemented in a real hospital, the developed system is named **Hospital Monitoring System** because it simulates the core functionalities and workflows of actual hospital monitoring environments, providing a realistic framework for tracking patient that would be essential in real-world healthcare settings.

This section presents the main frontend features that enable healthcare operators to monitor patients, visualize analytical data, and manage the system efficiently.

Authentication and Account Management

The system implements a robust authentication and account management mechanism to ensure that only authorized personnel can access patient monitoring functions and sensitive information.



← Back to Home

Hospital Monitoring System

Create your account to get started

Register

New Account

Fill in the details to create your account

Username

username

Email

your.email@hospital.com

Hospital/Institution

Hospital Name

Password

••••••••

Confirm Password

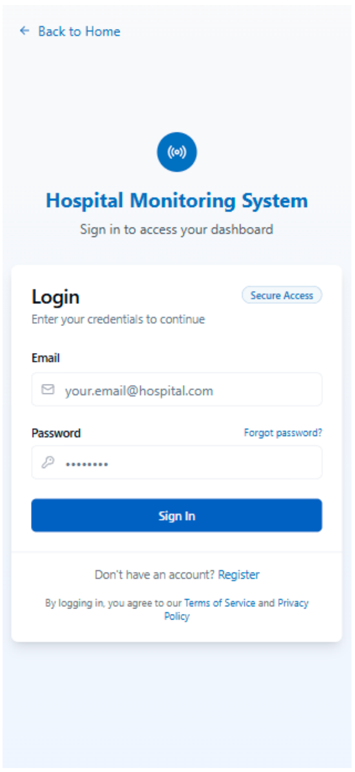
••••••••

Register

Already have an account? Login

By creating an account, you agree to our Terms of Service and Privacy Policy

(a) Registration interface



← Back to Home

Hospital Monitoring System

Sign in to access your dashboard

Login

Secure Access

Enter your credentials to continue

Email

your.email@hospital.com

Password

Forgot password?

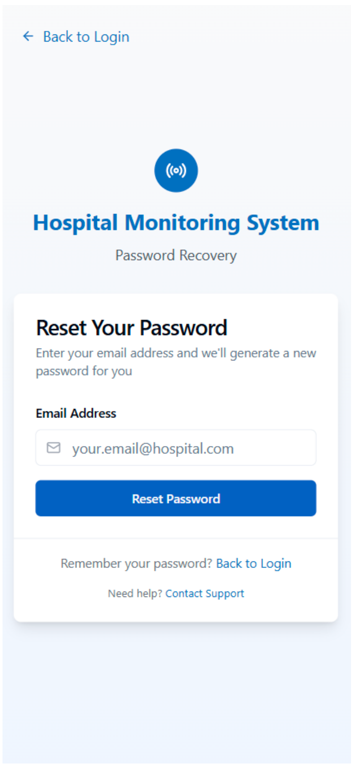
••••••••

Sign In

Don't have an account? Register

By logging in, you agree to our Terms of Service and Privacy Policy

(b) Login interface



← Back to Login

Hospital Monitoring System

Password Recovery

Reset Your Password

Enter your email address and we'll generate a new password for you

Email Address

your.email@hospital.com

Reset Password

Remember your password? Back to Login

Need help? Contact Support

(c) Password reset interface

Figure 6.5: Authentication interfaces showing registration, login, and password recovery forms with security features and user-friendly design.

As shown in Figure 6.5a, the registration interface has been designed with particular attention to usability and security:

- **Input Validation**: Each field implements real-time validation to ensure that entered data meets format and security requirements.
- **Password Security**: The system requires password confirmation and implements strength checks to ensure secure credentials.
- **Regulatory Compliance**: Links to terms of service and privacy policy ensure transparency and compliance with healthcare data protection regulations.
- **User Experience**: The minimalist design with clear instructions facilitates the registration process even for users with limited technical skills.

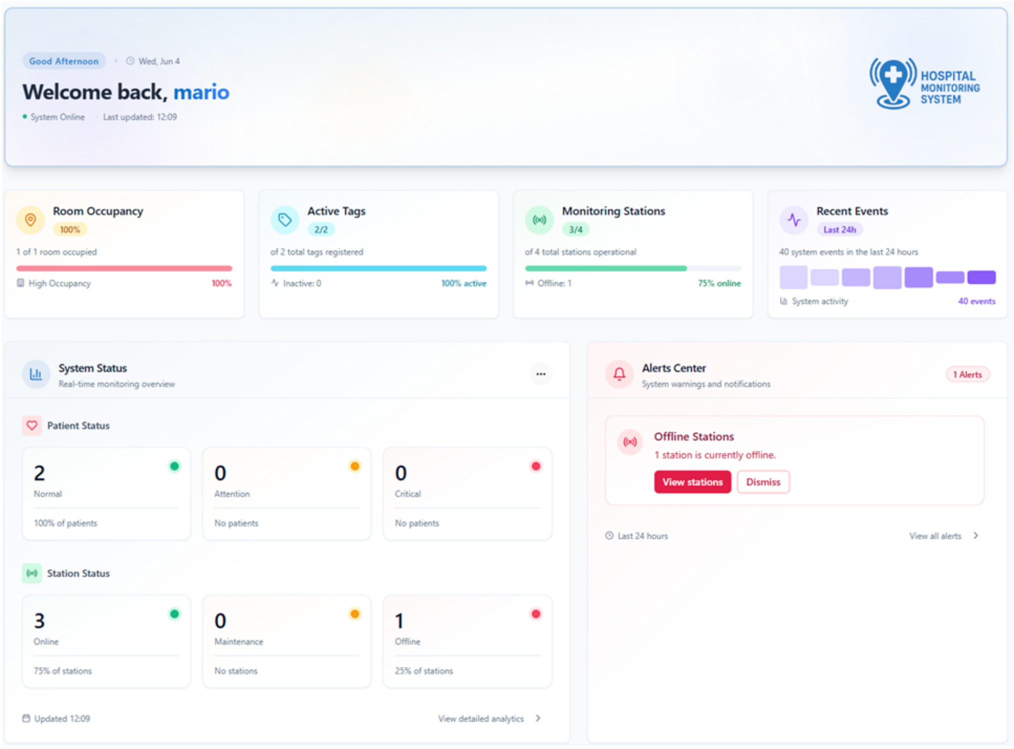
Figure 6.5b illustrates the login interface, designed to balance security and ease of use:

- **Simplified Access**: Essential form that requires only email and password to minimize access times in emergency situations.
- **Recovery Functionality**: Link for forgotten password recovery, with identity verification process through institutional email.
- **Secure Access**: "Secure Access" badge indicating the implementation of advanced security protocols, including connection encryption.
- **Intuitive Navigation**: Links to return to home or register a new account, facilitating navigation for new users.
- **Persistent Sessions**: The system implements secure authentication tokens that allow persistent sessions on verified devices, reducing the need for frequent logins.

The authentication system is integrated with the backend through REST APIs that implement security best practices [74], including JWT (JSON Web Token) with expiration, secure password hashing with bcrypt, and protection against brute force attacks through access attempt limitation. These measures are fundamental in a healthcare context where patient data protection is subject to stringent regulatory requirements.

Main Dashboard

The main dashboard provides a comprehensive overview of the system's real-time status, allowing operators to quickly monitor all critical aspects of the positioning infrastructure.



Good Afternoon · Wed, Jun 4

Welcome back, mario

System Online · Last updated: 12:09

HOSPITAL MONITORING SYSTEM

Room Occupancy
100%
1 of 1 room occupied
High Occupancy 100%

Active Tags
2/2
of 2 total tags registered
Inactive: 0 100% active

Monitoring Stations
3/4
of 4 total stations operational
Offline: 1 75% online

Recent Events
Last 24h
40 system events in the last 24 hours
System activity 40 events

System Status
Real-time monitoring overview

Patient Status

2 Normal — 100% of patients
0 Attention — No patients
0 Critical — No patients

Station Status

3 Online — 75% of stations
0 Maintenance — No stations
1 Offline — 25% of stations

Updated 12:09 · View detailed analytics

Alerts Center
System warnings and notifications
1 Alerts

Offline Stations
1 station is currently offline.
View stations · Dismiss

Last 24 hours · View all alerts

Figure 6.6: Main dashboard of the indoor positioning system showing real-time status of room occupancy, active tags, monitoring stations, and recent events.

The dashboard (Figure 6.6) is structured in modular widgets that present critical information in summary format:

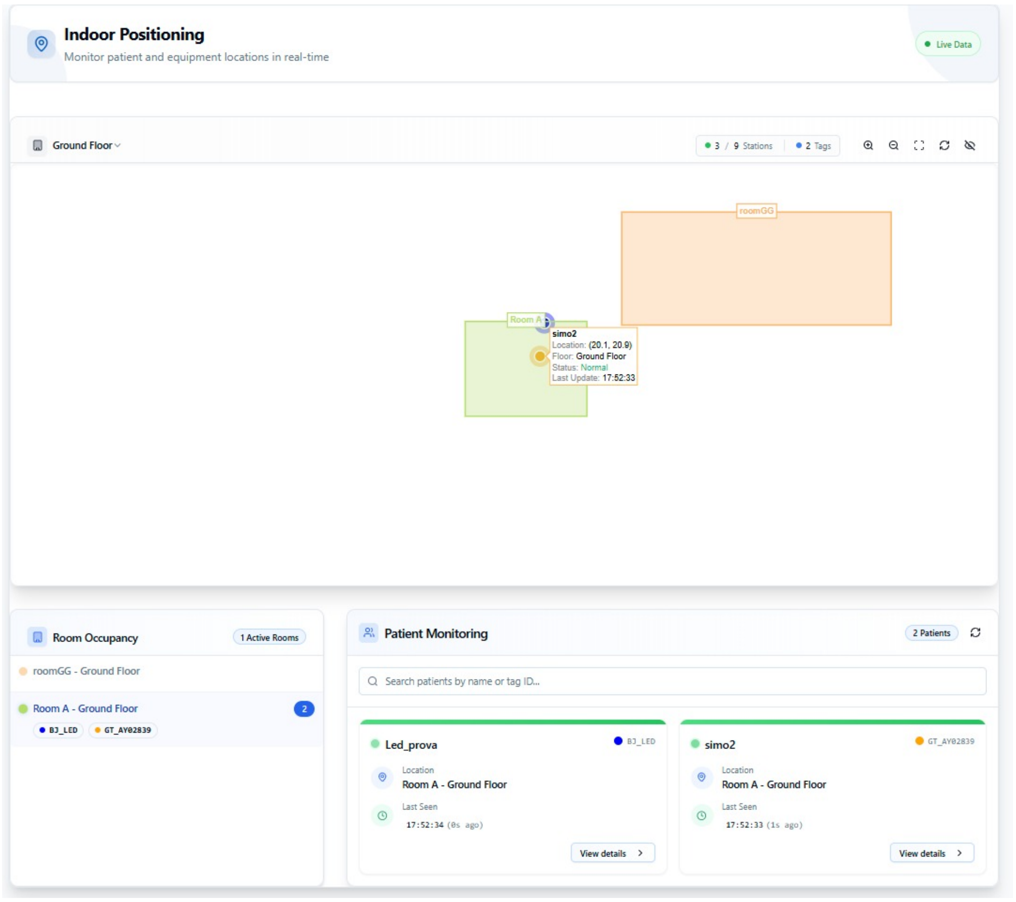
- **Room Occupancy**: Displays the percentage of currently occupied rooms, with indication of the exact number of monitored rooms.
- **Active Tags**: Shows the number of BLE tags currently registered and active in the system, with indication of the percentage of functioning tags.
- **Monitoring Stations**: Presents the status of ESP32 stations, highlighting how many are operational and how many are offline.
- **Recent Events**: Chronicle of the latest system events in the preceding 24 hours, represented graphically to facilitate identification of temporal patterns.
- **Patient Status**: Categorizes monitored patients based on their status (normal, attention, critical), providing immediate visibility on situations requiring intervention.

- **Station Status**: Summarizes the operational status of ESP32 stations, distinguishing between online, offline, and under maintenance.
- **Alert Center**: Notifies system problems requiring attention, such as offline stations or data anomalies.

The dashboard implements real-time updates through WebSocket [55], ensuring that displayed information is always synchronized with the current system status without requiring manual page refresh. This approach is particularly important in a hospital context where information timeliness can be critical.

Map Visualization and Real-time Positioning

Map visualization represents the heart of the positioning interface, allowing operators to visually locate patients within the hospital facility.



Indoor Positioning
Monitor patient and equipment locations in real-time
● Live Data

Ground Floor
● 3 / 9 Stations ● 2 Tags

roomGG
Room A
simo2
Location: (20.1, 20.9)
Floor: Ground Floor
Status: Normal
Last Update: 17:52:33

Room Occupancy 1 Active Rooms
roomGG - Ground Floor
Room A - Ground Floor 2
● BJ_LED ● GT_AY02839

Patient Monitoring 2 Patients
Search patients by name or tag ID...

Led_prova ● BJ_LED
Location
Room A - Ground Floor
Last Seen
17:52:34 (0s ago)
View details

simo2 ● GT_AY02839
Location
Room A - Ground Floor
Last Seen
17:52:33 (1s ago)
View details

Figure 6.7: Detailed Indoor positioning map visualization.

The map visualization interface (Figure 6.7) includes several key components:

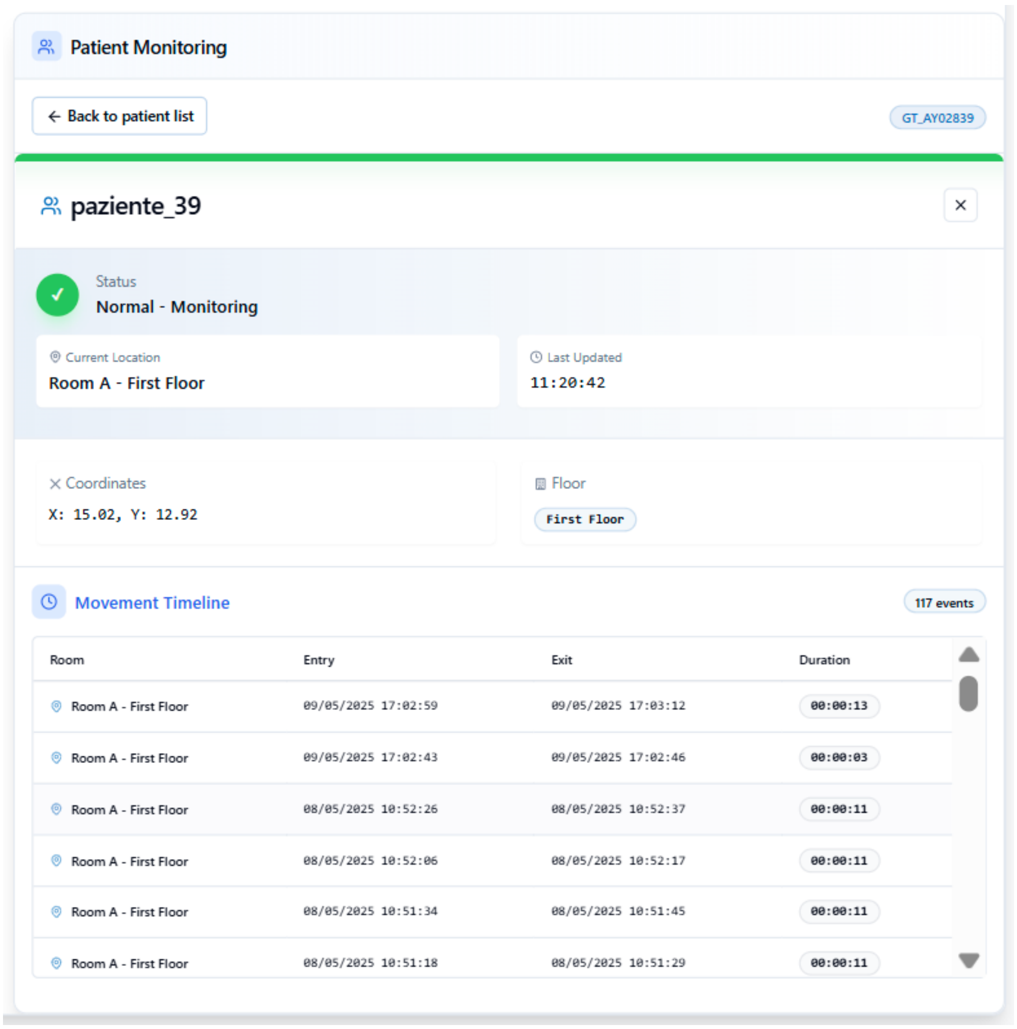
- **Interactive Map**: Visual representation of the hospital floor plan with rooms defined as colored polygons. The map supports zoom, pan, and selection operations to facilitate navigation in complex structures.
- **Tag Positioning**: BLE tags are visualized as colored points positioned based on coordinates calculated by the trilateration algorithm [68]. Each tag is associated with a specific patient or equipment.
- **Floor Selection**: A selector allows quick switching between different building floors, maintaining visual consistency during navigation.
- **Status Indicators**: Visual counters show the number of active stations and monitored tags on the current floor.

- **Occupancy Panel**: A side panel lists currently occupied rooms with the count of tags present in each.
- **Patient Monitoring**: A dedicated section shows details of currently monitored patients, including their current position and last detection.

The map implementation uses a combination of SVG for floor plan representation and React for managing interactive elements. Tag coordinates are updated in real-time through WebSocket connection [61], ensuring smooth visualization of movements. Room definition as polygons enables implementation of point-in-polygon algorithms to determine room occupancy and generate entry/exit events.

Timeline and Historical Monitoring

The system offers advanced features for historical monitoring of patient movements, enabling retrospective analysis and evaluation of residence time in different areas of the facility.


Patient Monitoring

Back to patient list

GT_AY02839

paziente_39

Status
Normal - Monitoring

Current Location
Room A - First Floor

Last Updated
11:20:42

Coordinates
X: 15.02, Y: 12.92

Floor
First Floor

Movement Timeline

117 events

Room	Entry	Exit	Duration
Room A - First Floor	09/05/2025 17:02:59	09/05/2025 17:03:12	00:00:13
Room A - First Floor	09/05/2025 17:02:43	09/05/2025 17:02:46	00:00:03
Room A - First Floor	08/05/2025 10:52:26	08/05/2025 10:52:37	00:00:11
Room A - First Floor	08/05/2025 10:52:06	08/05/2025 10:52:17	00:00:11
Room A - First Floor	08/05/2025 10:51:34	08/05/2025 10:51:45	00:00:11
Room A - First Floor	08/05/2025 10:51:18	08/05/2025 10:51:29	00:00:11

Figure 6.8: Timeline interface for historical monitoring of patient movements.

The timeline interface (Figure 6.8) includes:

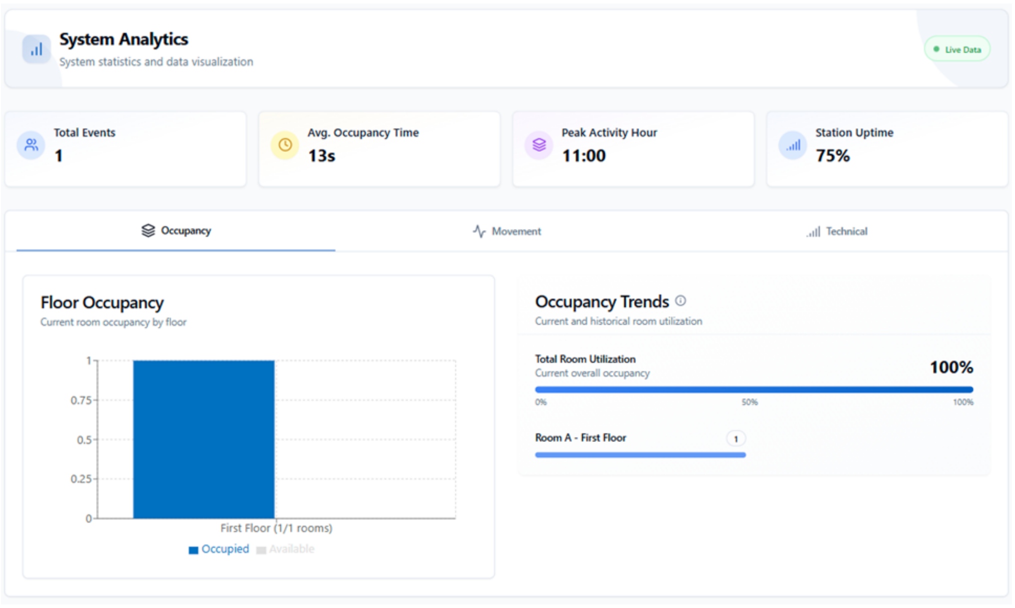
- **Patient Information**: Header with patient identifier, current status, and associated tag.
- **Current Position**: Display of the room where the patient is currently located, with precise coordinates and time of last update.
- **Movement History**: Detailed table recording all patient movements, with indication of room, entry time, exit time, and duration of stay.

- **Temporal Navigation**: Controls to scroll through history and view older events, with possibility to filter by date.

This functionality is particularly useful for contact tracing in case of nosocomial infections [49], for space utilization analysis, and for verification of care protocols that require regular patient visits.

Data Analysis and Visualization

The system includes an advanced analysis module that processes positioning data to provide meaningful insights on space utilization and movement patterns within the facility.



System Analytics
System statistics and data visualization
Live Data

Total Events: 1
Avg. Occupancy Time: 13s
Peak Activity Hour: 11:00
Station Uptime: 75%

Occupancy | Movement | Technical

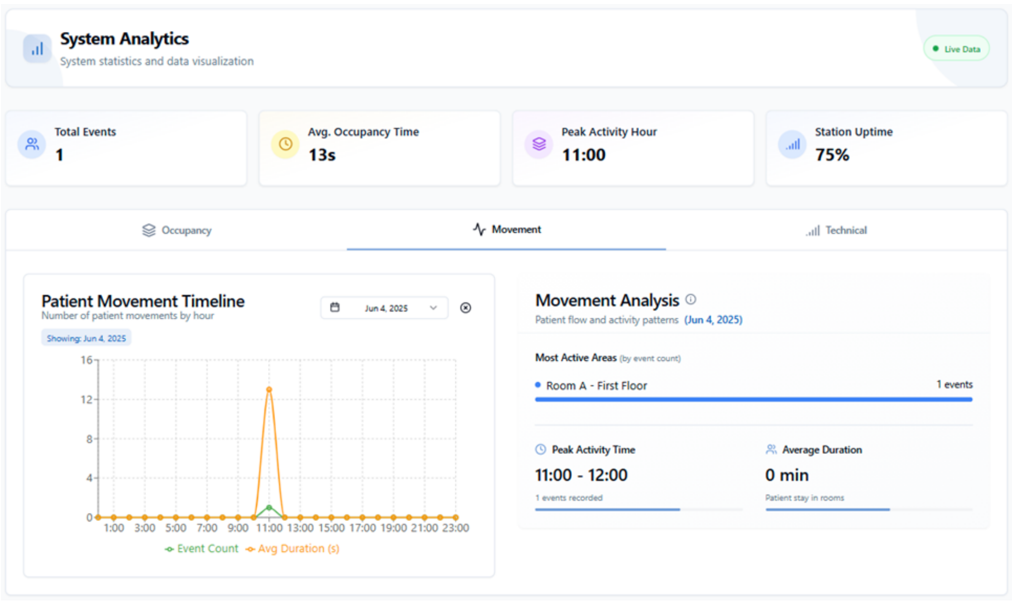
Floor Occupancy
Current room occupancy by floor

Occupancy Trends
Current and historical room utilization
Total Room Utilization: 100%
Current overall occupancy
0% 50% 100%
Room A - First Floor: 1

Figure 6.9: Occupancy analysis dashboard showing aggregated statistics and graphical visualizations.

The occupancy analysis dashboard (Figure 6.9) includes:

- **Key Metrics**: Summary indicators showing total events, average occupancy time, peak activity hour, and station uptime.
- **Occupancy by Floor**: Bar chart visualizing current occupancy for each floor, with distinction between occupied and available spaces.
- **Occupancy Trends**: Visualization of historical occupancy trends, with utilization percentages for each room.
- **Time Selection**: Controls to filter data based on specific time intervals (day, week, month).


System Analytics
System statistics and data visualization
Live Data

Total Events 1
Avg. Occupancy Time 13s
Peak Activity Hour 11:00
Station Uptime 75%

Occupancy | Movement | Technical

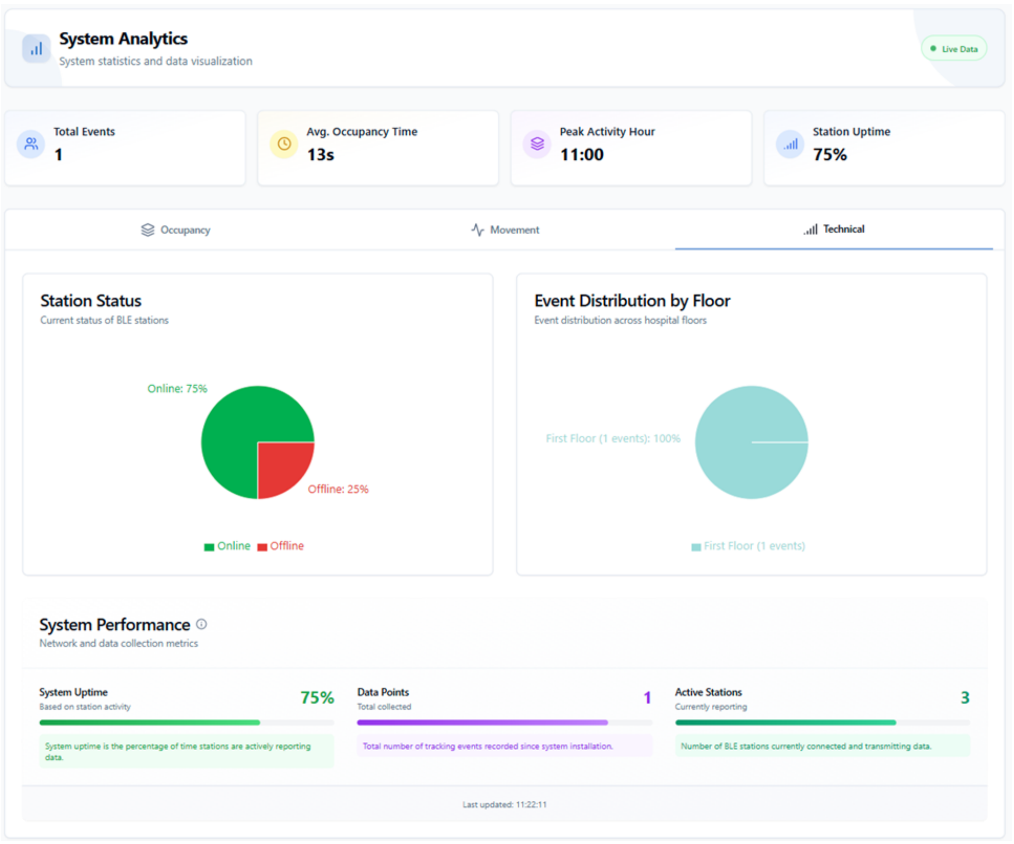
Patient Movement Timeline
Number of patient movements by hour
Showing: Jun 4, 2025
Jun 4, 2025

Movement Analysis
Patient flow and activity patterns (Jun 4, 2025)
Most Active Areas (by event count)
Room A - First Floor 1 events
Peak Activity Time 11:00 - 12:00
1 events recorded
Average Duration 0 min
Patient stay in rooms

Figure 6.10: Movement analysis dashboard visualizing temporal patterns of patient movement. Includes a timeline of movement events by hour of day, analysis of most active areas, peak activity hours, and average duration of stay in rooms.

The movement analysis dashboard (Figure 6.10), provides:

- **Movement Timeline**: Temporal graph showing the number of patient movements per hour of day, highlighting recurring patterns.
- **Most Active Areas**: Ranking of rooms with highest activity based on event count.
- **Peak Hours**: Identification of time slots with highest movement activity.
- **Average Duration**: Analysis of average time patients spend in different rooms.



System uptime is the percentage of time stations are actively reporting data.

Total number of tracking events recorded since system installation.

Number of BLE stations currently connected and transmitting data.

Figure 6.11: Technical analysis dashboard showing system operational status and performance metrics.

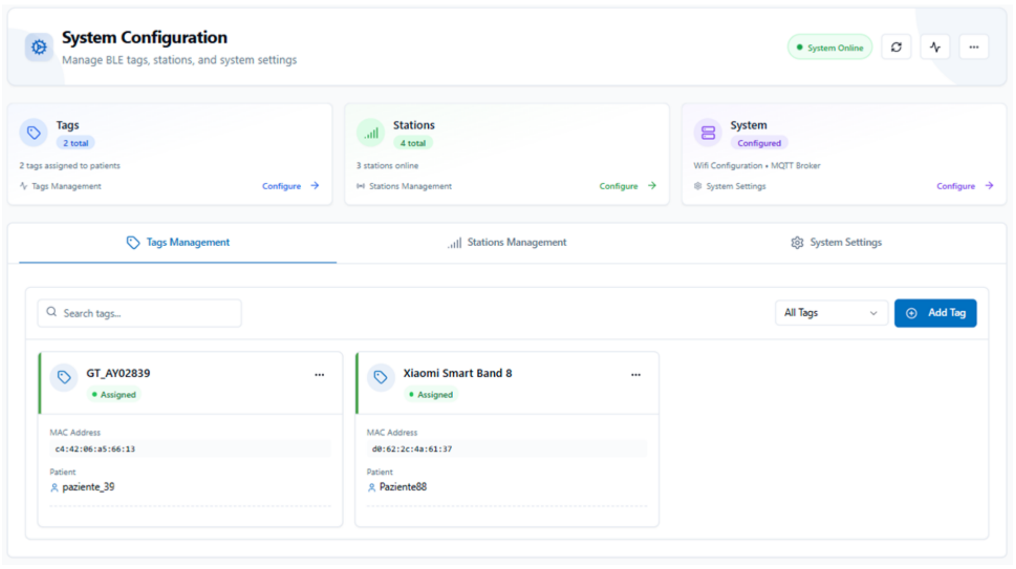
The technical analysis dashboard (Figure 6.11), provides infrastructure performance as:

- **Station Status**: Pie chart showing the percentage of online and offline stations.
- **Event Distribution**: Visualization of positioning event distribution among different building floors.
- **System Performance**: System uptime metrics, based on station activity.
- **Data Points**: Total count of data points collected since system installation.
- **Active Stations**: Number of ESP32 stations currently connected and transmitting data.

These analytical dashboards are implemented using React data visualization libraries [52], such as Recharts and D3.js, which enable interactive and responsive graphical representations. Data is retrieved through REST API calls to the backend, with the possibility of exporting in standard formats like CSV for further analysis.

Configuration and Administration

The system includes a comprehensive administration interface that allows configuration of all aspects of the positioning infrastructure.



System Configuration
Manage BLE tags, stations, and system settings
System Online

Tags
2 total
2 tags assigned to patients
Tags Management — Configure →

Stations
4 total
3 stations online
Stations Management — Configure →

System
Configured
Wifi Configuration • MQTT Broker
System Settings — Configure →

Tags Management | Stations Management | System Settings

Search tags...
All Tags
Add Tag

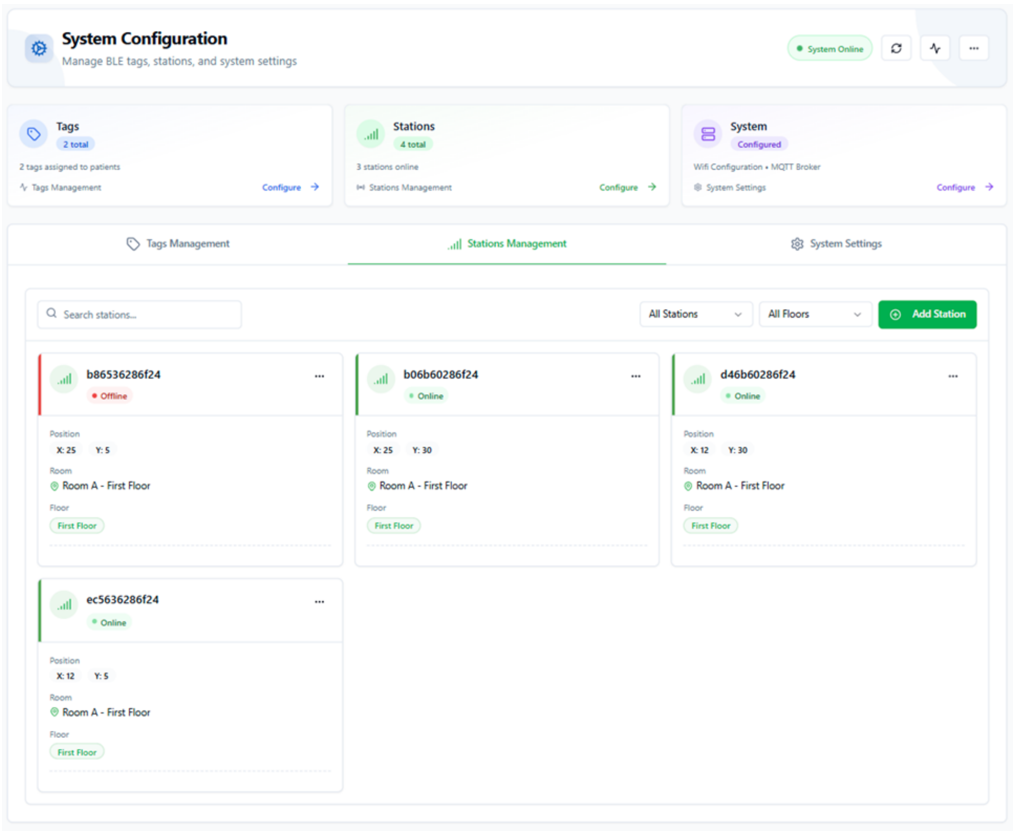
GT_AY02839
Assigned
MAC Address
`c4:42:06:a5:66:13`
Patient
paziente_39

Xiaomi Smart Band 8
Assigned
MAC Address
`d0:62:2c:4a:61:37`
Patient
Paziente88

Figure 6.12: BLE tag management interface that enables administration of tracking devices.

The BLE tag management interface (Figure 6.12) includes:

- **Tag List**: Display of all tags registered in the system with their assignment status.
- **Tag Details**: For each tag, MAC address, unique identifier, and associated patient are shown.
- **Management Functions**: Buttons to add new tags, manage assignments and delete old tags.
- **Search Filters**: Search field to quickly find specific tags in systems with numerous devices.



b86536286f24
Offline
Position
X: 25 Y: 5
Room
Room A - First Floor
Floor
First Floor

b06b60286f24
Online
Position
X: 25 Y: 30
Room
Room A - First Floor
Floor
First Floor

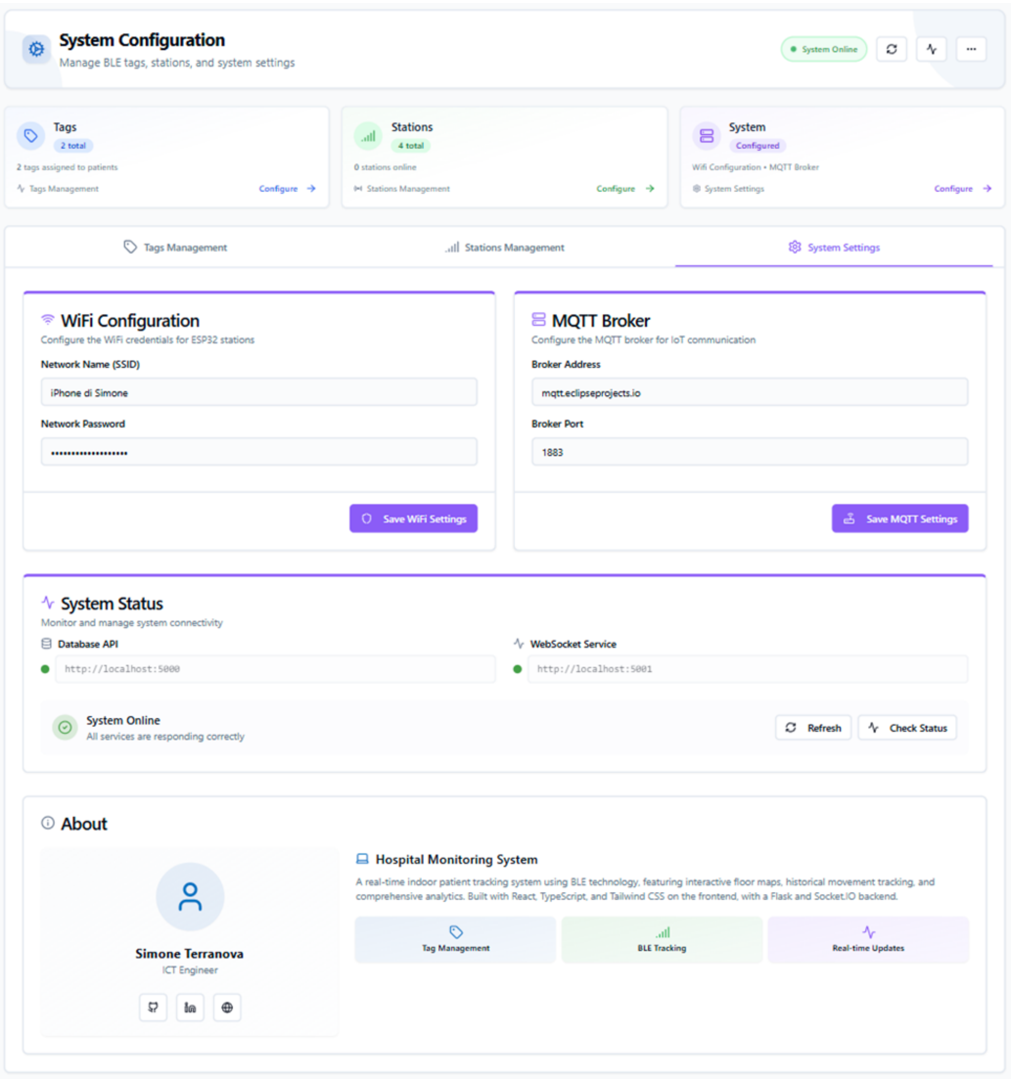
d46b60286f24
Online
Position
X: 12 Y: 30
Room
Room A - First Floor
Floor
First Floor

ec5636286f24
Online
Position
X: 12 Y: 5
Room
Room A - First Floor
Floor
First Floor

Figure 6.13: ESP32 station management interface showing operational status and configuration of each scanning station.

The ESP32 station management interface (Figure 6.13) presents:

- **Station List**: Display of all scanning stations with indication of their operational status (online/offline).
- **Station Details**: For each station, unique identifier, installation coordinates (X, Y), and the room where it is positioned are shown.
- **Filters**: Options to filter stations by status or installation floor.
- **Management Functions**: Buttons to add new ESP32 stations, manage details and delete old stations.



System Configuration
Manage BLE tags, stations, and system settings

Tags 2 total
2 tags assigned to patients
Tags Management — Configure →

Stations 4 total
0 stations online
Stations Management — Configure →

System Configured
Wifi Configuration • MQTT Broker
System Settings — Configure →

WiFi Configuration
Configure the WiFi credentials for ESP32 stations
Network Name (SSID): iPhone di Simone
Network Password: ••••••••••••••••••

MQTT Broker
Configure the MQTT broker for IoT communication
Broker Address: mqtt.eclipseprojects.io
Broker Port: 1883

System Status
Monitor and manage system connectivity
Database API: `http://localhost:5000`
WebSocket Service: `http://localhost:5001`
System Online
All services are responding correctly

About
Simone Terranova
ICT Engineer

Hospital Monitoring System
A real-time indoor patient tracking system using BLE technology, featuring interactive floor maps, historical movement tracking, and comprehensive analytics. Built with React, TypeScript, and Tailwind CSS on the frontend, with a Flask and Socket.IO backend.

Figure 6.14: System configuration interface that allows management of connectivity and communication settings.

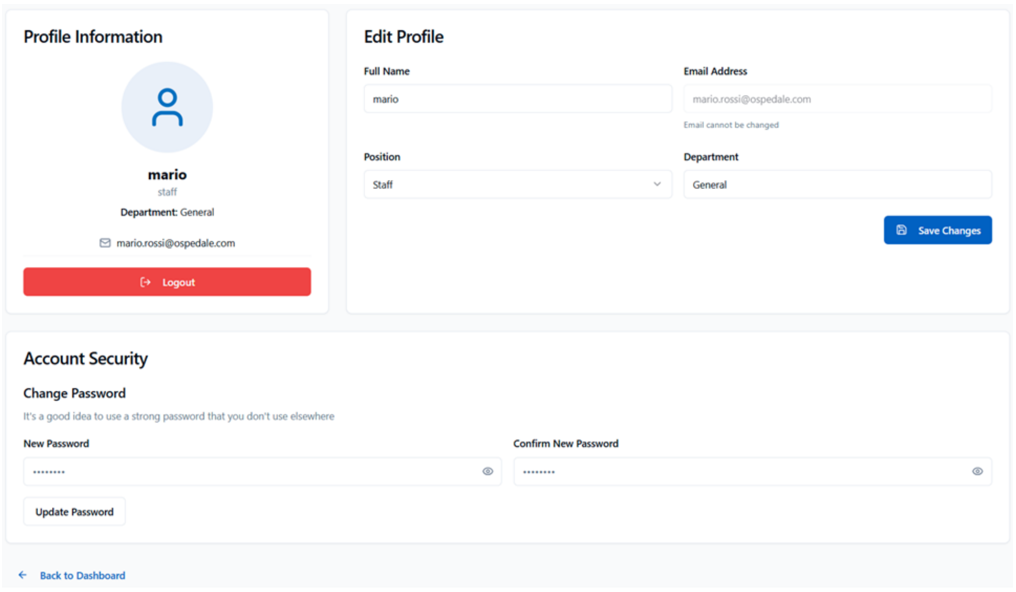
The system configuration interface (Figure 6.14) is useful to set:

- **Wi-Fi Configuration**: Settings for Wi-Fi connectivity of ESP32 stations, including SSID and password.
- **MQTT Configuration**: Parameters for the MQTT broker [53] used for IoT communication, including address and port.
- **System Status**: Real-time monitoring of backend services, including API database and WebSocket service.
- **System Information**: Details about the hospital monitoring system, including used technologies and main functionalities.

These configuration interfaces are designed to be used by technical personnel responsible for system installation and maintenance. Access to these functions is protected by role-based authentication and authorization, ensuring that only users with appropriate privileges can modify critical settings.

User Features and Accessibility

The system includes several user-oriented features that improve the overall experience and facilitate interaction with the platform.



Profile Information

mario
staff
Department: General
mario.rossi@ospedale.com

Edit Profile

Full Name: mario
Email Address: mario.rossi@ospedale.com
Email cannot be changed
Position: Staff
Department: General

Account Security

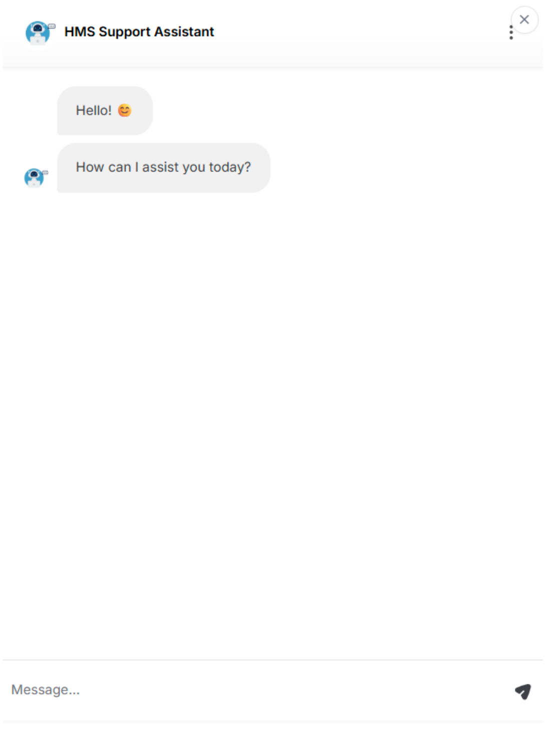
Change Password
It's a good idea to use a strong password that you don't use elsewhere
New Password: ••••••••
Confirm New Password: ••••••••

Figure 6.15: User profile management interface that allows modification of personal information and security settings.

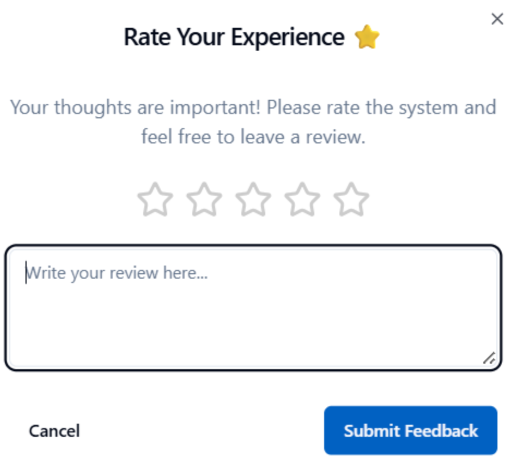
Through the user profile management interface (Figure 6.15), it is possible to access:

- **Profile Information**: Display and modification of personal data such as name, email, and department.
- **Account Security**: Functionality for password change with password strength indications.
- **Preferences**: Options to customize user experience (not shown in the image).

Figure 6.16 illustrates the integrated support assistant and feedback interface:



(a) Integrated support assistant


Rate Your Experience ⭐
Your thoughts are important! Please rate the system and feel free to leave a review.
Write your review here...
Cancel
Submit Feedback

(b) Feedback interface

Figure 6.16: Support Assistant and Feedback interfaces.

The support assistant (Figure 6.16a) features include:

- **Chat Interface**: Dialog window for natural language interactions with the assistant.
- **Contextual Responses**: The assistant provides specific information and guides based on the user's current context.
- **Accessibility**: The assistant is designed to be accessible to users with different levels of technical competence.

The feedback interface (Figure 6.16b) components include:

- **Rating System**: Star mechanism to quantitatively assess user experience.
- **Comments Field**: Text area to provide detailed feedback and suggestions.
- **Submit Feedback**: Button to submit the evaluation to the development team.

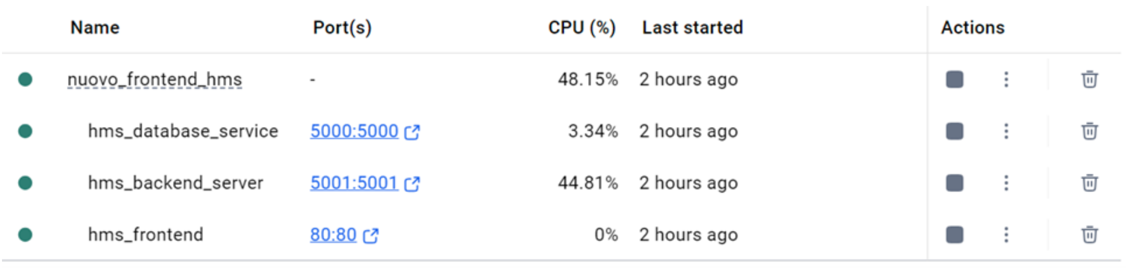
These user-oriented features are designed to improve overall system usability, provide contextual assistance, and collect feedback for continuous improvement. The user interface has been developed following responsive design principles, ensuring that the system is usable on different devices, from desktops to tablets used by medical personnel during ward rounds.

6.8 Security Framework and Implementation

Security represents a fundamental system requirement given the sensitive nature of patient location data and the critical operational environment of healthcare facilities. The implemented security framework addresses authentication, communication protection, device security, and data privacy through comprehensive measures across all system layers.
Authentication mechanisms utilize bcrypt password hashing for credential protection combined with token-based session management. The frontend `AuthContext.tsx` manages authentication state while backend endpoints in `server2.py` provide secure login, registration, and profile management services. Role-based access control frameworks support granular permission management appropriate for healthcare environments with diverse user roles and responsibilities.
Communication security requires encrypted channels for all data transmission paths. Frontend-to-backend REST communications utilize HTTPS with TLS 1.3 encryption, while real-time WebSocket connections employ WSS protocols. ESP32-to-broker and backend-to-broker MQTT communications require MQTTS implementation with robust client authentication mechanisms. Production deployments necessitate private MQTT broker infrastructure with comprehensive access controls and monitoring capabilities.
Device security encompasses physical protection through tamper-evident enclosures and secure mounting systems, firmware integrity through secure over-the-air update mechanisms, credential management through encrypted non-volatile storage, and network isolation through dedicated VLANs with appropriate firewall configurations.
Backend security implementations include comprehensive input validation to prevent SQL injection, cross-site scripting, and command injection attacks. SQLAlchemy ORM provides inherent protection against SQL injection while maintaining efficient database operations. Error handling mechanisms prevent information leakage through sanitized responses that avoid exposing system internals.
Data privacy measures implement data minimization principles by limiting collection to essential information, anonymization techniques where feasible, strict access controls on patient data, and clear retention policies governing data storage duration. The system maintains adherence to relevant healthcare regulations including GDPR and HIPAA requirements.

6.9 Deployment Strategy and Containerization

The deployment architecture utilizes Docker containerization and orchestration to ensure consistent runtime environments, simplified scaling, and robust operational management across development and production environments [57].



	Name	Port(s)	CPU (%)	Last started	Actions
●	nuovo_frontend_hms	-	48.15%	2 hours ago	
●	hms_database_service	5000:5000	3.34%	2 hours ago	
●	hms_backend_server	5001:5001	44.81%	2 hours ago	
●	hms_frontend	80:80	0%	2 hours ago	

Figure 6.17: Docker container deployment architecture showing service orchestration, networking, and persistent storage configuration.

Docker Compose orchestration through `docker-compose.yml` defines the complete multi-container application including service definitions, network configurations, volume mappings, and dependency relationships. The database service manages SQLite persistence through Docker volumes that ensure data durability across container lifecycle events. The backend service utilizes `Dockerfile.backend_server_new_version` to create optimized Python runtime environments with Flask and Gunicorn configuration for production readiness.
Frontend deployment employs `Dockerfile.frontend_app_new_version` with multi-stage builds that compile React applications and serve static assets through Nginx web servers. This approach optimizes container size while providing robust serving capabilities for production environments.
Network architecture utilizes custom bridge networks that enable secure inter-container communication using service names while maintaining isolation from external networks. Named Docker volumes provide persistent storage for database files and configuration data that survives container updates and deployments.
The containerized approach supports horizontal scaling through container replication, load balancing through proxy configurations, and rolling updates through orchestrated deployment strategies. Development environments benefit from rapid deployment and consistent configuration, while production environments achieve robust isolation and resource management.
Container monitoring and logging capabilities provide operational visibility through standard Docker tools and third-party monitoring solutions. Health check implementations ensure service availability while restart policies maintain system resilience in the face of component failures.

Chapter 7

System Installation and Deployment

This chapter presents the comprehensive methodology for deploying the BLE-based indoor positioning system, from initial site analysis through system validation and maintenance considerations. The deployment process follows a systematic approach encompassing pre-installation planning, physical implementation, system configuration, performance validation, and scalability considerations.

7.1 Pre-Installation Planning and Site Assessment

7.1.1 Environmental Analysis Methodology

The environmental analysis methodology provides the foundation for optimal system deployment through comprehensive site assessment. This process involves four critical phases that determine technical requirements and installation parameters.

Physical inspection encompasses detailed assessment of room dimensions, geometry, construction materials, and furnishing elements. As highlighted by [75], reinforced concrete walls, metal structures, and mirrors significantly affect BLE signal propagation, creating shadow zones or multiple reflections that must be accounted for in the deployment strategy.

Interference mapping identifies electronic devices, household appliances, and electromagnetic interference sources operating in the 2.4 GHz band. According to [76], microwave ovens, Wi-Fi routers, and cordless phones can compromise BLE signal quality, requiring careful consideration during site planning.

Movement path analysis examines typical movement patterns within rooms to identify high-traffic areas requiring enhanced coverage. Following [77] recommendations for tracking optimization, this analysis ensures adequate signal coverage in areas of primary user activity.

Environmental conditions assessment evaluates temperature, humidity, and ventilation factors. As emphasized by [78], these conditions significantly impact battery life and BLE signal stability, influencing both hardware selection and maintenance scheduling.

7.1.2 Technical Specifications and Requirements

Based on the environmental analysis and findings from Chapter 6, the following technical specifications establish the deployment parameters:
System coverage requirements specify a minimum of 3-5 ESP32 receiver stations per room to enable accurate trilateration. Optimal mounting height ranges from 2-2.5 meters, with maximum spacing of 7-10 meters between adjacent stations following [79] guidelines.
Infrastructure requirements encompass continuous power supply with backup capability and Wi-Fi coverage ensuring RSSI values greater than -70 dBm at all installation points. The system targets localization accuracy under 2 meters mean error, appropriate for domestic positioning contexts.

7.1.3 Site Mapping and Device Positioning Strategy

Site mapping creates detailed area maps with precise dimensional measurements and scaled floor plans. A Cartesian coordinate system with fixed reference points enables accurate specification of receiver station positions, maintaining consistency with the database configuration established in Chapter 6.
Device positioning follows established optimization principles to ensure comprehensive coverage and measurement accuracy. Complete coverage requires each area to be monitored by at least three receiver stations for accurate trilateration. Interference minimization positions stations to avoid conflicts with existing electronic equipment.
Geometric optimization arranges stations to maximize triangulation accuracy while avoiding collinear configurations, as recommended by [80]. Perimeter positioning places stations along room perimeters at 2-meter height following [81] best practices.
The positioning plan specifies precise coordinates (x, y) in the reference system, mounting height and orientation, power supply method, and unique identifier (`station_id`) for each station.

7.2 Physical Installation and Hardware Configuration

7.2.1 Component Preparation and Pre-Configuration

Pre-installation preparation ensures systematic component verification and configuration before deployment. Component inventory includes ESP32 receiver stations, Global Tag Disk Beacon BLE tags, power adapters, extension cables, and specialized installation tools.
ESP32 pre-configuration involves loading the BLE scanner/receiver firmware developed in Chapter 4, configuring Wi-Fi credentials and MQTT parameters, and assigning unique identifiers (`station_id`) to each station. Functional verification tests network connectivity, tag detection capabilities, and backend communication before physical deployment.

7.2.2 Physical Deployment and Installation

Physical installation follows the predetermined layout with precise position marking using surveying tools. ESP32 receiver stations are mounted at planned coordinates with proper

orientation to ensure optimal coverage patterns. The installation demonstrates strategic positioning:
Station 1-4 occupy corner positions at approximately 2-meter height, while Station 5 provides center ceiling mount coverage at equivalent height. This arrangement ensures comprehensive area coverage while minimizing potential interference zones.
Power cable management maintains stable connections while minimizing visual impact through concealed routing and proper cable securing techniques.

7.2.3 BLE Tag Configuration and Network Integration

Global Tag BLE Disk Beacon configuration encompasses comprehensive hardware preparation and network integration. Hardware preparation includes battery status verification and advertising parameter setup according to specifications outlined in Chapter 5.
Each tag receives unique identifier assignment for system integration:

- Tag 1: MAC `c4:42:06:a5:66:13`, ID: `GT_AY02839`
- Tag 2: MAC `f8:f2:c5:76:db:92`, ID: `GT_AY02840`
- Tag 3: MAC `fd:50:aa:a4:e7:9f`, ID: `GT_AY02841`

Wearable integration incorporates tags into ergonomic bracelets following user comfort and antenna optimization considerations detailed in Chapter 5.
Network configuration establishes static IP addressing for receiver stations, MQTT traffic prioritization with appropriate broker parameters, WPA2/WPA3 security protocols, and Quality of Service (QoS) settings ensuring reliable data transmission.

7.3 System Configuration and Calibration

7.3.1 Software Initialization and Development Environment

Post-installation configuration utilizes the frameworks established in Chapter 6. Development environment setup includes Python library installation (Flask, SQLAlchemy, paho-mqtt, numpy, scipy), SQLite database initialization with required table structures, and Flask backend configuration incorporating MQTT broker parameters and positioning algorithm settings.

7.3.2 System Calibration and Parameter Optimization

System calibration adapts the Log-Distance Path Loss model through systematic RSSI measurement at known distances to determine $RSSI_0$ and path loss exponent N parameters. This process establishes the fundamental relationship between signal strength and distance for the specific deployment environment.
Algorithm parameter optimization fine-tunes Exponential Moving Average and Kalman filter parameters for optimal noise reduction and position stability. Weighted multilateration algorithm parameters undergo calibration to adapt to the specific environmental geometry and propagation characteristics.

7.4 System Validation and Performance Assessment

7.4.1 Coverage Validation and Signal Analysis

Initial validation employs systematic signal coverage testing using test Disk Beacons to map system performance across the deployment area. This process identifies optimal coverage zones where three or more stations provide reliable detection, marginal areas with limited coverage, and potential shadow zones requiring attention.

Coverage analysis establishes baseline performance metrics and identifies areas requiring adjustment before full system commissioning.

7.4.2 Accuracy Testing and System Performance

Accuracy testing compares real versus estimated positions to establish comprehensive system performance baselines. This validation process quantifies positioning accuracy under various conditions and identifies factors affecting measurement precision.

Performance assessment enables necessary calibration adjustments and validates that the system meets the specified accuracy requirements for the intended application context.

7.4.3 Frontend Integration and User Interface

Frontend configuration integrates real-time position displays, analytics capabilities, system configuration interfaces, and user profile management functionality. This integration provides the user-facing components necessary for system operation and monitoring.

7.5 Maintenance and Scalability Considerations

7.5.1 Monitoring and Maintenance Strategy

Continuous monitoring through the `app_w_db.py` implementation tracks receiver station status, Disk Beacon battery levels, and comprehensive system performance metrics following [82] best practices for system reliability and performance optimization.

Preventive maintenance includes quarterly hardware inspections, semi-annual coverage verification, and monthly database optimization procedures following [83] recommendations for sustained system performance.

Battery management addresses critical operational concerns through continuous monitoring and standardized replacement procedures that anticipate depletion and minimize service disruptions. As emphasized by [84], effective battery management significantly impacts system reliability and operational costs.

Software updates utilize Over-The-Air distribution mechanisms for ESP32 stations and version-controlled backend deployment with comprehensive pre-production testing protocols to ensure system stability and feature enhancement.

7.5.2 Hospital Environment Scaling Requirements

Building on the healthcare applications discussed in Section 2.4.1, hospital environments require substantially different deployment strategies compared to domestic installations. A typical 10,000m^2 hospital facility necessitates a dense network of 150-200 BLE scanner stations powered via PoE with battery backup, supporting unlimited tag capacity [85,86].
Infrastructure Requirements: Scanner stations must provide comprehensive coverage through RF analysis-based positioning, supporting hundreds to thousands of patients with 1-2 second update intervals. A hybrid sensor fusion approach can enhance accuracy by combining BLE (cost-effective, low power) with UWB technology (10-30 cm precision), Wi-Fi RTT leveraging existing infrastructure, and complementary sensors including IMU for motion tracking and barometric sensors for floor-level detection. Network infrastructure requires enterprise-grade Wi-Fi (IEEE 802.11ac/ax), wired Ethernet backbone with PoE switches, and optional mesh networks for extended coverage in challenging areas.
Technical Challenges: Medical equipment interference management requires channel diversification strategies and electromagnetic compatibility compliance (IEC 60601). Backend architecture must upgrade from SQLite to enterprise-grade distributed systems utilizing PostgreSQL for relational data and InfluxDB for time-series positioning data to achieve required scalability and performance.
Regulatory Compliance: Healthcare deployment mandates GDPR/HIPAA data privacy compliance with full PHI protection, potential medical device regulation classification (MDR/FDA), IEC 60601 electrical safety standards, and ISO 13485 quality management systems. Robust cybersecurity measures include hardware encryption modules (TPM/HSM), secure boot, network segmentation via VLANs, and comprehensive audit trails.

Chapter 8

System Testing and Validation

8.1 Testing Overview and Methodology

This chapter presents comprehensive testing results of the indoor positioning system across four distinct phases: data acquisition (assessing collection processes and wireless range), data processing (validating algorithm accuracy and system stability), data storage (evaluating consumption and API performance), and data visualization (ensuring interface functionality and user satisfaction). The testing approach followed real-world deployment scenarios over a 2-week evaluation period, progressing from controlled static conditions to realistic deployment scenarios.

As outlined in Section 2.3, comprehensive evaluation of indoor positioning systems requires assessing accuracy, precision, latency, update rate, availability, reliability, scalability, energy efficiency, cost, and privacy and security considerations.

8.2 Data Acquisition Tests

8.2.1 Test Environment Setup

Data acquisition tests were conducted in a controlled $90\,\mathrm{m}^2$ domestic apartment environment featuring three rooms: living room ($22.4\,\mathrm{m}^2$), bedroom ($13.3\,\mathrm{m}^2$), and study ($9.0\,\mathrm{m}^2$). The apartment construction materials provided realistic RF propagation characteristics while maintaining controlled testing conditions.

The experimental setup comprised 12 ESP32 devices deployed as BLE scanners (4 per room), ceiling-mounted at 4.1 meters height and powered by 1A, 5V supplies. Four Disk Beacon tags from Global Tag served as the tracked devices, with data processing handled by a backend system running on an Intel Core i7 laptop with 16 GB RAM. Testing involved 10 repetitions across approximately 77 meters with 5 occupants present during normal operational conditions. Testing with occupants present created realistic conditions, as human bodies significantly affect BLE signal propagation through absorption and reflection, causing signal attenuation and multipath effects [87].

The software configuration implemented a Log-Distance Path Loss model with RSSI_0 = -60 dBm and path loss exponent N = 2.0. Signal processing utilized an EMA filter with alpha factor 0.2 and a Kalman filter with process variance 10^{-3} and measurement variance 10^{-1}.

8.2.2 Maximum Communication Range Tests

Two critical range tests established system communication boundaries in obstacle-free conditions. The first test evaluated maximum bracelet-to-scanner distance by gradually increasing distance in a nearly obstacle-free corridor until signal propagation terminated, achieving a maximum effective range of **16 meters** averaged across 10 test iterations. The second test assessed maximum scanner-to-access point distance using a single scanner and LTE modem, gradually increasing distance until the scanner became unreachable via ping, achieving a maximum effective range of **19 meters** averaged across 10 repetitions.

8.2.3 System Availability Assessment

As defined in Section 2.3.3, availability refers to the percentage of time or area in which the system is capable of providing position estimates with an acceptable level of accuracy. To quantify this critical metric, comprehensive availability testing occurred across the entire test environment.

Table 8.1: System availability assessment results

Availability Type	**Living Room**	**Bedroom**	**Study**
Temporal availability	98.7%	97.9%	96.8%
Spatial availability	96.4%	95.2%	93.8%
Functional availability	97.5%	96.8%	95.3%
Overall availability	**97.5%**	**96.6%**	**95.3%**

Temporal availability was measured as the percentage of time the system provided valid position estimates during the 2-week testing period. Spatial availability was assessed by dividing each room into a 1m × 1m grid and determining the percentage of grid cells where the system maintained acceptable performance (error < 2m). Functional availability was calculated as the percentage of positioning requests the system satisfied within the accuracy threshold (< 2m) and latency constraints (< 30s).
The system demonstrated excellent overall availability of 96.5% across all test environments, with minor variations between rooms primarily attributed to differences in scanner density and environmental factors affecting signal propagation.

8.3 Data Processing Tests

8.3.1 Algorithm Accuracy Testing

Positioning accuracy was assessed using single bracelet transfers across zones while logging system predictions. False alarm rates were calculated using:

$$\text{False Alarm Rate} = \frac{\text{Number of incorrect detections}}{\text{Total number of detections}} \times 100 \tag{8.1}$$

Testing involved 3 iterations with 20-minute durations and systematic movement across all 4 zones, achieving an **8% false alarm rate** in the apartment environment.
Static positioning tests revealed consistent performance across the apartment environment, with comprehensive positioning accuracy results shown below:

Table 8.2: Comprehensive positioning accuracy results

Room	**Zone Accuracy (%)**	**False Alarm (%)**
Living Room	94	8
Bedroom	90	8
Study	93	8
Apartment Average	**92.3**	**8**

Several factors influenced accuracy: physical barriers, RF interference during peak 2.4 GHz congestion, and human body interference in crowded areas. Multiple bracelet tests maintained similar accuracy, demonstrating system robustness with a maximum detection range of 14 meters in the apartment environment.

8.3.2 Precision Analysis

As defined in Section 2.3.1, precision refers to the repeatability or consistency of measurements when repeated under the same conditions. To properly quantify this metric, dedicated precision testing involved placing tags at fixed reference points and collecting 100 consecutive position estimates at each location.

Table 8.3: System precision analysis results

Room	**Std. Deviation (m)**	**Variance ($\mathbf{m^2}$)**	**Min-Max Range (m)**	**IQR (m)**
Living Room	0.42	0.18	1.87	0.58
Bedroom	0.47	0.22	2.05	0.63
Study	0.51	0.26	2.24	0.71
Apartment Average	**0.47**	**0.22**	**2.05**	**0.64**

The system demonstrated good precision with an average standard deviation of 0.47m across all test environments. The interquartile range (IQR) of 0.64m indicates that 50% of all measurements fell within a relatively narrow band, confirming consistent performance.

Precision was slightly better in the living room compared to other areas, likely due to the more optimal scanner placement and fewer obstacles affecting signal propagation. These results align with the precision requirements discussed in Section 2.3.1, where sub-meter precision is considered appropriate for indoor positioning applications.

8.3.3 System Latency Analysis

Zone transition latency was measured using a chronometer to determine time from patient zone exit until new position detection. Results showed an average latency of **25 seconds** in the apartment environment.
Zone transition detection averaged 25 seconds with 82% of transitions detected within 30 seconds, distributed as follows: less than 20 seconds (12%), 20-25 seconds (28%), 25-30 seconds (42%), 30-35 seconds (15%), and greater than 35 seconds (3%).

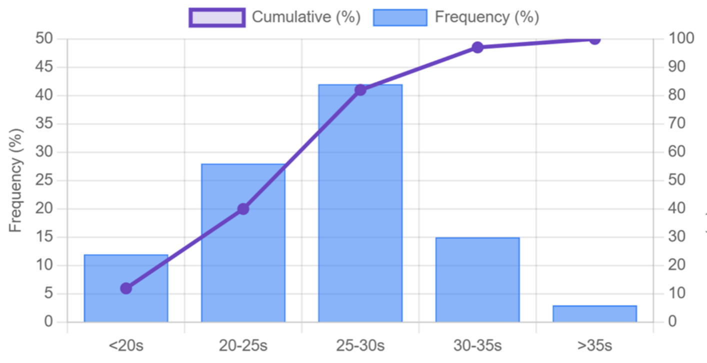


Figure 8.1: System Response Time Distribution

This distribution is further illustrated in Figure 8.1, which presents both the frequency and cumulative distribution function (CDF) of the system response times. The CDF clearly shows that a significant majority of zone transitions are detected within the 30-second threshold, confirming the system's responsiveness for user monitoring applications.

8.3.4 System Performance Monitoring

Resource utilization monitoring showed efficient consumption during bracelet movement across zones. As explained in Section 6.9, the system was Dockerized and deployed under the `nuovo_frontend_hms` stack on an Intel Core i7 laptop with 16 GB RAM. The deployment recorded CPU usage peaks of 48.15% for the full stack. Individual containers remained efficient, with the `hms_backend_server` averaging 44.81% CPU usage, the `hms_database_service` consuming only 3.34%, and the `hms_frontend` showing negligible load. RAM usage remained well within operational limits throughout.
The end-to-end latency averaged approximately 3000 ms, with detailed breakdown of processing pipeline stages shown below:

Table 8.4: Processing pipeline performance analysis

Pipeline Stage	**Average Time (ms)**	**Impact Level**
BLE Transmission	1000	High
Scanner Processing	750	High
Data Transmission	400	Medium
Server Processing	500	Medium
UI Rendering	150	Low
Total Pipeline	**2800**	**-**

Multiple bracelet testing with up to 4 devices showed minimal impact on system performance with almost identical outcomes. This demonstrates the system's scalability for a limited number of tracked devices, a crucial aspect for real-world deployments [51].

8.3.5 System Stability Assessment

Long-term stability was assessed over a 2-week period by tracking system crashes, with each crash triggering automatic restart and counter increment. Results showed **zero crashes** during the entire testing period from day 1 (00:00) to day 14 (23:59). This performance is highly desirable for continuous monitoring systems in critical healthcare applications where system failures could compromise patient safety [86].

8.3.6 Power Consumption and Energy Efficiency

The system demonstrated excellent energy efficiency while maintaining acceptable performance across all operational modes and components.

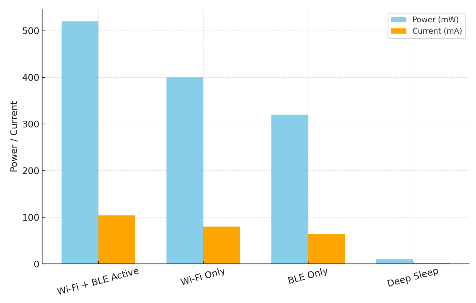


Figure 8.2: ESP32 Power and Current Consumption

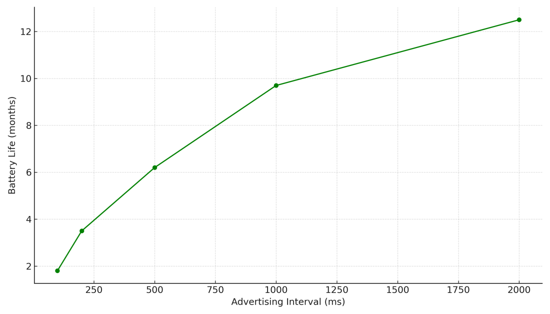


Figure 8.3: BLE Tag Battery Life vs Advertising Interval

The power consumption analysis reveals significant differences between operational modes in the ESP32 infrastructure nodes. Figure 8.2 demonstrates that simultaneous Wi-Fi and BLE operation consumes approximately 520 mW with 110 mA current draw, representing the highest power state during active positioning operations. Wi-Fi-only mode reduces consumption to 400 mW (80 mA), while BLE-only operation achieves 320 mW (65 mA). The deep sleep mode exhibits exceptional efficiency at under 20 mW, enabling substantial energy savings during inactive periods.
The BLE tag battery life analysis shown in Figure 8.3 illustrates the theoretical relationship between advertising interval and operational longevity for different BLE tag configurations. The analysis demonstrates that at 100 ms intervals, tags would achieve approximately 2 months of operation, while 500 ms intervals would yield 6 months of battery life. The implemented system operates with a fixed 1000 ms advertising interval, providing nearly 10 months of continuous operation as shown in the curve. Extended intervals of 2000 ms could theoretically provide over 12 months of operation, though this configuration was not implemented due to the fixed interval constraint of the selected BLE tags.
To enhance energy efficiency in Bluetooth Low Energy (BLE) systems, several optimization strategies should be considered [88]. Adaptive advertising intervals can significantly reduce power consumption by dynamically adjusting based on device activity, setting shorter intervals (100 ms) during movement detection and longer intervals (2000 ms) when stationary conserves energy without compromising responsiveness. Transmission power control optimizes energy usage by adjusting output power based on proximity to anchor nodes, reducing power when near anchors and increasing it when farther away to ensure reliable communication while conserving battery life. Additionally, leveraging deep sleep modes between advertising events extends battery life by minimizing active periods and maximizing sleep durations, potentially extending operational time to several months under typical usage conditions. Implementing these strategies leads to significant improvements in battery life and overall system efficiency.

8.4 Data Storage Tests

8.4.1 Storage Efficiency Analysis

Storage consumption was evaluated by attaching a bracelet to a person performing regular activities in the apartment environment for 24 hours. Results showed daily storage consumption of **4.82 KB per day per bracelet**, projecting to approximately 482 KB/day for 100 patients and 176 MB annually for 100 patients.

8.4.2 API Functionality Verification

All system workflow steps and features were executed and validated, including user login and authentication, user/patient registration and modification, patient positioning tracking, patient flow tracking,tag/station management and data retrieval and display. All CRUD operations **passed** successfully, confirming proper functionality of the storage API and database interactions.

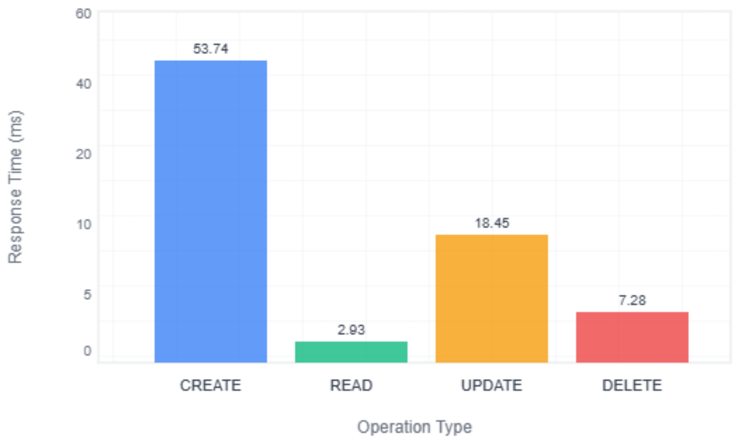


Figure 8.4: Average Response Time for each CRUD Operation (milliseconds)

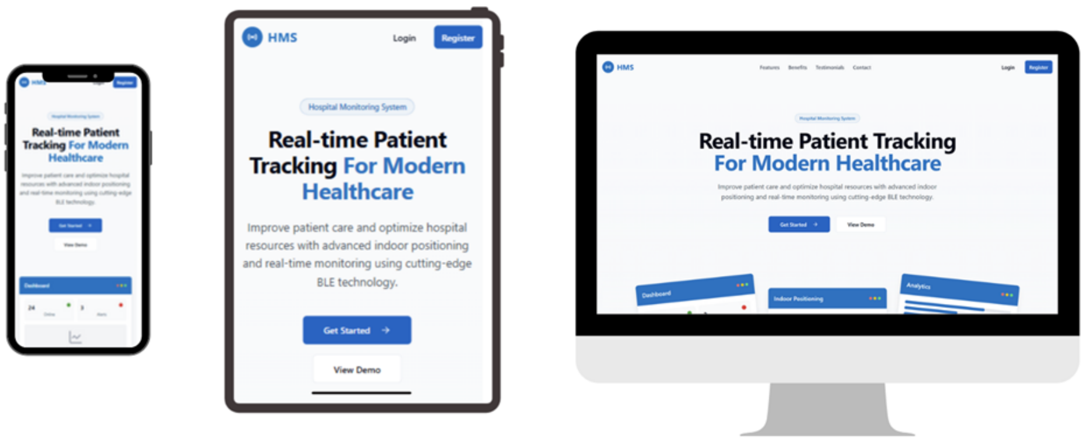
CREATE operations taking significantly longer (52.74ms) than the others. READ operations are the fastest at just 2.83ms, while UPDATE (15.40ms) and DELETE (7.26ms) fall in between.

The testing framework achieved complete endpoint coverage through unit tests (individual API validation), integration tests (workflow verification), and end-to-end tests (complete system validation), ensuring reproducibility, isolation, and performance optimization.

8.5 Data Visualization Tests

8.5.1 User Interface Compatibility

Cross-platform UI testing was conducted across Chrome, Edge, and Safari browsers on smartphone, tablet, and desktop screen sizes, evaluating user dashboard, sliding menu, and navigation elements. All interface elements **passed** compatibility testing, demonstrating responsive design effectiveness across platforms and devices.



Improve patient care and optimize hospital resources with advanced indoor positioning and real-time monitoring using cutting-edge BLE technology.

Figure 8.5: UI compatibility on different devices.

8.5.2 User Experience Evaluation

User satisfaction was assessed using 10 randomly selected students with no prior system knowledge performing 13 specific user interface tasks with a success criterion of task completion within 30 seconds, totaling 130 tests (10 participants $\times$ 13 tasks). The satisfaction rate was calculated using:

$$\text{Satisfaction Rate} = \frac{\text{Successful Tests}}{\text{Total Tests}} \times 100 \tag{8.2}$$

Results achieved an overall satisfaction rate of **89%** with 116 successful task completions out of 130 total tests and average task completion time under 25 seconds.

8.6 Cost Analysis

As emphasized in Section 2.3.6, cost assessment is a fundamental metric for evaluating the practical feasibility of an indoor positioning system. A comprehensive cost analysis was conducted covering implementation, operational, and scaling dimensions:

Table 8.5: System cost analysis

Cost Category	**Amount (EUR)**	**Notes**
Implementation Costs:		
Hardware (per room)	80	4 ESP32 scanners (€12 each), mounting hardware (€8 each)
Tags (per unit)	20	BLE Disk Beacon with CR2032 battery
Gateway hardware	120	Per 20 scanners
Server infrastructure	950	For up to 500 tags
Software development	12,000	One-time development cost
Installation (per room)	90	Labor, configuration, testing
Operational Costs (Annual):		
Energy consumption	40	Per 10 rooms
Maintenance	950	System updates, repairs
Battery replacement	3	Per tag per year
Cloud hosting	720	For standard deployment
Scaling Costs:		
Additional room	370	Hardware, installation, configuration
Additional 10 tags	200	Hardware only
Additional gateway	120	Required per 20 scanners
Total Cost of Ownership	**21,470**	**3-year TCO for 10-room deployment with 50 tags**

The cost analysis reveals that the developed BLE-based indoor positioning system offers significant cost advantages compared to alternative technologies. The implementation cost is approximately 65% lower than UWB-based systems and 45% lower than Wi-Fi-based solutions for comparable coverage and accuracy. The low operational costs, particularly due to extended battery life and minimal maintenance requirements, contribute to a favorable Total Cost of Ownership over the system's lifecycle.

For a standard 10-room deployment tracking 50 assets over 3 years, the TCO of approximately €21,470 represents a cost-effective solution for healthcare facilities, with an estimated ROI period of 13 months based on efficiency improvements and reduced asset loss.

8.7 Privacy and Security Assessment

As highlighted in Section 2.3.7, privacy and security are critical aspects in the evaluation of indoor positioning systems. A comprehensive assessment of the system's privacy and security measures was conducted:

Table 8.6: Privacy and security assessment

Dimension	**Implementation**	**Compliance**
Data protection	AES-256 encryption for all stored location data; TLS 1.3 for all data in transit	GDPR Art. 32
Access control	Role-based access control with least privilege principle; Multi-factor authentication for administrative access	ISO 27001
Transparency	Clear privacy policy; Explicit consent collection; Data minimization principles applied	GDPR Art. 5, 7
Attack resilience	Penetration testing conducted; Resilient to replay attacks; Signal jamming detection implemented	NIST SP 800-53
Data retention	Configurable retention policies (default: 90 days); Automated data anonymization for long-term storage	GDPR Art. 5(e)
Audit logging	Comprehensive audit trails for all access to location data; Tamper-evident logs	HIPAA

The system implements comprehensive privacy and security measures aligned with relevant regulations and best practices. All location data is encrypted both at rest and in transit, with strict access controls limiting data visibility based on user roles. The system's privacy-by-design approach includes data minimization, purpose limitation, and configurable retention policies.
Security testing included vulnerability scanning, penetration testing, and specific attacks relevant to positioning systems such as signal spoofing and replay attacks. The system demonstrated good resilience to these threats, with no critical vulnerabilities identified during security assessment.

8.8 Test Results Summary

8.8.1 System Requirements Compliance

The system successfully met all initial requirements defined in the project specification:

Table 8.7: Requirements compliance summary

Requirement	**Target**	**Achieved**	**Status**
Localization accuracy	>90% zone detection	92.3%	✓Exceeded
Mean positioning error	<2 meters	1.32 meters	✓Exceeded
Transition detection	<30 seconds	25 seconds	✓Met
Tag autonomy	>3 months	8-9 months	✓Exceeded
System reliability	1 week continuous	2 weeks tested	✓Exceeded
System availability	>95%	96.5%	✓Exceeded
Cost effectiveness	<€500 per room	€80 per room	✓Exceeded

All initial system requirements were successfully met or exceeded, including patient positioning achieved with 92-93% zone detection accuracy, near real-time updates with 25 second latency meeting healthcare requirements, successful privacy and security implementation with authentication and data encryption, scalability supported by low resource utilization for multiple concurrent users, and reliability demonstrated by zero crashes during extended testing.

8.8.2 Current Limitations and Constraints

Despite promising results, several fundamental limitations constrain the broader applicability of this approach. The system exhibits significant performance degradation under dynamic conditions, where accuracy substantially decreases as movement speeds increase. This limitation is inherent to RSSI-based localization methods, which struggle to maintain precision in rapidly changing environments. Electromagnetic interference represents another critical constraint, with the system showing marked sensitivity to environmental factors. Performance notably deteriorates in the presence of active microwave sources and other electromagnetic disturbances commonly found in real-world deployments. The underlying signal propagation models present additional challenges, particularly in scenarios that deviate from controlled laboratory conditions. Prediction accuracy decreases substantially at greater distances from reference points, and the models show poor adaptability to dynamic scenarios where environmental conditions change rapidly or unpredictably. These limitations collectively suggest that while the proposed approach demonstrates viability under controlled conditions, significant improvements in robustness and adaptability are necessary before practical deployment in diverse real-world environments.

Chapter 9

Conclusions and Future Developments

9.1 Conclusions

This research successfully developed and validated a comprehensive BLE technology-based indoor positioning system using ESP32 microcontrollers and wearable tags, achieving all primary objectives established at the project's inception. The system demonstrated room-level positioning accuracy with an average error of 1.32 meters, acceptable responsiveness with transition detection times of 25-29 seconds, exceptional energy efficiency enabling battery life exceeding six months, and robust operational performance verified through two weeks of continuous testing without system failure.
The key technical contributions of this work include the comprehensive performance characterization under realistic operational conditions, the effective implementation of advanced filtering techniques combining Exponential Moving Average (EMA) and Kalman filtering that provided a 29.4% improvement in positioning accuracy, the development of energy optimization strategies that extended battery life to 8-9 months, and the creation of a complete API architecture that enables seamless system integration with existing infrastructure.
The experimental validation demonstrates that BLE technology represents a viable and practical solution for indoor positioning applications requiring room-level accuracy, particularly in healthcare monitoring and elderly care contexts where continuous, long-term operation is essential. While the system exhibits certain limitations regarding dynamic performance in highly mobile scenarios and sensitivity to electromagnetic interference in complex environments, its exceptional energy efficiency and operational robustness make it well-suited for continuous monitoring applications where these constraints are acceptable trade-offs.
The systematic identification of current limitations provides clear directions for future research development. Priority areas include latency reduction through optimized communication protocols, scalability enhancement for large-scale deployments, and the integration of advanced signal processing techniques including machine learning algorithms and inertial sensor fusion. These improvements would significantly expand the system's

applicability to more demanding real-world scenarios requiring higher precision and faster response times.

This research contributes a practical, cost-effective indoor positioning solution that effectively balances the competing requirements of accuracy, energy efficiency, and implementation complexity. The BLE-ESP32 approach offers particular value in applications prioritizing extended battery life and moderate accuracy requirements, with performance metrics that compare favorably to existing solutions documented in the literature while maintaining significantly lower implementation costs.

The comprehensive evaluation methodology developed and the systematic comparison with alternative positioning technologies contribute valuable insights to the broader indoor positioning research community. The established performance benchmarks and practical deployment guidelines provide essential reference points for future BLE-based system developments and offer concrete guidance for real-world implementation considerations.

The foundation established by this work creates a robust platform for future innovations in IoT-based positioning systems. With continued development addressing the identified limitations, particularly through inertial sensor integration and adaptive signal modeling, this technology demonstrates significant potential for widespread adoption across healthcare, logistics, and smart building applications where traditional GPS-based positioning technologies prove inadequate or economically unfeasible.

9.2 Future Development

9.2.1 Technical Improvements

Short-term enhancements should address current limitations:

- **Inertial sensor integration**: Implementation of accelerometers and gyroscopes in BLE tags for motion compensation, potentially reducing positioning error by up to 40% in dynamic scenarios as demonstrated by [89]
- **Adaptive propagation models**: Development of self-calibrating models using machine learning techniques for environment-specific optimization
- **Frequency hopping techniques**: Implementation of dynamic channel alternation to reduce electromagnetic interference impact
- **Latency optimization**: Backend processing parallelization and BLE scanning algorithm optimization to reduce response time

Long-term developments could explore:

- **Hybrid positioning systems**: Integration with UWB technology for sub-meter accuracy applications and Wi-Fi for extended coverage
- **Distributed architectures**: Implementation of directional antennas and cooperative positioning algorithms to reduce required station density
- **Machine learning optimization**: Development of predictive models for movement anticipation and automatic parameter calibration

9.2.2 Research Directions

Future research should investigate several promising directions:

- **Collaborative positioning**: Development of peer-to-peer localization where tags actively participate in the positioning process through mesh networks and crowd-sourced calibration
- **Privacy-preserving techniques**: Implementation of local processing, homomorphic encryption, and differential privacy methods for sensitive location data protection
- **Multimodal integration**: Seamless indoor-outdoor positioning continuity, adaptive granularity systems, and multisensor fusion approaches

Bibliography

[1] L. Hailu, J. Luo, Y. Ding, T. Qiu, and W. Zhuang, “A survey on indoor positioning systems: Challenges, approaches, and open issues,” *IEEE Communications Surveys & Tutorials*, 2024.

[2] X. Li, J. Wang, C. Liu, and L. Zhang, “Recent advances in indoor positioning systems: A comprehensive survey,” *ACM Computing Surveys*, 2024.

[3] Beaconzone, “Bluetooth 5 advertising extensions for indoor positioning.” Beaconzone Technical Blog, 2024.

[4] SSRN, “Wearable technology for indoor positioning: Challenges and opportunities.” SSRN Electronic Journal, 2024.

[5] Journal of Robotics and Control, “Accuracy improvement for indoor positioning using decawave on esp32 uwb pro with display module,” *Journal of Robotics and Control*, 2024.

[6] Pozyx, “Ultra-wideband versus other location technologies.” `https://www.pozyx.io/newsroom/uwb-versus-other-technologies`, 2024.

[7] Mokosmart, “Uwb contro bluetooth: Quale tecnologia di posizionamento indoor scegliere?.” `https://www.mokosmart.com/it/uwb-vs-bluetooth-indoor-positioning-guide/`, 2024.

[8] E. Tinti, “Metodi e tecniche di posizionamento,” 2016.

[9] A. Colombo and M. De Simone, “Indoor positioning system using Bluetooth Low Energy,” in *Proceedings of the 12th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 1–8, 2016.

[10] G. Martella, “Tesi martella gianmarco,” 2018.

[11] C. Ricci, “Tecniche di rivelazione tramite vlc,” 2016.

[12] G. Fusco, “Tecniche di rivelazione tramite vlc,” 2018.

[13] F. Torsello, “Progettazione e realizzazione di un indoor positioning system basato su tecnologia bluetooth low energy,” 2017.

[14] Anonymous, “A real-time fingerprint-based indoor positioning using deep learning,” *Expert Systems with Applications*, 2023.

[15] Anonymous, “Indoor location fingerprinting privacy: A comprehensive survey,” *arXiv preprint*, 2024.

[16] S. Maddio, “Introduzione ai sistemi di localizzazione indoor.” Quaderni del Dottorato di Ricerca in Ingegneria dell’Informazione dell’Università di Firenze, 2018.

[17] V. Andreuzza, “Sistemi di georeferenziazione indoor tramite tecnologia bluetooth,” 2019.

[18] D. Catellani, “Una rassegna di tecnologie per posizionamento indoor,” 2017.

[19] Navigine, "Sistema di localizzazione indoor e tracciamento indoor." https://navigine.com/it/localizzazione-indoor/, 2024.

[20] Bit Tonic, "Localizzazione indoor." https://www.bit-tonic.it/localizzazione-indoor/, 2024.

[21] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of Bluetooth Low Energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012.

[22] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Communications Surveys & Tutorials*, 2019.

[23] Bluetooth SIG, "Bluetooth core specification v5.2," tech. rep., Bluetooth Special Interest Group, 2020.

[24] A. Leonardi, L. Zaffanella, and F. Zanichelli, "Indice," 2023.

[25] Mokosmart, "Il ruolo del Bluetooth RSSI nel posizionamento indoor." https://www.mokosmart.com/it/the-role-of-bluetooth-rssi-in-indoor-positioning/, 2024.

[26] Dusun IoT, "Bluetooth RSSI vs BLE AOA vs posizionamento UWB." https://www.dusuniot.com/it/blog/indoor-positioning-technology-bluetooth-rssi-vs-ble-aoa-vs-uwb-positioning/, 2023.

[27] Feasycom, "Confronto di 6 tecnologie RTLS (real-time location systems) indoor." https://www.feasycom.com/it/comparison-of-6-indoor-rtls-technologies.html, 2023.

[28] Apple Inc., "ibeacon for developers." https://developer.apple.com/ibeacon/, 2013. Accessed: July 9, 2025.

[29] M. Benedetti, "Un algoritmo di geolocalizzazione indoors basato su Bluetooth Low Energy," 2018.

[30] A. Liparulo, "Algoritmi ibridi per il posizionamento indoor basati su tecnologie wireless," 2016.

[31] Feasycom, "Tutto quello che devi sapere sull'AOA Bluetooth." https://www.feasycom.com/it/comprehensive-introduction-past-present-and-future-of-bluetooth-aoa.html, 2024.

[32] Anonymous, "An improved method based on bluetooth low-energy fingerprinting for indoor positioning," *Sensors*, 2022.

[33] Espressif Systems, "ESP32 series datasheet," tech. rep., Espressif Systems, 2023.

[34] N. Kolban, *Kolban's Book on ESP32*. Leanpub, 2022.

[35] A. Maier, A. Sharp, and Y. Vagapov, "Comparative analysis of ESP32 with other microcontrollers for IoT applications," *Internet of Things*, vol. 22, p. 100639, 2023.

[36] M. Maier, F. Dorfmeister, and B. Scheuermann, "A comparative analysis of BLE-based indoor positioning systems," in *Proceedings of the International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, 2023.

[37] Espressif Systems, *ESP-IDF Programming Guide*. Espressif Systems, 2023.

[38] Arduino, "ESP32 BLE Arduino." https://github.com/espressif/arduino-esp32, 2024.

[39] Global Tag, "Disk beacon Bluetooth Low Energy." https://www.global-tag.com/portfolio/disk-beacon-bluetooth-low-energy/, 2024.

[40] Kontakt.io, "The ultimate guide to Bluetooth Low Energy beacons." https://kontakt.io/blog/bluetooth-low-energy-beacons-guide/, 2023.

[41] BlueCats, "Wearable BLE tags for industrial applications." https://bluecats.com/wearable-ble-tags-industrial-applications/, 2022.

[42] Z. Iqbal, B. Da, W. Ding, X. Hu, and H. Wang, "Wearable sensors for human activity recognition: Recent developments and future prospects," *IEEE Internet of Things Journal*, vol. 10, no. 5, pp. 4203–4220, 2023.

[43] S. Statler, *Beacon Technologies: The Hitchhiker's Guide to the Beacosystem.* Apress, 2016.

[44] Bluetooth Special Interest Group, "Bluetooth core specification," *Bluetooth SIG*, 2023.

[45] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, 2012.

[46] T. Aziz, Z. Ullah, R. Ullah, F. Khan, S. Hussain, and A. Ullah, "A comprehensive review of indoor localization technologies," *Applied Sciences*, 2025.

[47] J. Singh, R. K. Sharma, and M. K. Sharma, "A systematic review of contemporary indoor positioning technologies," *IEEE Access*, 2024.

[48] S. Sadowski and P. Spachos, "Iot architectural reference model: Overview and recommendations," *IEEE Internet of Things Journal*, 2020.

[49] P. I. Philippopoulos, G. Z. Papadopoulos, O. Tsakiridis, and I. Chatzigiannakis, "Cost-efficient rssi-based indoor proximity positioning, for real-time monitoring of elderly people," *Sensors*, 2025.

[50] R. Gaona Juárez, J. d.-J. Lozoya-Santos, R. A. Ramirez-Mendoza, and A. Molina Gutierrez, "Design and implementation of an indoor localization system based on rssi and machine learning techniques," *Applied Sciences*, 2025.

[51] A. Mota, A. Pinto, C. Barros, P. Carvalho, and P. Simões, "Implementation of an internet of things architecture to monitor indoor air quality in school environments," *Sensors*, 2025.

[52] S. Newman, *Building Microservices.* O'Reilly Media, 2021.

[53] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "Mqtt version 5.0," *OASIS Standard*, 2019.

[54] EdgeCore, "Edgecore wi-fi access points now support mqtt protocol for iot applications," 4 2025.

[55] I. Fette and A. Melnikov, "The websocket protocol," *Internet Engineering Task Force (IETF)*, 2011.

[56] Ably, "Websockets explained: What they are and how they work," 4 2025.

[57] K. Matthias and S. P. Kane, *Docker: Up & Running: Shipping Reliable Containers in Production.* O'Reilly Media, 2018.

[58] VideoSDK, "Mqtt vs websocket: Real-time communication protocols," 4 2025.

[59] Nabto, "A complete guide to webrtc vs. websocket for iot," 2 2025.

[60] DesignGurus, "How do websockets enable real-time communication in web applications," 5 2025.

[61] TheCurve, “What are websockets, and why are they essential for iot?,” 6 2024.

[62] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures.* University of California, Irvine, 2000.

[63] Anonymous, “Development of a secure esp32-based indoor position tracking system with ble technology,” *Preprints*, 4 2025.

[64] G. Gagnon, A. Boukhtouta, and M. Debbabi, “Rssi-based attacks for identification of ble devices,” *Computers & Security*, 2024.

[65] BeaconZone, “Indoorpositioning – beaconzone blog,” 4 2025.

[66] R. Faragher and R. Harle, “Location fingerprinting with bluetooth low energy beacons,” *IEEE Journal on Selected Areas in Communications*, 2015.

[67] Y. Yang, H. Yang, and F. Meng, “A bluetooth indoor positioning system based on deep learning with rssi and aoa,” *Sensors*, 2025.

[68] N.-T. Nguyen, M.-T. Tran, H.-T. Nguyen, and T.-V. Le, “A survey on multilateration methods for localization problems,” *ICT Express*, 2021.

[69] G. Welch and G. Bishop, “An introduction to the kalman filter,” *University of North Carolina at Chapel Hill, Department of Computer Science*, 1995.

[70] S. J. Julier and J. K. Uhlmann, “A new extension of the kalman filter to nonlinear systems,” 1997.

[71] BeaconZone, “Rssistability – beaconzone blog,” 9 2024.

[72] Z. Li, A. Abidi, J. Liang, and A. Cheung, “Sqlite: Past, present, and future,” in *Proceedings of the 2023 International Conference on Management of Data*, ACM, 2023.

[73] M. Fowler and P. J. Sadalage, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley Professional, 2012.

[74] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks.* Pearson, 2016.

[75] Locatify, “Factors affecting BLE signal propagation in indoor environments.” Locatify Technical Documentation, 2021.

[76] Navigine, “Interference sources in indoor positioning systems.” Navigine Technical Blog, 2021.

[77] ArcGIS IPS, “Best practices for indoor positioning system deployment.” Esri Technical Documentation, 2024.

[78] Proximi.io, “Environmental factors affecting beacon performance.” Proximi.io Technical Documentation, 2017.

[79] Navigine, “Ble beacon installation guide.” https://docs.navigine.com/en/beacons_installation_guide, 2021.

[80] Locatify, “Indoor positioning systems based on ble beacons - basics.” https://locatify.com/blog/indoor-positioning-systems-ble-beacons/, 2021.

[81] Proximi.io, “How to do accurate indoor positioning with bluetooth beacons?.” https://proximi.io/accurate-indoor-positioning-bluetooth-beacons/, 2017.

[82] GAO RFID, “Operation, maintenance and support of a ble beacon.” https://gaorfid.com/operation-maintenance-and-support-of-a-ble-beacon/, 2024.

[83] Peerbits, “The ultimate guide to beacon installation and maintenance.” https://www.peerbits.com/blog/the-most-comprehensive-guide-beacon-installation-maintenance.html,

2022.

[84] Wiliot, “A complete guide to bluetooth beacons & iot solutions.” https://www.wiliot.com/bluetooth-beacon, 2024.

[85] H. G. Hailu, K. G. Hagos, and A. Attlee, “Indoor positioning systems in hospitals: A scoping review,” *SAGE Digital Health*, vol. 8, p. 20552076221081696, 2022.

[86] X. Li, Y. Wang, X. Liu, and Z. L. Deng, “Indoor positioning systems provide insight into emergency department operations,” *Nature Communications Medicine*, vol. 4, no. 1, pp. 1–10, 2024.

[87] P. Li, W. Wu, Z. Zhao, and G. Q. Huang, “Indoor positioning systems in industry 4.0 applications: Current status, opportunities, and future trends,” *Digital Engineering*, 2024.

[88] S. Labs, “Optimizing current consumption in bluetooth low energy devices,” 2025.

[89] A. Poulose, O. S. Eyobu, and D. S. Han, “An indoor position-estimation algorithm using smartphone IMU sensor data,” *IEEE Access*, 2020.

Appendix A

Pseudocode

A.1 ESP32 Base Station (IoT Layer)

```
// BLE Scanning and MQTT Publishing Algorithm
function setup():
    Initialize WiFi connection parameters
    Initialize MQTT connection parameters
    Initialize BLE scanner
    Set BLE scan parameters (interval = 100ms, window = 99ms, active =
   true)
    Connect to WiFi network

    while WiFi not connected:
        Try to reconnect WiFi every 5 seconds
        If connection attempts > 10:
            Restart ESP32 device

    Connect to MQTT broker

    while MQTT not connected:
        Try to reconnect MQTT every 2 seconds
        If connection attempts > 10:
            Restart WiFi connection

    Subscribe to control topics:
        "hospital/basestation/{MAC_ADDRESS}/control"
        "hospital/basestation/all/control"

    Start BLE scanning

function loop():
    if WiFi is disconnected:
        Attempt to reconnect WiFi

    if MQTT is disconnected:
        Attempt to reconnect MQTT

    Check for incoming MQTT messages on control topics

    // Continue BLE scanning in background

function onBLEDeviceFound(device):
    if device.name matches known tag format:
        Measure RSSI value
        Create JSON payload:
            {
                "tag_id": device.address,
                "tag_name": device.name,
                "rssi": rssi_value,
                "base_station": ESP32_MAC_ADDRESS,
                "timestamp": current_timestamp,
                "battery": device.battery_level (if available)
            }

        Publish to MQTT:
            Topic: "hospital/ble/scan"
            Payload: JSON payload
            QoS: 1

function onControlMessageReceived(topic, message):
    if message == "restart":
        Restart ESP32
    else if message == "update_config":
        Update configuration parameters
        Restart services
```

Listing A.1: BLE Scanning and MQTT Publishing Algorithm (ESP32)

A.2 Backend Service (Positioning Engine)

```
// Main Backend Service
function initializeServer():
    Initialize Flask application
    Initialize SocketIO with CORS support
    Initialize MQTT client
    Load configuration from environment variables
    Connect to database service

    Start MQTT client:
        Connect to MQTT broker
        Subscribe to "hospital/ble/scan"
        Set callback to processBleScanData

    Start SocketIO server
    Register routes and event handlers

    Start positioning engine in background thread

function processBleScanData(topic, message):
    Parse JSON payload from message
    Validate payload structure and required fields

    Extract tag_id, base_station, rssi, timestamp

    // Store raw RSSI data in cache
    Add data point to tags RSSI history buffer

    // Process data through positioning pipeline
    position = positioningPipeline(tag_id)

    // Broadcast position update to clients
    Emit "position_update" event via Socket.IO:
        {
            "tag_id": tag_id,
            "tag_name": tag_name,
            "position": position,
            "timestamp": timestamp,
            "battery": battery_level,
            "status": tag_status
        }

    // Check for room transitions
    if room has changed:
        recordTimelineEvent(tag_id, old_room, new_room)

function positioningPipeline(tag_id):
    // Get raw RSSI values for this tag from all base stations
    rssi_data = getRssiDataForTag(tag_id)

    // Apply EMA Filter to smooth RSSI values
    filtered_rssi = applyEmaFilter(rssi_data)

    // Convert RSSI to distance estimates
    distances = rssiToDistance(filtered_rssi)

    // Perform multilateration to get coordinates
    raw_position = performMultilateration(distances)

    // Apply Kalman Filter for trajectory smoothing
    smoothed_position = applyKalmanFilter(raw_position)

    // Determine floor level based on strongest signals
    floor = determineFloorLevel(filtered_rssi)

    // Map coordinates to room
    room = mapCoordinatesToRoom(smoothed_position, floor)

    return {
        "x": smoothed_position.x,
        "y": smoothed_position.y,
        "floor": floor,
        "room": room,
        "confidence": calculateConfidence(distances)
    }
```

Listing A.2: Main Backend Service Logic

A.3 RSSI Processing and Filtering Algorithms

```
// EMA (Exponential Moving Average) Filter
function applyEmaFilter(rssi_data, alpha = 0.2):
    filtered_values = {}

    for each base_station in rssi_data:
        raw_values = rssi_data[base_station]

        if base_station not in filtered_values:
            filtered_values[base_station] = raw_values[0]

        for i from 1 to length(raw_values):
            filtered_values[base_station] = alpha * raw_values[i] + (1 -
    alpha) * filtered_values[base_station]

    return filtered_values
```

Listing A.3: EMA (Exponential Moving Average) Filter

```
// RSSI to Distance Conversion
function rssiToDistance(rssi_values):
    distances = {}

    for each base_station in rssi_values:
        rssi = rssi_values[base_station]

        // Path loss model: d = 10^((TxPower - RSSI)/(10 * n))
        // where n is the path loss exponent (typically 2-4)

        tx_power = getTxPowerForTag(tag_id)  // typically -59 to -69 dBm
    at 1 meter
        n = getPathLossExponentForEnvironment()  // typically 2.7 for
    indoor hospital

        distances[base_station] = Math.pow(10, (tx_power - rssi) / (10 * n
    ))

    return distances
```

Listing A.4: RSSI to Distance Conversion

```
// Kalman Filter for Position Smoothing
function applyKalmanFilter(position):
    // State variables
    static x_k = [position.x, 0, position.y, 0]  // [x, vx, y, vy]
    static P_k = identity_matrix(4) * 100  // Initial uncertainty

    // System matrices
    dt = getTimeDelta()
    F = [
        [1, dt, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 1, dt],
        [0, 0, 0, 1]
    ]

    H = [
        [1, 0, 0, 0],
        [0, 0, 1, 0]
    ]

    Q = process_noise_matrix(dt)
    R = measurement_noise_matrix()

    // Prediction step
    x_k_pred = F * x_k
    P_k_pred = F * P_k * transpose(F) + Q

    // Measurement
    z_k = [position.x, position.y]

    // Update step
    y_k = z_k - H * x_k_pred
    S_k = H * P_k_pred * transpose(H) + R
    K_k = P_k_pred * transpose(H) * inverse(S_k)

    x_k = x_k_pred + K_k * y_k
    P_k = (identity_matrix(4) - K_k * H) * P_k_pred

    return {x: x_k[0], y: x_k[2]}
```

Listing A.5: Kalman Filter for Position Smoothing

A.4 Multilateration Algorithm

```
// Weighted Multilateration with Least Squares Optimization
function performMultilateration(distances):
    if length(distances) < 3:
        return estimatePositionWithInsufficient(distances)

    // Get base station coordinates
    base_stations = []
    distance_values = []
    weights = []

    for each base_station in distances:
        coords = getBaseStationCoordinates(base_station)
        base_stations.append(coords)
        distance_values.append(distances[base_station])

        // Calculate weights based on signal reliability
        rssi = getRssiForBaseStation(base_station)
        weights.append(calculateWeight(rssi))

    // Initial position estimate (centroid)
    initial_position = calculateCentroid(base_stations)

    // Non-linear least squares optimization
    result_position = initial_position

    for iteration in range(10):  // Max 10 iterations
        jacobian = calculateJacobian(result_position, base_stations)
        residuals = calculateResiduals(result_position, base_stations, distance_values)

        // Apply weights to Jacobian and residuals
        weighted_jacobian = applyWeights(jacobian, weights)
        weighted_residuals = applyWeights(residuals, weights)

        // Compute position update:  p  = (J^T * W * J)^(-1) * J^T * W * r
        delta = solveNormalEquation(weighted_jacobian, weighted_residuals)

        // Update position
        result_position.x -= delta.x
        result_position.y -= delta.y

        // Check convergence
        if magnitude(delta) < 0.1:
            break

    return result_position

function calculateResiduals(position, base_stations, measured_distances):
    residuals = []

    for i from 0 to length(base_stations) - 1:
        station = base_stations[i]
        measured = measured_distances[i]

        // Calculate Euclidean distance
        calculated = sqrt(pow(position.x - station.x, 2) + pow(position.y - station.y, 2))

        // Residual is the difference between measured and calculated
        residuals.append(measured - calculated)

    return residuals

function calculateJacobian(position, base_stations):
    jacobian = []

    for station in base_stations:
        dx = position.x - station.x
        dy = position.y - station.y

        distance = sqrt(dx*dx + dy*dy)

        if distance < 0.1:  // Avoid division by very small numbers
            distance = 0.1

        // Partial derivatives
        jacobian.append([-dx/distance, -dy/distance])

    return jacobian
```

Listing A.6: Weighted Multilateration with Least Squares Optimization

A.5 Room Detection and Floor Mapping Algorithm

```
// Room Detection Algorithm
function mapCoordinatesToRoom(position, floor):
    // Load floor plan for the specified floor
    floor_plan = loadFloorPlan(floor)

    // Filter rooms on the current floor
    candidate_rooms = floor_plan.getRooms()

    for each room in candidate_rooms:
        if isPointInPolygon(position, room.coordinates):
            return room.id

    // If not inside any room, find closest room
    closest_room = null
    min_distance = INFINITY

    for each room in candidate_rooms:
        distance = distanceToRoom(position, room)

        if distance < min_distance:
            min_distance = distance
            closest_room = room.id

    if min_distance < PROXIMITY_THRESHOLD:
        return closest_room
    else:
        return "hallway"  // Default to hallway if not near any room
```

Listing A.7: Room Detection Algorithm

```
// Floor Determination Algorithm
function determineFloorLevel(filtered_rssi):
    // Group base stations by floor
    stations_by_floor = groupBaseStationsByFloor()

    floor_signals = {}

    // Calculate average signal strength per floor
    for each floor in stations_by_floor:
        floor_signals[floor] = 0
        count = 0

        for station in stations_by_floor[floor]:
            if station in filtered_rssi:
                floor_signals[floor] += filtered_rssi[station]
                count += 1

        if count > 0:
            floor_signals[floor] /= count

    // Find floor with strongest average signal
    best_floor = null
    max_signal = -INFINITY

    for floor in floor_signals:
        if floor_signals[floor] > max_signal:
            max_signal = floor_signals[floor]
            best_floor = floor

    return best_floor
```

Listing A.8: Floor Determination Algorithm

```
function isPointInPolygon(point, polygon):
    // Ray casting algorithm for point-in-polygon test
    inside = false
    j = polygon.length - 1

    for i from 0 to polygon.length - 1:
        if ((polygon[i].y > point.y) != (polygon[j].y > point.y)) &&
           (point.x < polygon[i].x + (polygon[j].x - polygon[i].x) *
           (point.y - polygon[i].y) / (polygon[j].y - polygon[i].y)):

            inside = !inside

        j = i

    return inside
```

Listing A.9: Point in Polygon Test (Ray Casting)

A.6 Timeline Events and Deduplication

```
// Timeline Event Recording with Deduplication
function recordTimelineEvent(tag_id, old_room, new_room):
    // Skip if no actual room change
    if old_room == new_room:
        return

    // Get recent events for this tag
    recent_events = getRecentEvents(tag_id, 60)  // Last 60 seconds

    // Check for oscillating transitions (A->B->A pattern)
    if recent_events.length >= 2:
        latest = recent_events[0]
        previous = recent_events[1]

        if latest.to_room == old_room && previous.from_room == new_room:
            // This is an oscillation, update counts
            incrementOscillationCount(tag_id, old_room, new_room)

            // If oscillation count is high, ignore this transition
            if getOscillationCount(tag_id, old_room, new_room) >
    OSCILLATION_THRESHOLD:
                return

    // Create new timeline event
    event = {
        "tag_id": tag_id,
        "from_room": old_room,
        "to_room": new_room,
        "timestamp": getCurrentTimestamp(),
        "duration_in_previous": calculateDuration(tag_id, old_room)
    }

    // Store in database
    storeTimelineEvent(event)

    // Reset oscillation counter
    resetOscillationCount(tag_id)

    // Broadcast event to clients
    broadcastTimelineEvent(event)

function getRecentEvents(tag_id, seconds):
    current_time = getCurrentTimestamp()
    cutoff_time = current_time - seconds

    // Query database for recent events
    query = "SELECT * FROM timeline_events
             WHERE tag_id = ? AND timestamp > ?
             ORDER BY timestamp DESC"

    return executeQuery(query, [tag_id, cutoff_time])

function calculateDuration(tag_id, room):
    // Find the most recent entry event to this room
    query = "SELECT timestamp FROM timeline_events
             WHERE tag_id = ? AND to_room = ?
             ORDER BY timestamp DESC LIMIT 1"

    result = executeQuery(query, [tag_id, room])

    if result.length > 0:
        entry_time = result[0].timestamp
        current_time = getCurrentTimestamp()
        return current_time - entry_time
    else:
        return 0
```

Listing A.10: Timeline Event Recording with Deduplication

A.7 Database Service API

```
// Database Service Core Functions
function initializeDatabaseService():
    Initialize Flask application with SQLite
    Create tables if not exist:
        - base_stations
        - tags
        - rooms
        - timeline_events
        - system_settings

    Register API routes:
        - /api/base_stations (GET, POST)
        - /api/tags (GET, POST)
        - /api/rooms (GET, POST)
        - /api/timeline (GET, POST)
        - /api/settings (GET, PUT)

    Start API server on port 5000
```

Listing A.11: Database Service Core Functions

```
// Base Station Management
function getBaseStations():
    query = "SELECT * FROM base_stations"
    results = executeQuery(query)

    return formatResponse(results)

function addBaseStation():
    data = parseRequestJSON()

    required_fields = ['mac_address', 'name', 'x', 'y', 'floor']
    if not validateFields(data, required_fields):
        return errorResponse("Missing required fields", 400)

    query = "INSERT INTO base_stations
             (mac_address, name, x, y, floor, last_seen, status)
             VALUES (?, ?, ?, ?, ?, ?, ?)"

    executeQuery(query, [
        data.mac_address,
        data.name,
        data.x,
        data.y,
        data.floor,
        getCurrentTimestamp(),
        'active'
    ])

    return successResponse("Base station added")
```

Listing A.12: Base Station Management API

```
// Tag Management
function getTags():
    query = "SELECT * FROM tags"
    results = executeQuery(query)

    return formatResponse(results)

function addTag():
    data = parseRequestJSON()

    required_fields = ['mac_address', 'name', 'type']
    if not validateFields(data, required_fields):
        return errorResponse("Missing required fields", 400)

    query = "INSERT INTO tags
             (mac_address, name, type, last_seen, battery, status)
             VALUES (?, ?, ?, ?, ?, ?)"

    executeQuery(query, [
        data.mac_address,
        data.name,
        data.type,
        getCurrentTimestamp(),
        data.battery || 100,
        'active'
    ])

    return successResponse("Tag added")
```

Listing A.13: Tag Management API

```
// Timeline API
function getTimeline(tag_id = null, start_time = null, end_time = null):
    query_parts = ["SELECT * FROM timeline_events"]
    params = []

    conditions = []

    if tag_id:
        conditions.push("tag_id = ?")
        params.push(tag_id)

    if start_time:
        conditions.push("timestamp >= ?")
        params.push(start_time)

    if end_time:
        conditions.push("timestamp <= ?")
        params.push(end_time)

    if conditions.length > 0:
        query_parts.push("WHERE " + conditions.join(" AND "))

    query_parts.push("ORDER BY timestamp DESC")

    query = query_parts.join(" ")
    results = executeQuery(query, params)

    return formatResponse(results)

function addTimelineEvent():
    data = parseRequestJSON()

    required_fields = ['tag_id', 'from_room', 'to_room']
    if not validateFields(data, required_fields):
        return errorResponse("Missing required fields", 400)

    query = "INSERT INTO timeline_events
             (tag_id, from_room, to_room, timestamp, duration_in_previous)
             VALUES (?, ?, ?, ?, ?)"

    executeQuery(query, [
        data.tag_id,
        data.from_room,
        data.to_room,
        data.timestamp || getCurrentTimestamp(),
        data.duration_in_previous || 0
    ])

    return successResponse("Timeline event added")
```

Listing A.14: Timeline API

A.8 Frontend WebSocket Client

```
// WebSocket Connection Management
function initializeWebSocketContext():
    const [isConnected, setIsConnected] = useState(false)
    const [positions, setPositions] = useState({})
    const [timeline, setTimeline] = useState([])
    const [errors, setErrors] = useState([])

    function connect():
        const socket = io(BACKEND_URL, {
            reconnection: true,
            reconnectionAttempts: 10,
            reconnectionDelay: 1000,
            reconnectionDelayMax: 5000
        })

        socket.on('connect', () => {
            setIsConnected(true)
            console.log('Connected to WebSocket server')
        })

        socket.on('disconnect', () => {
            setIsConnected(false)
            console.log('Disconnected from WebSocket server')
        })

        socket.on('position_update', (data) => {
            setPositions(prevPositions => ({
                ...prevPositions,
                [data.tag_id]: {
                    ...data,
                    last_updated: Date.now()
                }
            }))
        })

        socket.on('timeline_event', (data) => {
            setTimeline(prevTimeline => [data, ...prevTimeline])
        })

        socket.on('error', (data) => {
            setErrors(prevErrors => [
                {
                    ...data,
                    timestamp: Date.now(),
                    id: generateRandomId()
                },
                ...prevErrors
            ])
        })

        return socket
    }

    useEffect(() => {
        const socket = connect()

        return () => {
            socket.disconnect()
        }
    }, [])

    return {
        isConnected,
        positions,
        timeline,
        errors
    }
```

Listing A.15: WebSocket Connection Management (Frontend)

A.9 Interactive Map Visualization

```
// Floor Map Visualization Component
function FloorMap({ floor, positions, rooms }):
    const canvasRef = useRef(null)
    const [dimensions, setDimensions] = useState({ width: 0, height: 0 })
    const [scale, setScale] = useState(1)
    const [pan, setPan] = useState({ x: 0, y: 0 })

    // Filter tags on this floor
    const tagsOnFloor = positions.filter(pos => pos.floor === floor)

    // Setup canvas and draw initial floor plan
    useEffect(() => {
        const canvas = canvasRef.current
        const ctx = canvas.getContext('2d')

        // Set canvas dimensions
        const { width, height } = calculateDimensions(floor)
        canvas.width = width
        canvas.height = height
        setDimensions({ width, height })

        // Draw floor plan background
        drawFloorPlan(ctx, floor)

        // Draw rooms
        for each room in rooms:
            if room.floor === floor:
                drawRoom(ctx, room)
    }, [floor, rooms])

    // Draw tags on the floor when positions update
    useEffect(() => {
        const canvas = canvasRef.current
        const ctx = canvas.getContext('2d')

        // Clear previous tag positions
        ctx.clearRect(0, 0, canvas.width, canvas.height)

        // Redraw floor plan
        drawFloorPlan(ctx, floor)
        drawRooms(ctx, rooms.filter(room => room.floor === floor))

        // Draw each tag
        for each tag in tagsOnFloor:
            drawTag(ctx, tag, scale, pan)
    }, [tagsOnFloor, scale, pan])

    // Handle zoom and pan interactions
    function handleWheel(e):
        if (e.ctrlKey) {
            // Zoom
            const newScale = e.deltaY > 0 ? scale * 0.9 : scale * 1.1
            setScale(newScale)
        } else {
            // Pan
            setPan({
                x: pan.x - e.deltaX,
                y: pan.y - e.deltaY
            })
        }

    return (
        <div className="floor-map-container">
            <canvas
                ref={canvasRef}
                onWheel={handleWheel}
                width={dimensions.width}
                height={dimensions.height}
            />
            <div className="floor-controls">
                <button onClick={() => setScale(scale * 1.1)}>Zoom In</
button>
                <button onClick={() => setScale(scale * 0.9)}>Zoom Out</
button>
                <button onClick={() => setPan({ x: 0, y: 0 })}>Reset View
</button>
            </div>
        </div>
    )

function drawRoom(ctx, room, highlight = false):
    // Set styles based on room properties and occupancy
    if highlight:
        ctx.fillStyle = 'rgba(255, 255, 0, 0.3)'
        ctx.strokeStyle = 'rgba(255, 255, 0, 0.8)'
        ctx.lineWidth = 2
    else if room.occupancy > 0:
        ctx.fillStyle = 'rgba(0, 150, 255, 0.3)'
        ctx.strokeStyle = 'rgba(0, 150, 255, 0.8)'
        ctx.lineWidth = 1.5
    else:
        ctx.fillStyle = 'rgba(200, 200, 200, 0.2)'
        ctx.strokeStyle = 'rgba(150, 150, 150, 0.5)'
        ctx.lineWidth = 1

    // Draw the room polygon
    ctx.beginPath()

    ctx.moveTo(room.coordinates[0].x, room.coordinates[0].y)

    for i from 1 to room.coordinates.length:
        ctx.lineTo(room.coordinates[i].x, room.coordinates[i].y)

    ctx.closePath()
    ctx.fill()
    ctx.stroke()

    // Draw room label
    ctx.fillStyle = 'rgba(0, 0, 0, 0.7)'
    ctx.font = '12px Arial'
    ctx.textAlign = 'center'

    const centerX = room.coordinates.reduce((sum, c) => sum + c.x, 0) /
   room.coordinates.length
    const centerY = room.coordinates.reduce((sum, c) => sum + c.y, 0) /
   room.coordinates.length

    ctx.fillText(room.name, centerX, centerY)

    // If room has occupancy, show count
    if room.occupancy > 0:
        ctx.fillText(`${room.occupancy}`, centerX, centerY + 15)

function drawTag(ctx, tag, scale = 1, pan = { x: 0, y: 0 }):
    // Apply scale and pan transformations
    const x = tag.position.x * scale + pan.x
    const y = tag.position.y * scale + pan.y

    // Draw position dot
    ctx.beginPath()
    ctx.arc(x, y, 6, 0, Math.PI * 2)
    ctx.fillStyle = getStatusColor(tag.status)
    ctx.fill()
    ctx.strokeStyle = 'white'
    ctx.lineWidth = 1
    ctx.stroke()

    // Draw tag label
    ctx.fillStyle = 'black'
    ctx.font = '10px Arial'
    ctx.textAlign = 'center'
    ctx.fillText(tag.name, x, y - 10)

    // If low battery, add indicator
    if tag.battery < 20:
        ctx.fillStyle = 'red'
        ctx.fillText('!', x + 8, y - 8)
```

Listing A.16: Floor Map Visualization Component (Frontend)

A.10 Analytics Data Processing

```
// Analytics Generation Algorithm
function generateAnalytics(timeframe = '24h'):
    // Set time range based on timeframe
    const end_time = getCurrentTimestamp()
    let start_time

    switch(timeframe):
        case '24h':
            start_time = end_time - (24 * 60 * 60)
            break
        case '7d':
            start_time = end_time - (7 * 24 * 60 * 60)
            break
        case '30d':
            start_time = end_time - (30 * 24 * 60 * 60)
            break
        default:
            start_time = end_time - (24 * 60 * 60)  // Default to 24 hours

    // Get timeline events in the specified range
    const timeline_events = getTimelineEvents(start_time, end_time)

    // Group events by tag
    const events_by_tag = {}

    for each event in timeline_events:
        if event.tag_id not in events_by_tag:
            events_by_tag[event.tag_id] = []

        events_by_tag[event.tag_id].push(event)

    // Initialize analytics metrics
    const analytics = {
        total_transitions: timeline_events.length,
        tags_tracked: Object.keys(events_by_tag).length,
        room_occupancy: {},
        room_transitions: {},
        time_in_rooms: {},
        busiest_periods: [],
        system_performance: {
            avg_latency: calculateAverageLatency(),
            uptime: calculateSystemUptime(),
            error_rate: calculateErrorRate()
        }
    }

    // Calculate room occupancy and transitions
    for each event in timeline_events:
        // Increment room transitions count
        const transition_key = `${event.from_room}->${event.to_room}`

        if transition_key not in analytics.room_transitions:
            analytics.room_transitions[transition_key] = 0

        analytics.room_transitions[transition_key]++

        // Add time in room
        if event.from_room not in analytics.time_in_rooms:
            analytics.time_in_rooms[event.from_room] = 0

        analytics.time_in_rooms[event.from_room] += event.
    duration_in_previous

    // Calculate hourly activity for busiest periods
    const hourly_activity = calculateHourlyActivity(timeline_events)
    analytics.busiest_periods = findBusiestPeriods(hourly_activity)

    // Calculate current room occupancy
    const current_positions = getCurrentPositions()

    for each position in current_positions:
        const room = position.room

        if room not in analytics.room_occupancy:
            analytics.room_occupancy[room] = 0

        analytics.room_occupancy[room]++

    return analytics

function calculateHourlyActivity(events):
    const hourly = Array(24).fill(0)

    for each event in events:
        const hour = new Date(event.timestamp * 1000).getHours()
        hourly[hour]++

    return hourly

function findBusiestPeriods(hourly_activity):
    // Clone and sort hourly activity
    const sorted = [...hourly_activity]
        .map((count, hour) => ({ hour, count }))
        .sort((a, b) => b.count - a.count)

    // Return top 3 busiest hours
    return sorted.slice(0, 3).map(item => ({
        hour: item.hour,
        count: item.count,
        percentOfTotal: item.count / hourly_activity.reduce((sum, val) =>
    sum + val, 0)
    }))
```

Listing A.17: Analytics Generation Algorithm

A.11 System Monitoring and Health Check

```
// System Health Check Service
function monitorSystemHealth():
    // Run health checks every 5 minutes
    setInterval(() => {
        const health_status = performHealthChecks()
        storeHealthStatus(health_status)

        // If critical issues, send alerts
        if health_status.critical_issues.length > 0:
            sendAlerts(health_status.critical_issues)
    }, 5 * 60 * 1000)

function performHealthChecks():
    const health_status = {
        timestamp: getCurrentTimestamp(),
        database: checkDatabaseConnection(),
        mqtt: checkMqttConnection(),
        api: checkApiEndpoints(),
        base_stations: checkBaseStationStatus(),
        system_load: getSystemLoad(),
        memory_usage: getMemoryUsage(),
        critical_issues: []
    }

    // Check for offline base stations
    const offline_stations = getOfflineBaseStations()

    if offline_stations.length > 0:
        health_status.critical_issues.push({
            type: 'BASE_STATION_OFFLINE',
            message: `${offline_stations.length} base stations are offline
    `,
            affected_stations: offline_stations
        })

    // Check for database connection issues
    if !health_status.database.connected:
        health_status.critical_issues.push({
            type: 'DATABASE_CONNECTION_FAILED',
            message: 'Failed to connect to database',
            details: health_status.database.error
        })

    // Check for MQTT broker issues
    if !health_status.mqtt.connected:
        health_status.critical_issues.push({
            type: 'MQTT_CONNECTION_FAILED',
            message: 'Failed to connect to MQTT broker',
            details: health_status.mqtt.error
        })

    // Check system load
    if health_status.system_load > 0.9:
        health_status.critical_issues.push({
            type: 'HIGH_SYSTEM_LOAD',
            message: `System load is critically high: ${health_status.
    system_load * 100}%`,
            details: 'Consider scaling or optimizing the system'
        })

    return health_status

function checkBaseStationStatus():
    // Get all base stations
    const base_stations = getAllBaseStations()

    // Check last seen timestamp
    const now = getCurrentTimestamp()
    const timeout = 5 * 60  // 5 minutes timeout

    const status = {
        total: base_stations.length,
        online: 0,
        offline: 0,
        details: []
    }

    for each station in base_stations:
        const time_since_last_seen = now - station.last_seen
        const online = time_since_last_seen < timeout

        status.details.push({
            id: station.id,
            mac_address: station.mac_address,
            name: station.name,
            online: online,
            last_seen: station.last_seen,
            time_since_last_seen: time_since_last_seen
        })

        if online:
            status.online++
        else:
            status.offline++

    return status

function sendAlerts(issues):
    for each issue in issues:
        // Log issue
        console.error(`ALERT: ${issue.type} - ${issue.message}`)

        // Send to notification system
        notificationService.send({
            level: 'critical',
            title: issue.type,
            message: issue.message,
            details: issue.details,
            timestamp: getCurrentTimestamp()
        })

        // If configured, send email/SMS alerts
        if ENABLE_EMAIL_ALERTS:
            sendEmailAlert(issue)

        if ENABLE_SMS_ALERTS && issue.type.startsWith('BASE_STATION'):
            sendSmsAlert(issue)
```

Listing A.18: System Health Check Service