POLITECNICO DI TORINO

MSc's Degree in ICT for smart societies

MSc's Degree Thesis

Indoor BLE localization system for building facilities improvement

Supervisors

Prof. Fabio DOVIS

Prof. José-Fernán MARTÍNEZ-ORTEGA

Candidate

Riccardo NICOLICCHIA

July 2023

Abstract

Over the past few years, the concept of IoT has become more and more widespread. Many different kinds of systems can be categorized under this topic. Each of these systems stands out thanks to the variety and heterogeneity of sensors and actuators that make it possible to monitor and analyze systems in real-world settings. The support of this technology could be crucial for enhancing, say, some facets of building management. By incorporating the concept into so-called *Iot4FM*, for example, a company building may use it to improve staff productivity. The *Facility Management* methodology makes sure that all services and goods are maintained and even enhanced. Modern technology, particularly wireless ones, enables this. Significant aspects, such as the environment condition or occupancy, could be measured, for example. A substantial amount of data is generated as a result of several sensors constantly collecting data —in this project's case, the gateway collecting BLE (Bluetooth Low Energy) tags signal. This brings up data analysis, which is yet another important consideration. It helps you understand how to leverage the data and existing technologies to their fullest potential in order to provide high-quality services. The most effective way to accomplish the overall goal of enhancing the administration and use of office spaces is through the combination of new technologies and data analysis. This master's thesis attempts to use a localization infrastructure that is already in place by employing an innovative approach. In order to provide a better, more dynamic localization system, the project offers a full-stack solution that uses the BLE tags and gateways supplied by the company. The application generates the raw localization data, displays it using a straightforward user interface, and further elaborates it. This offers better control over the building's current occupancy as well as fast indications on potentially essential issues like room reservations or long-term hints, like in building security.

Acknowledgements

This work marks the completion of one of the final steps of an academic career that years ago I would not have imagined taking, after all the past difficulties and downfalls. Despite everything I'm here talking about it, and if all this was possible I have a few people to thank. Mine is an international route, and each one deserves a particular thanks in a different language, to better appreciate the gratitude I want to show.

A Miguel Ángel López Peña y Juan Sebastián Ochoa Zambrano por su paciencia, disponibilidad y profesionalidad que hicieron posible el desarrollo de este proyecto.

A todos los amigos de Madrid, españoles, internacionales y italianos, que me hicieron enamorar de la ciudad que se está convirtiendo cada vez más en mi segunda casa.

Agli amici di Torino che nonostante il mio espatrio mi vogliono sempre bene, che rendono la fredda e nordica Torino un posto meraviglioso dove ritornare e sentirmi a casa.

A Vincenzo e Gianluca, due amici che tutti sperano prima o poi di incontrare nella propria vita. Nonostante la distanza che ci separa rimaniamo sempre quelli di Villa Claretta, cresciuti come uomini a pane (duro della mensa ...) e sketch di Aldo Giovanni e Giacomo.

A Valentina la persona che in questi anni, più di chiunque altro, è riuscita, anche con piccoli gesti, a tirare fuori il meglio di me, che mi ha sempre supportato e mi ha aiutato a non mollare nonostante tutti gli ostacoli che mi si sono posti davanti.

Alla mia famiglia, che mi ha appogiato sempre, in ogni scelta e che nonostante la distanza riesce ad essermi sempre vicina.

A mio padre, con la speranza che da lassù vedendo i miei traguardi e la persona che sono diventato, continui a essere fiero di me. La dedica più grande, per i traguardi raggiunti e che raggiungerò andrà sempre a te...

Table of Contents

List of Tables v

List of Figures vi

Acronyms ix

1 Introduction 1
- 1.1 Conceptual framework . . . 1
  - 1.1.1 IoT & indoor tracking . . . 1
  - 1.1.2 Indoor tracking & Facility Management . . . 2
- 1.2 Data streaming . . . 3
- 1.3 Data analytic . . . 4
  - 1.3.1 Exploratory Data Analysis . . . 4
- 1.4 Motivation and justification . . . 4
- 1.5 Objectives . . . 4

2 State of the art 6
- 2.1 System hardware . . . 6
  - 2.1.1 BLE gateway: iGS01S . . . 6
  - 2.1.2 BLE lanyard tag . . . 7
- 2.2 System backend . . . 8
  - 2.2.1 Python and libraries used . . . 8
  - 2.2.2 Docker and Docker compose . . . 11
  - 2.2.3 Apache Kafka . . . 11
- 2.3 User interface: Frontend and Backend . . . 13
  - 2.3.1 Backend core: Express.js, Node.js & Javascript . . . 13
  - 2.3.2 Leaflet.js . . . 14
  - 2.3.3 KafkaJS . . . 14

3 Development 15
- 3.1 Microservice architecture . . . 16

3.1.1 Development methodology: CI/CD . . . 17
3.2 System configuration . . . 19
3.2.1 Gateway setting . . . 19
3.2.2 Tags settings . . . 22
3.2.3 Apache Kafka broker . . . 24
3.3 Services developed . . . 25
3.3.1 Deployment . . . 25
3.3.2 HTTP server for gateways . . . 26
3.3.3 Localization engine . . . 29
3.3.4 Occupation calculator engine . . . 53
3.3.5 Real time map . . . 57
3.4 Exploratory data analysis . . . 62

4 Results and conclusions 72
4.1 Results . . . 72
4.1.1 Localization engine results . . . 72
4.1.2 Occupation engine results . . . 75
4.1.3 User Interface . . . 75
4.2 Conclusions . . . 76
4.2.1 Future works . . . 78

A Kafka broker settings 80

B Localization services settings 83

Bibliography 84

List of Tables

2.1 iGS01S gateway fields description 7

3.1 Beacon's transmission power description 23
3.2 Kafka topics characterization 25
3.3 Gateway reference information 63
3.4 Kalman filter use cases 68

List of Figures

2.1 iGS01S gateway architecture 7
2.2 Docker stack 12
2.3 Apache Kafka architecture 12

3.1 System overview 15
3.2 Monolithic architecture vs microservices architecture 17
3.3 CI/CD GitLab tool 18
3.4 iGS01S gateway WiFi settings 19
3.5 iGS01S gateway network settings 19
3.6 iGS01S gateway HTTP configurations 20
3.7 iGS01S gateway RSSI signal configuration 20
3.8 iGS01S gateway timing settings 21
3.9 Tag setting app general view 22
3.10 Setting panel single tag 22
3.11 Advertising interval and signal stability 24
3.12 Topics dashboard overview 25
3.13 Base stations map 27
3.14 Localization engine operation flow 30
3.15 Dataframe of tags information 31
3.16 Dataframe of tags current status 32
3.17 Dataframe for Kalman filter calculation support 32
3.18 Dataframe for current RSSI tracking 33
3.19 Dataframe for last valid RSSI fingerprint tracking 33
3.20 RSSI Tag packet structure 35
3.21 One station trilateration 41
3.22 Two stations trilateration 41
3.23 Geometrical representation of trilateration 41
3.24 Localization engine JSON packet structure 45
3.25 Reference systems of developement maps 55
3.26 Architectural overview of real-time map 58
3.27 Process diagram of real-time map 59

3.28 Views on real-time map 61
3.29 EDA context 62
3.30 Signal evolution of ap_194 64
3.31 Signal evolution of ap_196 64
3.32 Signal evolution of ap_199 65
3.33 Signal evolution of ap_206 65
3.34 Signal evolution with Q=0.01 and R=5 68
3.35 Signal evolution with Q=0.05 and R=5 69
3.36 Signal evolution with Q=0.5 and R=5 69
3.37 Signal evolution with Q=0.1 and R=0.5 70
3.38 Signal evolution with Q=0.1 and R=1 70
3.39 Signal evolution with Q=0.1 and R=10 71
3.40 Signal evolution with Q=0.1 and R=5 71

4.1 One day position tracking 73
4.2 Error overview of the trace 73
4.3 Temporal improvements of position 74
4.4 Single tag position track 74
4.5 Layer control panel 75
4.6 Marker information showed 75
4.7 Only markers view 76
4.8 Only occupation view 76

Acronyms

AP

Access **P**oint

API

Application **P**rogramming **I**nterface

BLE

Bluetooth **L**ow **E**nergy

CI/CD

Continous **I**ntegration and **C**ontinous **D**eployment

CSV

Comma-**S**eparated **V**alues

CSS

Cascading **S**tyle **S**heets

dBm

deci**B**el **m**illiwatt

DHCP

Dynamic **H**ost **C**onfiguration **P**rotocol

DNS

Domain **N**ame **S**ystem

EDA

Exploratory **D**ata **A**nalysis

EJS

Embedded **J**avaScript **T**emplating

HTML

Hyper**T**ext **M**arkup **L**anguage

HTTP

Hyper**T**ext **T**ransfer **P**rotocol

HTTPS

Hyper**T**ext **T**ransfer **P**rotocol **S**ecure

ICT

Information and **C**ommunication **T**echnology

IoT

Internet **o**f **T**hings

IoT4FM

Internet **o**f **T**hings **F**or **F**acilities **M**anagement

IP

Internet **P**rotocol

JSON

Java**S**cript **O**bject **N**otation

MSE

Mean **S**quare **E**rror

OS

Operating **S**ystem

Paas

Platform **a**s **a** **S**ervice

QoS

Quality **o**f **S**ervice

REST

REpresentational **S**tate **T**ransfer

RSSI

Received **S**ignal **S**trength **I**ndicator

SQL

Structured **Q**uery **L**anguage

SSE

Server-**S**ent **E**vent

TCP

Transmission **C**ontrol **P**rotocol

UI

User **I**nterface

URL

Uniform **R**esource **L**ocator

Chapter 1

Introduction

For many years, the IoT (**I**nternet **o**f **T**hings) has been collecting more and more attention from the ICT (**I**nformation and **C**ommunication **T**echnology) sector. Although the market for this technology is currently enormous, the growth projections indicate an exponential upward trajectory[1]. According to its definition, the IoT environment consists of a wide range of smart devices that talk to each other.IoT offers a wide range of applications, including industrial automation, healthcare, environmental monitoring ... These applications are enabled by the momentum generated by artificial intelligence, data analytics, and new technologies (particularly wireless ones). In principle, the devices are used to track a range of factors, from the simplest to the most complex. If the information was kept on hand, it could be examined to look for patterns and trends. Even a single device with a reasonable sample rate can generate data that can be used to understand the current condition of the environment and to determine how to improve it. This data can inspire various actions, which can be translated by the users themselves or automatically by actuators. These broad assumptions lead us to the project's objective, which is to use wireless enabled devices to track employee activity in the workplace and provide management guidance for the building's spaces. The entire development was carried out in the central building of the company SATEC - Sistemas Avanzados de Tecnología S.A., which provided the facilities, the existing hardware structure and the company server and repository to support this project.

1.1 Conceptual framework

1.1.1 IoT & indoor tracking

The IoT technology [2][3][4] is a network of physical objects that contain embedded technology that allows them to communicate, sense, or interact with their internal

states or the external environment. Sensors can use communication channels [5] to send collected data to storage, interact with one another, or possibly with other actuators. As a result, they have generally embedded a minimum level of intelligence. Standards for IoT technology include ISO/IEC 21823-1:2019 [6] for systems interoperability, ISO/IEC TR 30166:2020[7] for industrial IoT ISO/IEC WD 30162 [8] for device compatibility within industrial IoT systems, and ISO/IEC AWI 30165 [9] for the real time frameworks.

One of the uses for IoT technology is indoor tracking, which gives businesses the ability to monitor the movement of workers, and other assets inside a building. Indoor tracking systems can give real-time information on the position of people and can be combined with other information like temperature, humidity, and air quality by using sensors and other tracking devices. These systems can interface with other equipment and systems in a building thanks to IoT technology, giving a more thorough view of facility operations. Organizations may increase the efficacy and efficiency of their operations by connecting indoor tracking with other IoT-enabled equipment and gaining even more insights into facility operations. These enhancements could be found throughout the scope of a smart building, such as increased energy efficiency, better space utilization, and enhanced user experience [10]. In the manufacturing application, the scope might be shifted as well. In this context, integrating indoor tracking with other IoT devices might assist increase safety, decrease downtime, and increase production process efficiency [11].

1.1.2 Indoor tracking & Facility Management

Facilities management and indoor tracking are two distinct domains that can be linked to improve the efficiency and efficacy of facility management activities. Facility Management [12][13] is a methodology for managing buildings facilities[1] and services. This methodology ensures that people working in a building are functional, comfortable, and safe. There are standards for facility management such as ISO 41001:2018[14], ISO 41011:2017[15], ISO 41012:2017 [16], ISO/TR 41013:2017[17], ISO/CD 41014 [18] and ISO/AWI 41015 [19]. Hence, integrating indoor tracking with facility management can offer a number of advantages. Facility managers can make the best use of available space within a facility by using indoor tracking data. They can identify inefficient locations and take action to better utilize space by analyzing how people and assets move throughout a facility. Indoor tracking can be utilized to boost facility security and safety. For instance, monitoring the movements of staff members and guests can assist uncover potential safety risks, and monitoring the movements of equipment and assets can help reduce theft and

[1]Facilities: the buildings, equipment, and services provided for a particular purpose

maximize asset use. Indoor tracking data can be utilized to determine trends of equipment usage allowing facility managers to more effectively schedule maintenance actions. Administrators can prevent breakdowns by proactively identifying when maintenance is necessary on their equipment by tracking how it is being used [20][21][22]. Supervisors can spot areas of congestion and take action to enhance traffic flow by monitoring how people move around a facility. Better user experiences may result from this, especially in settings like airports, hospitals, and shopping malls.

1.2 Data streaming

The term *data streaming* [23][24] describes the method of continuously processing and analyzing data as it comes in real-time from many sources. In the data stream scenario, input comes in very quickly and can only be temporarily stored in memory. The data must be processed by algorithms in a single or multiple passes, taking significantly less time or less space than the input size. A different approach to thinking about algorithms that adhere to these limitations on space, time, and the number of passes has recently been developed. Some of the techniques rely on pseudo-random computations and metric embeddings. In contrast to batch processing, which processes data in huge quantities at once, data streaming processes data in small, incremental bits or data chunks. With real-time data processing and receiving, data streaming enables quick detection and reaction to dynamic environment, trends, or abnormalities. In industries like finance, healthcare, and transportation, where the prompt identification and response to events can have a substantial impact on business outcomes, this real-time processing and analysis of data is frequently crucial. Data collection, data transformation, and data analysis are just a few of the procedures involved with data streaming. Applications for this set of circumstances, then, include processing enormous amounts of data in general, text message stream mining, and IP network traffic analysis. These data is gathered from a variety of sources, including sensors, programs, and databases. The data is then processed in real-time after being translated into a format that allows for processing and analysis. To enable data streaming, sophisticated software tools and platforms built to handle the constant flow of data are employed. Technologies like Apache Kafka [25][26][27], Apache Flink, or Apache Spark Streaming are some of the most used. Data streaming is becoming more and more crucial as businesses try to make sense of the enormous amounts of data produced by IoT [28][29] devices and other sources. Real-time processing and analysis of streaming data may help organizations make better decisions, identify issues earlier, and generate greater revenue.

1.3 Data analytic

1.3.1 Exploratory Data Analysis

EDA (**E**xploratory **D**ata **A**nalysis)[30][31][32] is a critical phase in any Data Analysis or Data Science project. It is the process of analyzing a dataset from as many different perspectives as possible in order to uncover patterns and abnormalities (outliers) and generate hypotheses based on our understanding of the dataset. The only constraints on such an analysis are those imposed by time constraints and the data analyst's creativity. In fact, as the acronym EDA implies, one is free to choose any process to analyze the data, and the key goals are to look at the data and think about it from various perspectives. This is due to the fact that EDA is not supported by a statistical model that includes a mathematical equation for such an impact. In order to better comprehend the data, graphical visualization is typically the early step. In order to grasp the trend and relationships between all the variables that are related to enhance the office environment, different visual representations of the data as well as specific numerical statistics will be shown in this project.

1.4 Motivation and justification

The management of a company office and the improvement of working conditions are two areas where the integration of IoT technology into Facility Management methodologies offers significant benefits. It is possible to get a sense of the current state of the office by continuously monitoring the environment. In particular, you get three benefits: you have a clear view of the state of occupation of the building, you can avoid unoptimized use of office spaces and you earn in terms of security planning in the company.

1.5 Objectives

The objectives of this thesis are to improve SATEC - Sistemas Avanzados de Tecnología S.A. company office's location solution, developing a full-stack solution (from data collection to data visualization) to improve the office environment in terms of occupancy and space management. The following milestones were developed to achieve these goals:

- Demonstrate the reliability of the data sources. Ensure that the data obtained are reasonable and adequately calibrated is critical in localization, therefore the pre processing stage has an impact on the subsequent steps. In addition

to this in the scope of this project is important to ensure a good timing management of all the measurements;

- Make a backend architecture to easily sink sensor data and make it accessible. The quality of the results need to allow you to have data almost in real time. In the specific case of use of location, the timing management of the data is crucial;
- Make the system modular so that it is feasible to make changes to the various parts without affecting how the system as a whole functions.
- Parameter the system, accurately to allow access to different QoS (**Q**uality **o**f **S**ervices);
- Create an algorithm providing a reasonable accuracy, that can instantly manage the calculation and publication of calculated positions, to make them available in a short time.
- Make the final results easier to interpret by visualizing them, and make it available in almost real-time.

Chapter 2

State of the art

2.1 System hardware

2.1.1 BLE gateway: iGS01S

The BLE/WiFi Gateway iGS01S[33] is utilized as a sink for all of the RSSI signals from all of the tags. It serves as a bridge to connect nearby WiFi-enabled or BLE devices, sensors, or beacons to the internet. One can configure the internet connection to a general cloud server, such as TCP (**T**ransmission **C**ontrol **P**rotocol), HTTP(S), or MQTT(Message Queue Telemetry Transport), through a web UI interface. The general behavior of an iGS01S-enabled system is depicted in Fig. 2.1. The iGS01S gateway, so, can read beacons, and iBeacon compatibility is crucial for this project. This positioning system tool will be discussed in depth in the section that follows. This gateway offers Station mode and AP (**A**ccess **P**oint) mode for WiFi connections. The device functions as a basic AP that supports DHCP (**D**ynamic **H**ost **C**onfiguration **P**rotocol) when in AP mode. This mode's primary function is configuration. In Station mode, a client device repeatedly attempts to connect to the WiFi AP that you specified. Once you've connected to the AP, the gateway can connect your BLE devices to a local TCP server for management. BLE is often in listening mode. It gathers the messages that other BLE devices advertise. The user-configured cloud server receives these messages once they have been transferred over WiFi. The packet will have the following format, and the fields are described in Table 2.1:

$$\$"report\_type","tag\_id","gateway\_id","RSSI","raw\_data","timestamp"$$

report_type	GPRP: general purpose report. SRRP: active scan response report
tag_id	MAC address or ID of tag/beacon
gateway_id	MAC address of gateway's BLE
RSSI	RSSI of tag/beacon
raw_data	Raw packet received by the gateway
timestamp	Unix epoch timestamp when NTP is enabled

Table 2.1: iGS01S gateway fields description

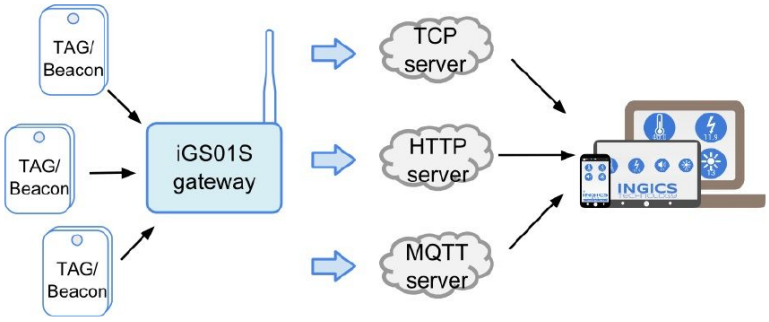


Figure 2.1: iGS01S gateway architecture

2.1.2 BLE lanyard tag

The system is supported by the use lanyard tags named kontakt.io KHWPO400F001 Lanyard Tag [34]. These tags transmit Bluetooth signals continuously for tracking reasons. The device is set up for use right out of the box. As soon as the device is assembled and programmed, it is ready to work. You can configure the transmission power, the advertising interval, and the type of packet using the kontakt.io settings app. You can use the kontakt.io format, Eddystone, or iBeacon for packets. The last one is the one that is used and will therefore get more attention. The iBeacon [35] is a technology that allows applications to be aware of their position using BLE . This gives to the device the ability to tell when it has arrived or departed the region and estimate its distance from a beacon. It's crucial, so, to think about how beacon signals are recognized and used to assess accuracy when working to guarantee a positive user experience. RSSI is frequently used to detect a beacon's signal and to assess both the accuracy of the beacon's estimated distance and the distance to the beacon itself. The stronger the signal, the more confident you can be about the beacon's proximity. The weaker the signal, the harder to predict you

are of the beacon's presence. Bluetooth Low Energy is therefore used by iBeacon devices to transmit signals. Since BLE operates at 2.4 GHz, it can be attenuated by a variety of physical objects, including walls, doors, and other buildings. Water can interfere with 2.4 GHz transmissions, hence the human body will likewise interfere with signals. This is important to know because weakening or attenuating the Bluetooth signal impacts the signal strength received. As results it affect the deployment of the application and the performance. Calibration in your deployment environment is essential for delivering the optimal user experience. A calibration step should be carried out as each beacon is placed. The estimation model needs to be calibrated at a distance of 1 meter from the beacon. To sum up, it is a very powerful resource that should be treated carefully. Finally, this technology also has the significant benefit of allowing devices to run on solely coin-cell batteries for up to a month, this make this technology really suitable for IoT applications that need good performances in terms of energy savings.

2.2 System backend

2.2.1 Python and libraries used

Python [36] is the programming language utilized to develop the application for this thesis project. It is a general-purpose, high-level, interpreted programming language. This language contains a large number of compatible packages and libraries, making it appropriate for a wide range of applications. The libraries used in this project are described in detail in the following sections.

NumPy and SciPy

Two related Python libraries for numerical computing and scientific computing, respectively, are NumPy [37] and SciPy [38]. While SciPy offers sophisticated mathematical functions and methods for scientific computing, NumPy supports massive, multi-dimensional arrays and matrices as well as a variety of mathematical functions to operate on these arrays. Since its creation, NumPy has grown into a popular open-source library for scientific computing and data analysis. It is extensively used in disciplines including data science, engineering, and physics. The ndarray is the main data structure of NumPy, and it offers quick and effective numerical operations on arrays of any dimension, which are essential in EDA (**E**xploratory **D**ata **A**nalysis) or data management for input into a mathematical model. In addition, NumPy offers a number of mathematical operations, including trigonometric, statistical, and linear algebra functions. On the other hand, SciPy was developed as a NumPy extension. It offers sophisticated mathematical functions and algorithms for scientific computing, including as statistics, signal processing,

optimization, and interpolation functions. Moreover, it has modules for machine learning, image processing, and working with sparse matrices. For data analysis and modeling, SciPy is widely utilized in the scientific and engineering disciplines. For scientific computing and data analysis in Python, NumPy and SciPy provide a robust ecosystem.

Pandas

Pandas [39][40] is a Python library for analyzing and manipulating data. For handling and processing structured data, such as tables, time series, and matrices, it offers data structures and operations. Because of its versatility and usability, Pandas—which is developed on top of the NumPy library—is a popular tool in data science and analytics. *DataFrame* and *Series* are the two main data structures in Pandas.The data frame is a two-dimensional object with rows and columns, but the series is a one-dimensional object with an array-like structure that may hold any form of data. Moreover, Pandas has many tools for interacting with these structures, such as grouping, indexing, merging, and reshaping. Data formats that Pandas can handle include CSV (**C**omma-**S**eparated **V**alues) files, Excel, SQL (**S**tructured **Q**uery **L**anguage) databases, and JSON (**J**ava**S**cript **O**bject **N**otation) files. Additionally, it offers resources for conducting statistical analysis and dealing with missing data.

Matplotlib & Seaborn

Two popular open-source Python libraries for data visualization are Matplotlib [41] and Seaborn [42]. These libraries are well-liked among data scientists, engineers, and academics because they offer a variety of options for producing static and interactive visualizations of data. Matplotlib is a low-level Python toolkit for making visualizations. Line plots, scatter plots, bar plots, histograms, and more types of plots are available. Users of Matplotlib can easily modify the color schemes, axes labels, legends, and other aspects of their plots. Matplotlib is a flexible tool for data visualization because it also allows for the creation of animations and 3D visuals. On the other hand, Seaborn, which is developed on top of Matplotlib, offers a more advanced interface for producing statistical visualizations. A variety of visualization options are offered by Seaborn, including heat maps, violin plots, and box plots. Pandas data structures and Seaborn were created to complement one other flawlessly. Both Matplotlib and Seaborn offer a plethora of documentation and documentation, making them both potent data visualization tools.

Falcon & bjoern

Bjoern [43] is a fast and lightweight Python web server built in C that can handle a high volume of requests with low overhead. It offers a straightforward, minimalist interface for serving web pages and applications and is built on top of the libev[1] event loop.
Falcon [44] is a high-performance Python web framework intended for creating API (**A**pplication **P**rogramming **I**nterfaces). It offers a variety of tools for creating RESTful (**RE**presentational **S**tate **T**ransfer) APIs and microservices and is designed for speed and efficiency. Falcon and Bjoern can process a lot of requests quickly and effectively because they are built to be quick and effective. Both are straightforward to understand and alter due to their small code bases and minimal dependencies. Bjoern offers an easy-to-use interface with support for fundamental HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) methods and request handling for serving web pages and apps. Falcon offers a number of tools for developing RESTful APIs and microservices and is built around the REST architectural style. In addition, the middleware architecture offered by Falcon makes it simple to integrate bespoke functionality into API endpoints. Together, Bjoern and Falcon offer a strong basis for creating high-performance APIs that can process numerous requests quickly and efficiently.

Confluent's Python Client for Apache Kafka

The Confluent Kafka Python library [45] is a client library that provides a simple and efficient way to connect with Kafka clusters using the Python programming language. It offers a high-level interface for sending and receiving messages from Kafka topics and is built on top of the librdkafka[2] C library. The library offers simple Producer and Consumer APIs so that programmers can publish and subscribe to Kafka topics. Avro, JSON, and plain text are just a few of the message formats that are supported by the library. The library comprises a high-level Consumer API that offers message delivery assurances, automatic offset management, and partition rebalancing. For sending and receiving messages, the library supports both synchronous and asynchronous Interfaces. A variety of configuration options and tweaking settings are offered by the library to help users maximize performance and scalability.

[1] libev reference

[2] librdkafka: the Apache Kafka C/C++ client library

Shapely

A toolkit for manipulating and analyzing planar geometric objects is the Shapely Python library. It is constructed on top of the computational geometry techniques provided by the GEOS (Geometry Engine - Open Source) package[3]. Shapely offers classes for expressing geometric objects like Points, Lines, Polygons, and MultiPolygons as well as operations for analyzing and manipulating these objects, including union, intersection, buffer, centroid, area, length, and many other operations. The library offers a straightforward and understandable interface for designing and modifying geometric objects.

2.2.2 Docker and Docker compose

Docker [46] is a set of Paas (**P**latform **a**s **a** **S**ervice) solutions that provide software in containers via OS-level (**O**perating **S**ystem-level) virtualization. A container is a standard unit of software that contains the code to be run as well as all of its dependencies, allowing the program to operate rapidly on various types of machines or operating systems while also providing considerable scalability. To run a Docker container, you have to create an image, which is a lightweight, standalone, executable package of software that includes everything needed to run the application: code, runtime, system tools, system libraries, and settings. Because containers isolate software from its environment, containerized software will always run the same, regardless of the infrastructure. They actually run directly in the host's kernel as if they were another application, but in isolation from the rest (Fig 2.2).

Docker Compose [47] is a tool provided by the Docker framework that assists in the definition and sharing of multi-container applications. Compose allows you to declare all of the services of a composite application in a single YAML[4] file and start it with a single command.

2.2.3 Apache Kafka

Apache Kafka [48][49] is a popular open-source distributed streaming platform that was created by the Apache Software Foundation. Apache Kafka is fundamentally a distributed messaging system that enables numerous producers to deliver messages to numerous consumers in a fault-tolerant and scalable manner. This is accomplished by employing a publish-subscribe approach, in which message publishers send messages to Kafka topics, and message subscribers receive the messages. As

[3]GEOS reference

[4]YAML format

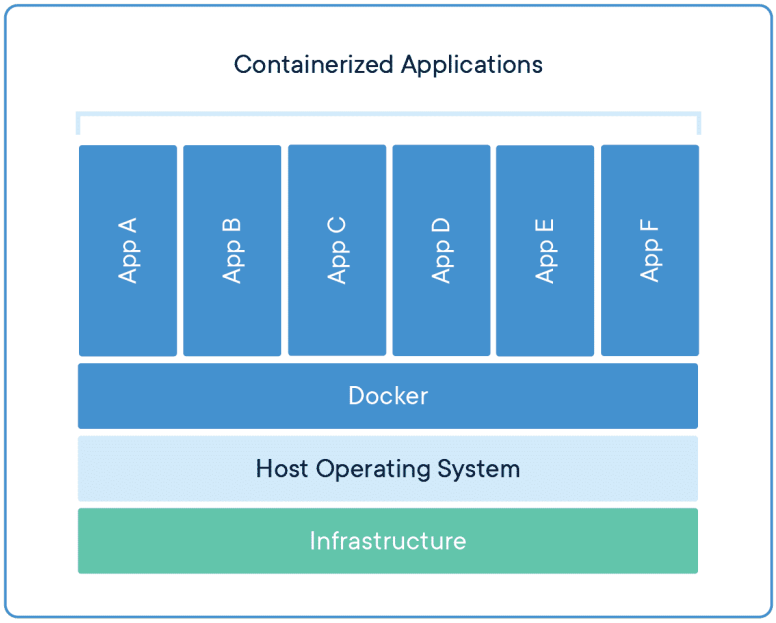


Figure 2.2: Docker stack

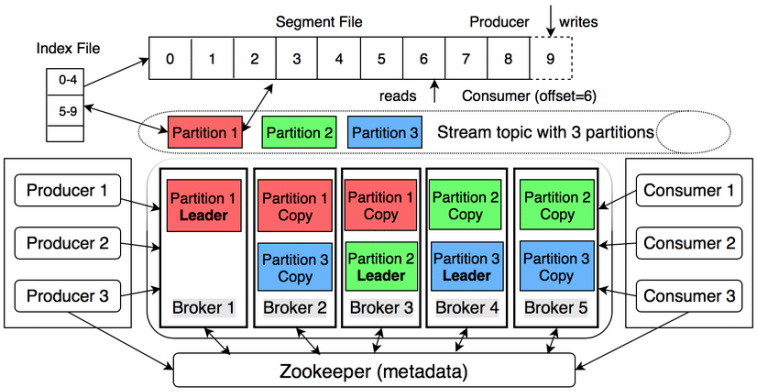


Figure 2.3: Apache Kafka architecture

indicated in Fig.2.3, the structure is very solid, with each topic representing a highly precise form of data stream; it functions similarly to a queue, receiving and delivering messages. Each topic can have one or more partitions; you must provide such number while creating the topic. A topic is divided into partitions. You can specify a key when creating a message for a certain topic to allocate it to the same partition. By default, the producers will deliver a message in a round-robin fashion if you do not provide a key. A message will be delivered to each partition (even if they are sent by the same producer). Due of this, ordered delivery is not

guaranteed at the partition level, therefore you have to include a key to messages if you want to send a message to the same partition every time. Each communication is going to be kept in the broker disk and given an offset (unique identifier). At the partition level, this offset is distinct; every partition has a different offset. As a result, Kafka is a reliable solution for managing enormous amounts of data, as it also provides strong durability and data retention assurances. Unlike a messaging system, which deletes the message after it has been read. Scalability is one of Apache Kafka's prominent characteristics. Kafka can readily scale horizontally by adding more brokers (nodes) to the cluster because it is made to operate as a distributed system. Because of this, it is able to manage vast volumes of data and serve an increasing number of users without suffering a significant performance hit.

Moreover, Apache Kafka has a variety of integration options, making it simple to integrate with other programs and systems. There are several client libraries included for Python, and it can also be coupled with a number of other big data technologies available in the Apache stack. ts ability to support multiple producers and consumers, combined with its durability and data retention guarantees, make it a popular choice for building real-time streaming applications.

2.3 User interface: Frontend and Backend

2.3.1 Backend core: Express.js, Node.js & Javascript

Express.js [50] is a popular open-source web application framework for Node.js [51]. It is built on top of Node.js, a JavaScript runtime environment for servers that enables programmers to create scalable network applications. Conversely, JavaScript [52] is a high-level, dynamic, interpreted programming language that is frequently employed for creating websites among other things.

With its event-driven, non-blocking I/O (Input/Output) approach, Node.js is quick and effective. It executes JavaScript code on the server-side using Google's V8 JavaScript engine, which is also utilized by the Chrome web browser. Using Node.js, you can create real-time, high-performance apps that can manage numerous concurrent connections. Built on top of Node.js, Express.js offers a variety of tools for creating online apps and APIs. Express.js provides a simple and intuitive API for creating routes and controlling HTTP requests. Middleware, or methods, are supported and enable you to modify request and response objects as well as add new functionality to web applications and APIs. EJS(**E**mbedded **J**avaScript **T**emplating)[53] is one of the template engines provided by Express.js that can be used to produce dynamic HTML (**H**yper**T**ext **M**arkup **L**anguage) content. It also provides a robust framework for handling errors with built-in error handlers and the choice to add special error-handling middleware. JavaScript is a powerful and versatile programming language that is used for web development. It is the only

programming language that can be executed natively in web browsers, and has become the de facto language for building client-side web applications. In summary, Express.js, Node.js, and JavaScript provide a powerful and versatile stack for building web applications and APIs. Node.js provides a fast and efficient runtime environment for executing JavaScript code on the server-side, while Express.js provides a feature-rich web application framework for building APIs and web applications. JavaScript provides a powerful and versatile programming language that is used for client-side and server-side web development, among other things.

2.3.2 Leaflet.js

Leaflet.js [54] is a well-known open-source JavaScript library used to create interactive maps and web-based geographic applications. It offers a simple, modular, and adaptable APIs for making maps and adding interactive features like layers, markers, popups, and other interactive elements to them. It supports a broad range of basemaps and tile suppliers, including OpenStreetMap, Google Maps, as well as custom maps, and renders the maps using web technologies including HTML, CSS (**C**ascading **S**tyle **S**heets), and SVG (Scalable Vector Graphics). Zooming, panning, dragging, and clicking are just a few of the interactive features that Leaflet.js offers, allowing users to explore and interact with maps in a broad range of ways. Using CSS and JavaScript, developers can alter the look and behavior of maps and markers. With a focus on speed and effectiveness, Leaflet.js uses methods like vector rendering and tile caching to enhance map rendering and reduce network queries.

2.3.3 KafkaJS

KafkaJS[55] is an open-source JavaScript client library for Apache Kafka that lets developers to create Kafka consumers and producers in Node.js and browser-based applications. It offers an effective and simple API for working with Kafka brokers, getting messages from topics, sending messages to topics, and maintaining consumer groups. You can use it to interact with Kafka clusters that are set either locally or in the cloud. With npm[5], KafkaJS may be quickly added to Node.js applications. In order to achieve low latency and high throughput, KafkaJS uses Node.js streams in addition to additional performance enhancements. Developers can quickly handle and recover from common Kafka issues including network timeouts, partition rebalancing, and producer failures thanks to KafkaJS's comprehensive error handling capabilities. KafkaJS supports both secure and non-secure connections and works with a variety of Kafka brokers and versions.

[5]npm:JavaScript Package Manager

Chapter 3

Development

In this chapter, the system as a whole and each of its parts will be thoroughly discussed. The development process and design decisions will also be covered, in addition to the component structure. The overall layout of the system is depicted in Fig. 3.1. You can see the path taken by the BLE signal as it leaves the BLE tag

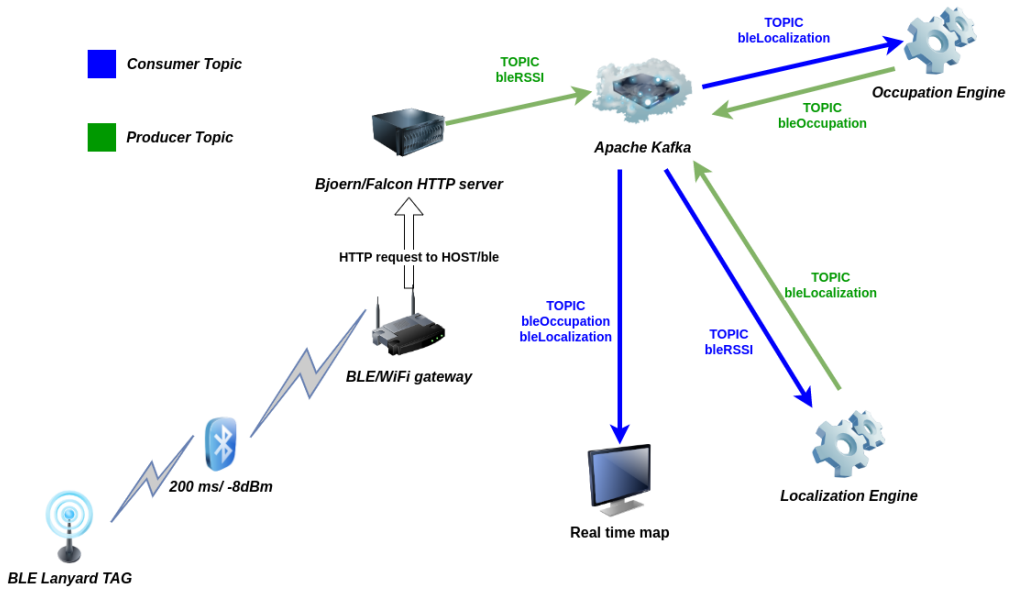


Figure 3.1: System overview

and arrives to the visualization. The Apache Kafka broker serves as the system's brain. For many different types of information necessary for the proper system's functioning, it serves as both a source and a sink. The HTTP server is a further vital component. The management of the packets coming in from all the BLE gateways depends on this. The combination of Bjoern and Falcon enables the very

quick and trustworthy management of a large number of requests, as was discussed in Section 2.2.1. This feature is crucial because each tag will communicate with the nearby gateways every 100 milliseconds since its communication is in advertise mode. With an average latency of less than 1 second, gateways will POST all the data they have gathered from the other side. The massive amount of data produced every second explains the use of the Apache Kafka platform, which, as stated in Section 2.2.3, is ideal for this use case. The localization engine and the occupation engine are the two functional components of the system. They are Docker-deployed microservices developed in Python. They enable the system to achieve its goal, namely to calculate the person's position in the office and track the use of various office spaces. Real-time visualization mapping is the final service. The system's frontend is quick and responsive thanks to Express.js and Docker. Results are almost instantaneously visualized on the map as soon as the other services push them for the topic of interest. In order to summarize the entire project, it offers an end-to-end solution for analyzing, filtering, and visualizing the position calculated by localization engine, using the BLE tag signals.

3.1 Microservice architecture

Microservice architecture [56][57][58][59] is a design approach that emphasizes building software applications as a group of tiny, independent services. This architecture separates the application into discrete services, each of which is in charge of manage a specific feature, which is opposed to conventional monolithic architectures, where the entire application is built as a single, tightly connected unit.

Each service may be built, deployed, and scaled separately under a microservice architecture and operates as a distinct entity. This degree of freedom enables parallel development by allowing development teams to work on various services at once. Developers also have the freedom to select the many programming languages, frameworks, and technologies that best meet the needs of each service.

Scalability is one of the microservice architecture's key benefits. Services can be scaled separately depending on the precise demand they encounter because they are separated from one another. It is particularly helpful for managing fluctuating loads, seasonal spikes, or rapid user base growth. Better fault isolation is an additional benefit. A flaw or failure in one area of the system might spread and affect the entire application in a monolithic design. Failures, however, are contained to the single service where they occur in a microservice design. Furthermore, independent service deployment enables quick and frequent updates, allowing businesses to react swiftly to shifting business needs and customer feedback. Despite the benefits, implementing a microservice design adds complexity to network

connection, service coordination, and data consistency. Careful planning and execution are necessary for the management of inter-service communication and guaranteeing data consistency across services. A technology that enables you to accomplish these task is Docker, including its extension, Docker compose. As described in Section 2.2.2, it is a virtualization technology that enables the development of independent containers and their intuitive connection to a digital networking system. The architectural choice made for this project is consistent with the concept of microservice architecture, as seen in Fig. 3.1, in order to take full advantage of the benefits of this technology, particularly with regard to fault tolerance.

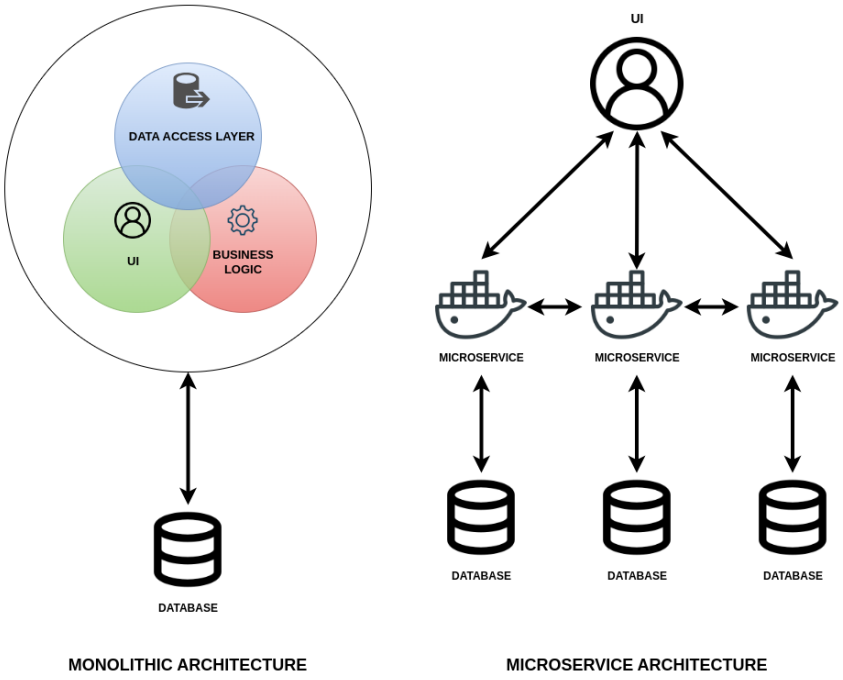


Figure 3.2: Monolithic architecture vs microservices architecture

3.1.1 Development methodology: CI/CD

The software development landscape is dominated by the Continuous Integration and Continuous Deployment (CI/CD) practice, which focuses on automating crucial processes including build, testing, and deployment. By streamlining operations and fostering better teamwork, this method strengthens the entire software delivery process. It improves problem fixes by minimizing dependency on manual testing and automating the development process and deliveries. Through fast and consistent configuration of new environments and a reduced chance of unintentional changes

or inconsistencies, this method simplifies the software development process. As a result, the developer can perform frequent releases and updates, test new versions, and undertake experiments. Additionally, CI/CD tools offer useful data for tracking development and pinpointing potential areas for improvement, such as build times, test coverage, defect rates, and test fix times. Development teams can identify possible areas for improvement and gauge how CI/CD processes are affecting company objectives by reviewing this data. The benefits offered by this approach, have been taken into account in this project, both to facilitate the development of the individual project, but also to make available the various updates to the working team. The project itself is part of a framework to improve the quality of work, the larger, of which the improvement of the building space management is only one module. The tools used to make it possible are those offered by the provider GitLab[1] , as you can see in Fig. 3.3.

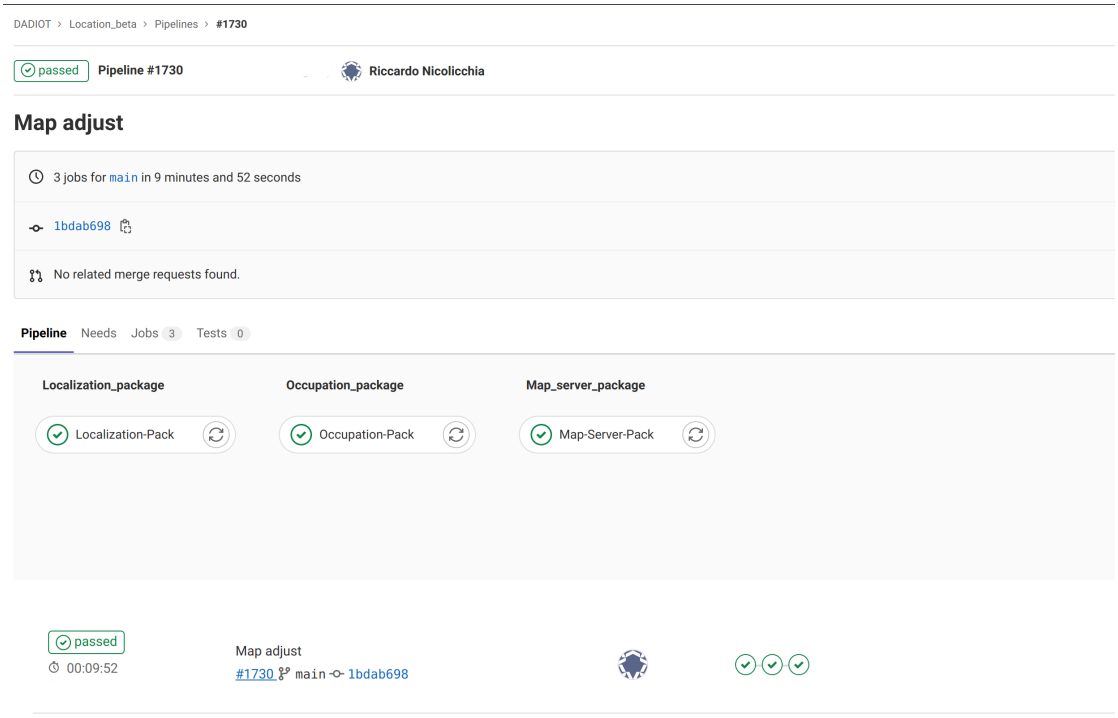


Figure 3.3: CI/CD GitLab tool

[1]GitLab CI/CD tool

3.2 System configuration

3.2.1 Gateway setting

System analysis and setting are the first, and most crucial, steps. Understanding the capabilities and features of the resources you are working with is crucial before you begin. The BLE gateways are the first resource to be considered, as mentioned in Section 2.1.1. To connect with another device, you must first reset each of them so that it acts as a standalone access point. After a quick authentication, you can connect to one of them and access the settings. To begin with, you must switch the gateway mode from AP—which is only practical for the initial connection—to station. The screen that appears when you do this is the one in Fig. 3.4. Entering the SSID(Service Set IDentifier,) and password in this area will enable WiFi.

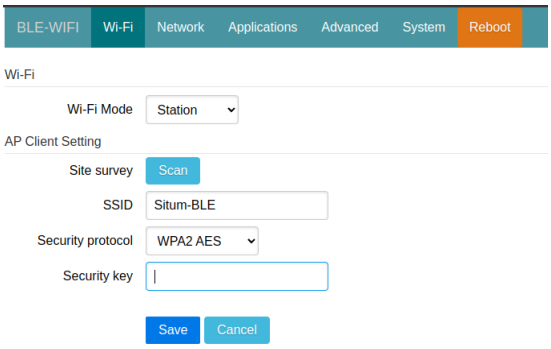


Figure 3.4: iGS01S gateway WiFi settings

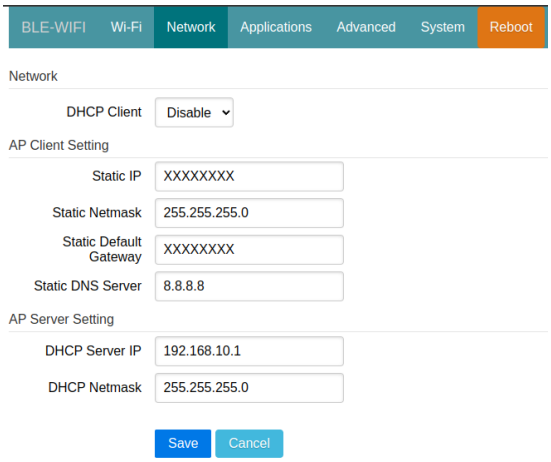


Figure 3.5: iGS01S gateway network settings

It is clear from Fig. 3.5 that the DHCP (**D**ynamic **H**ost **C**onfiguration **P**rotocol) protocol is not active. DHCP [60] is a network management protocol that adds the ability to automatically allocate reusable network addresses, i.e., IP addresses, as well as additional configuration options to the IP (**I**nternet **P**rotocol). The technology eliminates the need for network devices to be manually configured. If you want to have complete control over IP address assignment, you must remove it so that each device is assigned a unique IP address. The assigned IP is implied by the parameter "Static IP". The "Static Default Gateway" and "Static Netmask" specify the local network node that acts as the forwarding host and the range of addresses connected to the local network, respectively. 8.8.8.8 is the main DNS server for Google DNS when "Static DNS Server" is selected. Google DNS is a

public DNS service offered by Google that aims to make the Internet and the DNS system faster, safer, more secure, and more reliable for all Internet users. In the local company network, this configuration enables easy access to them, but it requires initial careful configuration. The following setting panel, depicted in Fig.

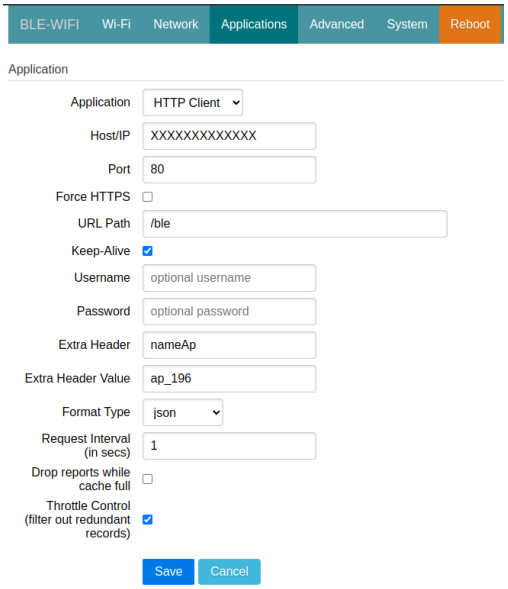


Figure 3.6: iGS01S gateway HTTP configurations

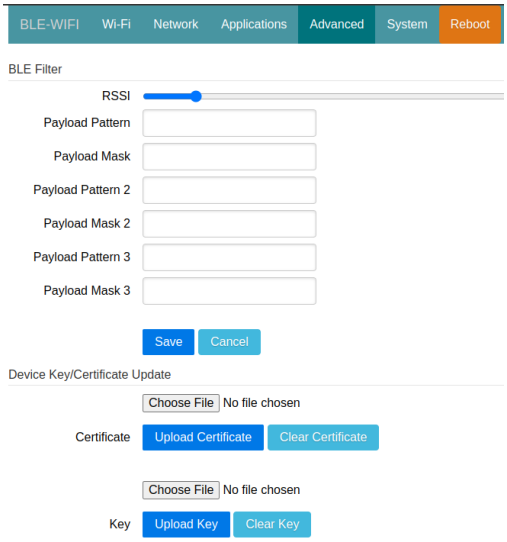


Figure 3.7: iGS01S gateway RSSI signal configuration

3.6, highlights all the settings required to connect to the server and send all the data collected by BLE devices. The host/IP designates the server's address, which may be a number or a string of letters. In the latter scenario, DNS consequently solves the address. Typically, the port is 80, the HTTP standard port, or 443, if the connection is encrypted using HTTPS. The URL (**U**niform **R**esource **L**ocator) path refers to the web server's route through which data is sent. Serving content at a specified URL is the fundamental idea behind every web framework. The term "routes" refers to the URL patterns used by an app to enable the content, such as a webpage or API response, to be served at these URLs. In this application, new data will induce Apache Kafka to take action, as will be explained later. As was mentioned in the previous section, each gateway is distinguished by a distinct IP address. As a result, it is possible to take advantage of this feature to simplify the following procedure by including a new field in the packet with the name "nameAp" and a value that refers to the final portion of the IP address. The JSON format for the packet is a development choice. Last but not least, the parameter request interval measures the space in seconds between requests. Because the device only

accepts integer values, it was set to the minimum value, i.e. 1, in order to have very responsive behavior and avoid creating a large data queue at the edge. In this manner, the system almost immediately picks up tag signals on the previously mentioned assigned server.

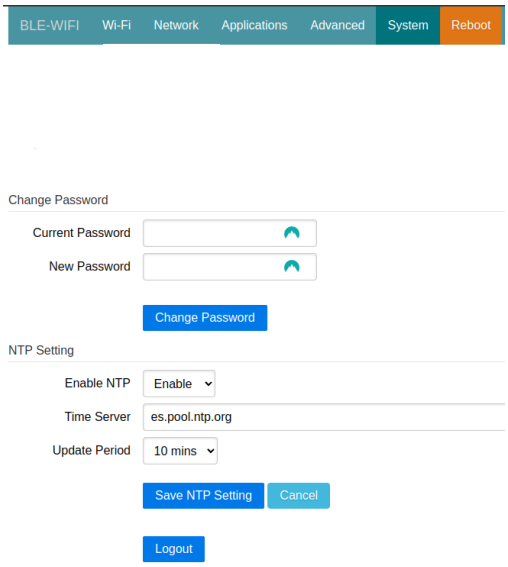


Figure 3.8: iGS01S gateway timing settings

The final two interface settings in Figs. 3.7 and 3.8 allow you to fine-tune two crucial technical elements. The first, respectively, enables you to apply a filter to the received packets' RSSI. To filter out all the packets and signals from other devices—such as smartphones, smartwatches, and other ones—present in the office that are completely irrelevant to the localization system, this is extremely important. The second make you able to reliably timestamp each packet, you can connect to an ntp.pool.org[2] server using the last setting panel. Every IoT application depends on time, but this one in particular depends on it to function. It is impossible to trilaterate the position without knowledge of the precise time of each signal.

[2]The NTP pool is a dynamic collection of networked computers that volunteer to provide highly accurate time via the Network Time Protocol to clients worldwide.

3.2.2 Tags settings

The tags are another essential part of the system because they allow for the location of individuals within the workplace. For a trustful result, these devices must be used properly. In this section, the emphasis will shift to the settings; the general functionality of these devices is covered in Section 2.1.2. As seen in Figs. 3.9 and 3.10, the manufacturer kontakt.io provides a simple but effective mobile app to both set up and monitor all of the nearby tags that are close to the phone. The actual settings of the tags can be seen in Fig. 3.10.

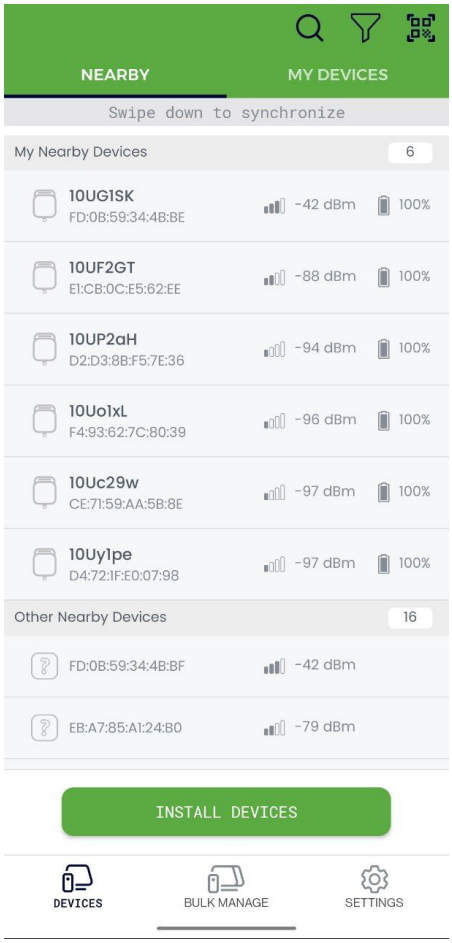


Figure 3.9: Tag setting app general view

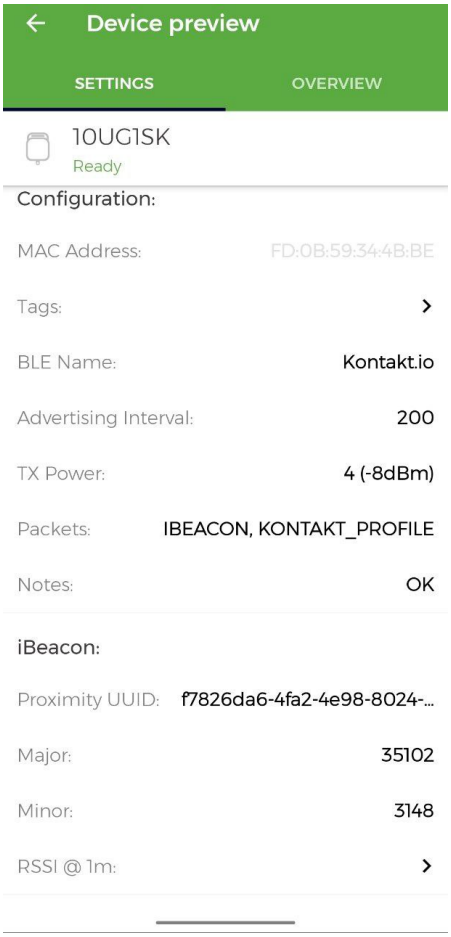


Figure 3.10: Setting panel single tag

The advertising interval and the transmission power are the two variables to take into account during setup for this project. Given that they will have an impact on the performance of the entire application, these parameters should be

selected as the network operates. There isn't a general use case, so every setup must be customized for each scenario. As consequence the system can function more effectively in terms of signal range, signal stability, and battery life with the right configuration. You couldn't have the best performances in all three fields in the large majority of applications. You have to choose the best trade off in order to reach the performance that best fit your needs. The actual needs that must be met in this project are essentially three. The first is adequate coverage with a radius of about 15–20 meters. The second is a quick packet transmission in order to have a strong and trustable cornerstone to ensure the data is published in almost real-time. The final one, signal stability, is influenced by both parameters, but the advertising interval has a greater impact. Therefore, in this situation, the need for faster data sinking will have a positive impact on the signal's stability. As shown in Fig. 3.11, the lower the interval, the better the stability. The use of quite high transmission power and a faster advertising interval unfortunately impact battery life. Although the battery life is not particularly brief, in the order of 1 year in duration, even if the settings do require battery effort.

Tx power	dBm	Expected range
1	-20	4 m
4	-8	30 m
7	4	70 m

Table 3.1: Beacon's transmission power description

The advertising interval is set to 200 ms while taking into account all of these technical considerations. This value provides sufficient packet flow and acceptable signal stability, which are both necessary for the system to function properly. Because the system gateways must deal with so many tags, there is actually a chance that some packets will get lost. Thus, following several experiments, this value is shown to be the appropriate trade off for a reliable but not particularly energy-consuming system. The transmission power setting, on the other hand, determines how powerfully the signal is transmitted by beacons. This is expressed in dBm (**d**eci**B**el **m**illiwatt) and ranges from 0 to 7 with 0 being the least powerful and 7 being the most powerful. As can be seen in Table 3.1, increasing the power of your transmission will broaden the reach of your signal. However, it is also true that a stronger transmission results in a greater energy drain and, as a result, a shorter battery life. The transmission power value selected for the purposes of this project is 4, or -8 dBm.

Figure 3.11: Advertising interval and signal stability

3.2.3 Apache Kafka broker

The Kafka broker is a neural point in the system. It is the module that allows the exchange of data between the various components of the system. The communication paradigm is publish/subscribe as described in Section 2.2.3. The framework in question basically allows for a very high horizontal scalability and also an excellent fault tolerance. The presence in an architecture of multiple brokers and certain retention policy, allow to have a strong availability of the data. In the context of the system developed, there was no need at this stage of development of scalability and fault tolerance, because the type of data collected even if lost in part does not compromise the system's functioning. In addition, the data managed by the system is not excessive, so this justifies the choice of a single broker for the management of the data collected. From a technical point of view, the broker used is the one provided by Confluent [61], because among the open-source solutions it is one that provides a more accurate documentation and a more extensive set of compatible tools. With regard to the graphic display of information by the broker Kafka, an open source solution [62] has been used that provides the broadest level of customization. The set of the broker Kafka, the Zookeeper[3] coordinator and the dashboard has been deployed using Docker Compose to allow easy interconnection of the 3 services, and manage it as a single stack, independent of the rest of the services developed. The structure of the deployment can be appreciated in Appendix A.
As a specific structure of this broker, from the point of view of the organization, the stream of data is managed by redirecting it to the topics defined arbitrarily. Each of them will have a defined structure, i.e. a predetermined number of partitions and a specific retention policy that allows to manage data storage as needed. From some points of view, a Kafka broker, within the limits, can be exploited as a database to accumulate data. It is possible to retain the data without deleting or compressing it automatically, indefinitely. In the case of this project, the defined topics can be seen in Table 3.2. As you can see, the retention policies are the same for each one, because the needs of data availability and storage are similar. In addition, as you can see in Fig. 3.12, the number of partitions is also different. The reasons for these choices will be discussed in the next sections. It will also be discussed how to

[3]Apache ZooKeeper is an open-source server for highly reliable distributed coordination of cloud applications

assign messages to the topics, in relation to the process performed.

Topic name	Partition	Retention Policy	Description
bleRSSI	5	DELETE	Sink continously all BLE RSSI data from HTTP
bleRSSI	5	DELETE	Sink continously all BLE RSSI data from HTTP
bleLocalization	1	DELETE	Receive all the calculated position
bleOccupation	1	DELETE	Receive all the occupation information in geoJSON format

Table 3.2: Kafka topics characterization

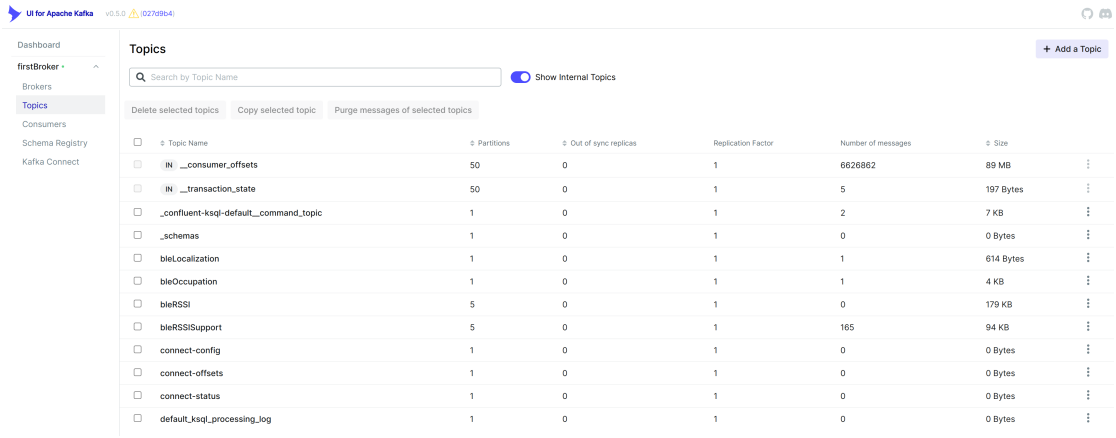


Figure 3.12: Topics dashboard overview

3.3 Services developed

In this section, all microservices related to the system's business logic for data collection from gateways, position and occupancy calculation, and visualization services will be described and detailed.

3.3.1 Deployment

All services developed, described in this section, have been containerized and treated as a single stack, within a Docker compose file. All of the services in question are

closely related to each other, and the malfunctioning of one may in some cases lead to malfunctions of the others. This can be seen from the architectural structure in Fig. 3.1. The structure of the deployment can be found in Appendix B

3.3.2 HTTP server for gateways

The web server used to sink all BLE packets was the first service produced. The fact that this web server is the node that must control the entire information flow arriving from all BLE gateways makes it a neuralgic point. It is because of this function that this component requires a specific design. The node should have a very quick response time when handling API calls from gateways and should be capable of handling numerous requests more or less simultaneously. These characteristics are fundamental because the webserver needs to manage dozens of gateways that continuously send HTTP requests every few seconds, and none of them are synchronized to send data with the others. The combination of Falcon, a WSGI[4] framework for creating extremely fast REST APIs, and the library bjoern, an HTTP/1.1 WSGI server, is the design choice that satisfies these requirements. They both and how they were used together were already covered in Section 2.2.1. The web server's actual operation is described in code snippet 3.1. As you can see, the code also makes use of the Producer module from the Apache Kafka Python library of Confluent in addition to the bjoern and Falcon libraries that were previously mentioned. A Falcon instance is started with the name "app" as the first step in the main (line 50). You should first develop a special class that will control the request for a particular route in order for the API to function. In this instance, catching the post requests is within the scope of the class KafkaRedirect. Line 51 is essential because it enables body data to be retrieved from each HTTP post request. Following the setting of this crucial feature, you must define the route, as seen in line 52. You specify the URL of the route and the custom object that must handle the packet arriving through this route using the instantiated Falcon object. Last but not least, you launch the bjoern WSGI HTTP server by binding the freshly created route. You have to indicate the web host on which the server is mounted, and specify the port to expose in order to listen to the requests. The kafkaRedirect serves as the web server's brains. It is a clear and simple Python class with the initial topic name, "bleRSSI," a counter that is an environment variable, and the flag requestStatus that can be used to identify errors in packet delivery to the Kafka broker as attributes. The *on_post* method, which defines the route for POST requests, is the main method defined in the class. When a new message is received, it is first parsed into JSON before being *produce()* and

[4]The Web Server Gateway Interface is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language

flush() into the active Kafka topic. The first will direct the JSON payload to a particular key and topic. In this instance, the custom name (*ap_N*) of the gateway that originated the post request serves as the key's representation. The N will be a value between the ones in shown in Fig. 3.13. The second method keeps trying until every message in the producer queue is delivered. The value in seconds enclosed in the brackets indicates the timeout.

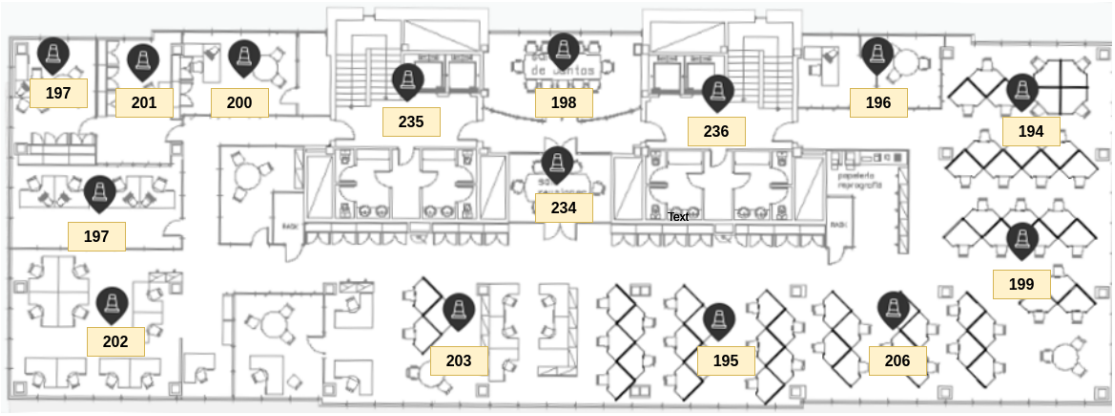


Figure 3.13: Base stations map

The production to Kafka broker is not always for the same topic, as the snippet demonstrates. Every time 250 messages are sent to the broker, the system will switch production between two topics, *bleRSSI* and *bleRSSISupport*, using an environmental variable, i.e., the object attribute *counter*. This design decision was made to offer a means of preventing the accumulation of excessive amounts of data. This mechanism takes advantage of topics' cleanup policies. The messages are stored in the topics for a maximum of 10 seconds before being completely deleted. Following a number of tests, the value of 250 was selected because it ensured both enough time to free the unused topic and enough time to ensure the system's proper operation. In summary, it takes the topic more than 10 seconds to accumulate 250 messages. By doing this, the unused topic will have enough time after switching to the other topic to be cleaned up and prevent data accumulation. The value of 250 is ideal for this use case, but in a more extensive scenario, it ought to be higher. This design decision was made for two reasons. Since there is a lot of data generated, clearing the space occasionally is a good practice to prevent system congestion. The final justification is that data collected after 4-5 seconds is completely useless for the system's goals. In the paragraph that follows, this statement will be better supported.

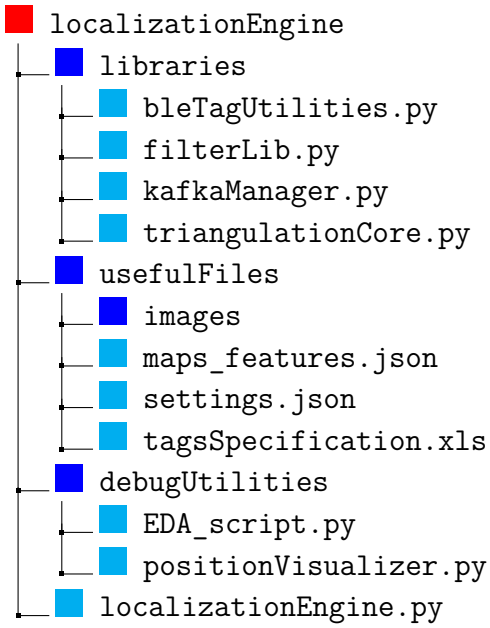
```
1 import bjoern, falcon
2 from confluent_kafka import Producer
3
```

```
# Create Producer instance
producer = Producer(config)

class kafkaRedirect(object):
    def __init__(self):
        self.topicName = 'bleRSSI'
        self.counter = 0
        self.requestStatus = 1

    def delivery_callback(self, err, msg):
        if err:
            self.requestStatus = 0
            self.err = err
        else:
            self.requestStatus = 1

    def on_post(self, req, resp):
        data = req.media
        jsonPayload = ujson.dumps(data)
        if (self.counter == 250):
            if (self.topicName == 'bleRSSI'):
                self.topicName = 'bleRSSISupport'
            else:
                self.topicName = 'bleRSSI'
            self.counter = 0

        self.counter += 1

        producer.produce(
            topic=self.topicName,
            value=jsonPayload,
            key=req.headers['NAMEAP'],
            callback=self.delivery_callback)
        if self.requestStatus == 0:
            resp.text = "ERROR"
            resp.status = falcon.HTTP_500
        else:
            resp.text = "SUCCESS"
            resp.status = falcon.HTTP_200
        producer.flush(0.5)

if __name__ == "__main__":
    # instantiate a callable WSGI app
    app = falcon.API()
    # long-lived resource class instance
    kafka = kafkaRedirect()
    # handle all requests to the '/ble' URL path
    app.req_options.auto_parse_form_urlencoded = True
    app.add_route('/ble', kafka)
```

```
53    bjoern.run(app, WEB_HOST, PORT, reuse_port=True)
```

Listing 3.1: HTTP server code

3.3.3 Localization engine



The system's functional center, which determines and handles the positions, is the localization engine. It makes use of Python libraries for the confluent Kafka client and Pandas. The actions of this element depends on the data consumed from bleRSSI and bleRSSISupport generated by the Bjoern/Falcon HTTP server. In essence, it converts the data from the unprocessed tag signals into positions that can be consumed. Fig. 3.14 gives a description of the overall operation flow.

To be clear in the description of the action performed by this service, it is necessary to define the four main structures that are essential to the system's operation, namely, bleTagInfo, dfTagStatus, dfKalmanFilters dfRSSICurrentFP, and dfRSSIBackupFP. Additionally, the parameters that are used to fine-tune the algorithm will be discussed. Understanding these parameters will allow you to comprehend how the algorithm works.

Structures

The first one is bleTagInfo. This dataframe includes the key information for each tag currently in use. Its structure is depicted in Fig. 3.15. It is essential to

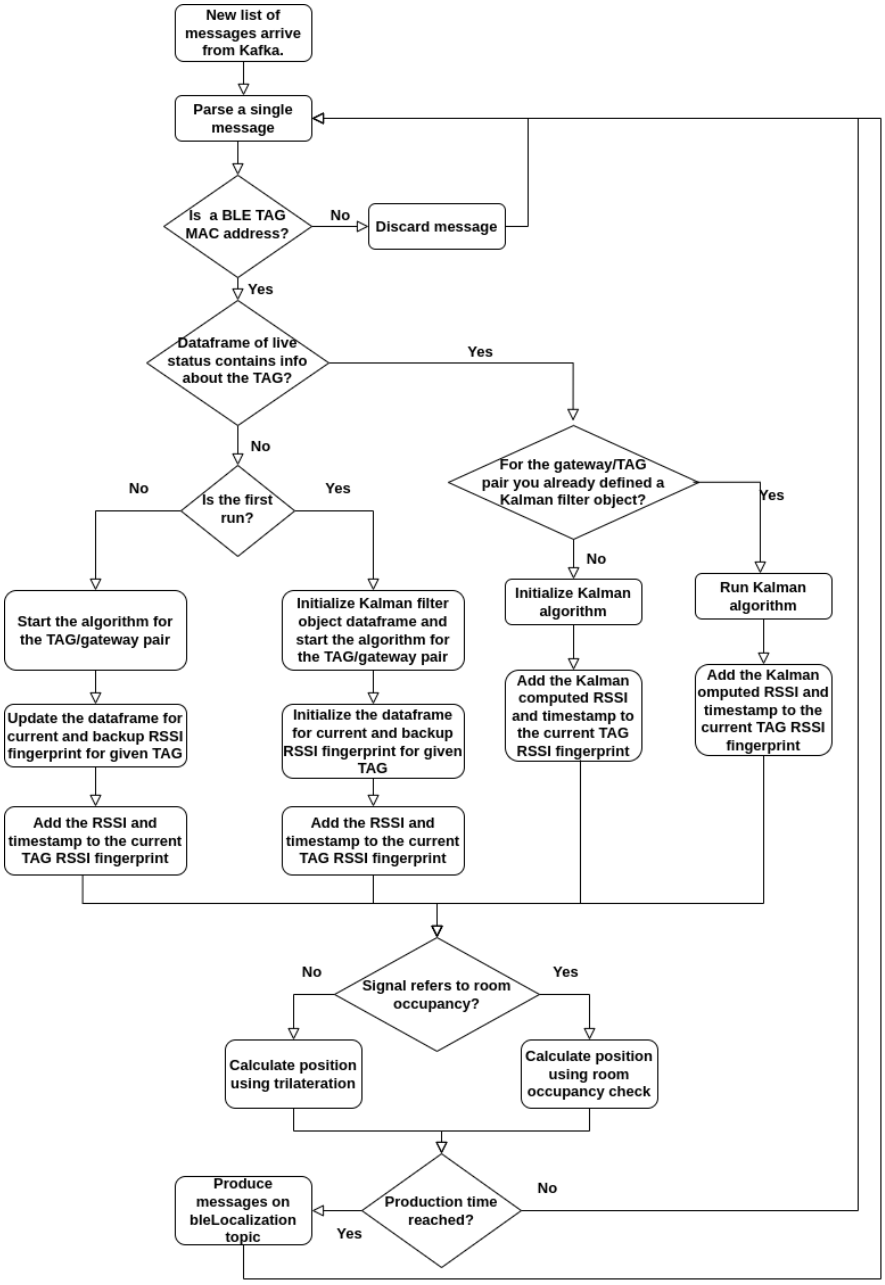


Figure 3.14: Localization engine operation flow

recognize which devices must have their signals discarded and which must have their signals analyzed by the system. The Excel file tagsSpecification.xls (that was provided to the system as input) and the data that were accessed directly through an API call from the Kontakt.io server make up the data that are stored

in this data structure. Recurrent API calls enable access to updated information regarding each tag's transmission feature. As you can see, it merely includes the details needed to identify each tag. The MAC address is the piece of information used to identify the target device. Each message that is received from gateways contains the MAC of the corresponding tag that sent the packet, as explained in Section 2.1.1. The MAC is linked to the tagID and it is the unique number that is written on each tag. You can find out who the owner is by using this information. Last but not least, for each device, the default RSSI signal at 1m as defined by the manufacturer is also available in addition to the current transmission power (the meaning is reported in Section 3.2.2).

	MAC_tag	tagID	Name	txPower	RSSI@1m
0	E797BFC256E1	10U51bz	Not_assigned	1	-84
1	EF7DD3F856BF	10UH1UJ	Not_assigned	1	-84

Figure 3.15: Dataframe of tags information

The second dataframe is dfTagStatus. This data structure is in charge of recording the historical position calculated and the adjusted RSSI at 1 meter. A new row is added when the system recognizes a new tag. Each line is distinguished by its distinct identifiers, MAC and tagID, as seen in Fig. 3.16. As previously stated, each tag has a default RSSI value at 1 m that varies depending on the transmission power. The manufacturer sets this value as a general guideline, but depending on the environment in which the device is used, it frequently needs to be modified. In the case of this project, the office context could result in extremely unstable signal propagation depending on the presence of barriers and windows. This is the rationale behind maintaining and updating the RSSI at 1 meter value in accordance with the situation. The last valid value discovered by the algorithm is represented on each line, along with the timestamp that goes with it. Overall, this data structure is crucial for tracking the adjusted RSSI at 1 meter and the historical position of detected tags. It allows for adjustments to be made based on the environment and ensures that the most recent valid RSSI value is used in calculations. This information is stored in a table with unique identifiers for each tag and timestamped values for each adjustment made.

	MAC_tag	tagID	pos_tracker	pos_history	last_RSSI@1m	last_ts
0	ED04D9931...	10UN2JY	[(1526, 205)]	[(1526, 205)]	-59.00000	1680367...
1	D4721FE00...	10Uy1pe	[(1532, 130)]	[(1532, 130)]	-57.00000	1680367...
2	DD217FDF...	10UG2JS	[(1513, 95)]	[(1513, 95)]	-57.00000	1680367...

Figure 3.16: Dataframe of tags current status

The management of the Kalman filter algorithm depends on the structure that is shown in Fig. 3.17. In the broader context of the trilateration algorithm, the algorithm provides the basis for denoising and signal tracking. Each cell in the dataframe has a KalmanFilterObj with the structure shown in Listing 3.2. For each gateway and tag pair, it is defined as an object. This is due to the fact that each tag's signal propagation in relation to each station is unique and requires a particular type of modeling. In the section that follows, the parameters of such an object will be described.

```
class KalmanFilterObj:
    def __init__(self, init_value, A=1, H=1, Q=0.1, R=20):
        self.X = init_value
        self.P = 0
        self.__A = A
        self.__Q = Q
        self.__H = H
        self.__R = R
        self.__K = None
```

Listing 3.2: Kalman filter object

	ap_194	ap_195	ap_196	ap_197	ap_198	ap_199	ap_200	ap_201	ap_202	ap_203	ap_206	ap_232	ap_234	ap_235	ap_236
ED04D9931E80	<libraries.filterLib.KalmanFilterObj ...	nan	<libraries...	nan	nan	<libraries...	nan	nan	nan	nan	<libraries...	nan	nan	nan	<libraries...
D4721FE00798	<libraries.filterLib.KalmanFilterObj ...	nan	<libraries...	nan	nan	<libraries...	nan	nan	nan	nan	<libraries...	nan	nan	nan	nan
DD217FDFF78A	<libraries.filterLib.KalmanFilterObj ...	nan	<libraries...	nan	nan	<libraries...	nan	nan	nan	nan	<libraries...	nan	nan	nan	nan
C6DBF694DB4E	nan	<libraries...	nan	nan	nan	nan	nan	nan	nan	nan	<libraries...	nan	nan	nan	nan

Figure 3.17: Dataframe for Kalman filter calculation support

The last two data structures are the ones that store a record of each tag's RSSI signal that the system has identified. The ID given to each BLE gateway is used to name the columns. The MAC address of the particular device serves as the row identifier for each MultiIndex row in this structure, which adds the RSSI value and the associated timestamp to this value. To be more specific, each cell contains the RSSI value of a specific station, identified by an ID, as received by a device, identified by the MAC, as well as the timestamp, which is critical in the

trilateration algorithm. The timestamp is updated with the one from the packet each time a new RSSI value arrives, replacing the previous one. Figures 3.19 and 3.19 depict the structures.

	ap_194	ap_195	ap_196	ap_197	ap_198	ap_199	ap_200	ap_201	ap_202	ap_203	ap_206	ap_232	ap_234	ap_235	ap_236
ED04D9931E80/value	-65.05	nan	-84.895	nan	nan	-72.116	nan	nan	nan	nan	-83.439	nan	nan	nan	-85.294
ED04D9931E80/ts	1680367...	nan	1680367...	nan	nan	1680367...	nan	nan	nan	nan	1680367...	nan	nan	nan	1680367...
D4721FE00798/value	-70.695	nan	-79.904	nan	nan	-71.547	nan	nan	nan	nan	-83.526	nan	nan	nan	nan
D4721FE00798/ts	1680367...	nan	1680367...	nan	nan	1680367...	nan	nan	nan	nan	1680367...	nan	nan	nan	nan

Figure 3.18: Dataframe for current RSSI tracking

	ap_194	ap_195	ap_196	ap_197	ap_198	ap_199	ap_200	ap_201	ap_202	ap_203	ap_206	ap_232	ap_234	ap_235	ap_236
ED04D9931E80	-65.0	nan	nan	nan	nan	-72.0	nan	nan	nan	nan	-83.0	nan	nan	nan	nan
D4721FE00798	nan	nan	-80.0	nan	nan	-72.0	nan	nan	nan	nan	-83.0	nan	nan	nan	nan
DD217FDFF78A	nan	nan	-79.0	nan	nan	-72.0	nan	nan	nan	nan	-84.0	nan	nan	nan	nan
C6DBF694DB4E	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

Figure 3.19: Dataframe for last valid RSSI fingerprint tracking

The principal distinction between the two structures is that dfRSSICurrentFP merely contains all new RSSI that arrived from the gateway associated with a specific tag, including any that are outdated or unusable for the calculation. Thus, it is a sort of buffer where all of a tag's RSSI values are sinking. The other, dfRSSIBackupFP, holds the arrays of valid values that were utilized in the most recent trilateration calculation, each referring to a particular tag.

Parameters

1. DBM_STARTER_TX_POWER_[NUM]_CORR : RSSI at 1m is not stable so each time is adjusted. This parameter refers to the dBm added to the kontakt.io standard to have a chance to find the correct one. Higher the value make possible to have a safer check, but a longer process to find optimal;

2. TIME_BOUNDARY_S : maximum time a Kafka packet is considered valid. Packets with timestamp older than this parameter w.r.t. current time are discarded ;

3. POSITION_VALIDITY_S: maximum time a position is valid. If a position for the same tag is retrieved in a time lower or equal to POSITION_VALIDITY_S then the last position retrieved still is valid, otherwise it is considered expired so next time is calculated again;

4. POSITION_TRACKER_BUFFER_LEN: length of buffer where the most recent positions are saved to evaluate movement/ adjust near position. Longer buffer means more stable positioning system but slower to recognize a change. Shorter buffer means more dynamical, so position could oscillate more frequently but the change are recognized faster;

5. MOVEMENT_THRESHOLD_PIXEL: maximum pixel distance to consider position static. If the difference between the position under exam and a new one is greater than MOVEMENT_THRESHOLD_PIXEL, the system consider this as a movement. To have an idea of the order in meters of this distance in pixel you have to take into account this equality:

$$distance[m] = \frac{distance[pixel] * MAP\_LENGTH[m]}{MAP\_LENGTH[pixel]}$$

6. NEAR_POS_VALIDITY_PIXEL: maximum pixel distance to consider new position calculated not an outlier w.r.t. the buffer ones. To have an idea of the order in meters of this distance in pixel you have to take into account the previous point equality.

7. ROOM_THRESH: threshold of signal strength in dBm that allows to recognize the occupation of a room, so skip the trilateration process;

8. MAP_PRODUCTION_TIME: it is the time set to trigger the production on Kafka topic bleLocalization;

9. TIME_VALIDITY_TO_PRODUCE: the time in seconds to consider a position valid to be produced on Kafka. Position older than this parameter w.r.t. current time are not produced;

10. Q and R: parameteres for tuning Kalman filter, that respectively are the measurement noise and the system noise;

11. N: parameter useful to calculate distance using the RSSI thar represents the path-loss exponent.

Process flow description

The algorithm 1, represents the pseudo-code of the entire algorithm, which high level representation is shown in Fig. 3.14. The process start at the first run by setting all the parameters mentioned in the previous subsection. To make possible the application to communicate with Kafka broker, it is fundamental to configure the producer and consumer. Notice that these operation are managed using the custom library *kafkaManager*. In this case the producer point to *bleLocalization*

topic. This topic presents only one partition (as shown in Fig. 3.12), because it has to manage not a large number of messages. Regarding the consumer it is connected to the two topic *bleRSSI* and *bleRSSISupport*, and to each of the partitions (5 for each one), to receive all the message that are published. From the lines 8/9 of the algorithm, it can be noticed that the consumer is configured inside a infinite loop, because if something wrong happen, in the communication with the broker, the consumer will be restarted and all the operations can continue as usual. The producer doesn't need this kind of treatment. Opposite to the consumer that has this pull behavior, so can suffer of critical interruption, the producer has push behavior, and if something is wrong, the worst scenario is that the message is not published, but it doesn't get stuck.

After that the operative part of the algorithm starts. Each 100 ms the consumer polls the broker to receive a message. As shown in Fig. 3.20 each message, that is consumed from Kafka, is a list of strings. Each message need to be iterated to analyze each packet. After a little processing, the key information, i.e. MAC, RSSI value and timestamp, are extracted. After this extraction, the MAC brought by packet is analyzed, and compared to the ones that are stored in bleTagInfo.

```
['$GPRP,E9ED9CC0020B,C81626CE8333,-87,0201060F166AFE0209020052F44175484E3072,1680367611',
 '$GPRP,F47B5664C5CF,C81626CE8333,-82,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E61F7AEB1C1,1680367611',
 '$GPRP,C46EDB230961,C81626CE8333,-90,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E7BBF02BFB3,1680367611',
 '$GPRP,E68F50A49D8B,C81626CE8333,-68,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E898DBA5CB8,1680367611',
 '$GPRP,FD0B59344BBF,C81626CE8333,-77,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E891E0C4CB8,1680367611',
 '$GPRP,E9ED9CC0020C,C81626CE8333,-85,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E332B61D7C2,1680367611',
 '$GPRP,DD217FDFF78B,C81626CE8333,-72,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E04A8F4D9B8,1680367611',
 '$GPRP,ED04D9931E81,C81626CE8333,-75,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893E3E0A1936B8,1680367611',
 '$GPRP,D4721FE00799,C81626CE8333,-75,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893EFB6FB5EDB8,1680367611',
 '$GPRP,E1CB0CE562EF,C81626CE8333,-75,0201061AFF4C000215F7826DA64FA24E988024BC5B71E0893EAB6AE0CDB8,1680367611']
```

Figure 3.20: RSSI Tag packet structure

If message refers to a recorded BLE tag, the packet is analyzed, otherwise is discarded. At the beginning the system has only the knowledge about the identity of the BLE tag involved by the system (bleTagInfo), but nothing about their current status. So each time a tag is detected for the first time, it is recorded in dfTagStatus, by adding a new row to the dataframe , and inserting the information arrived with the packet. If it is not the first time, that a sepcific tag is detected by the system, at this stage, dfTagStatus is not touched. The other two structures i.e. dfRSSICurrentFP and dfKalmanFilters, are also filled at this stage. In both case the dataframes have a structure in which , it has to be take into account not only the MAC but also the gateway from which the signal arrives. If it is the first time that a specific value referred to a tag-gateway pair:

- A KalmanFilterObj (as shown in the Listing 3.2) is initialized, using the RSSI value just arrived. This object then is stored in the cell of dfKalmanFilters, corresponding to tag-gateway pair.
- The RSSI value is directly stored in the corresponding to tag-gateway cell of dfRSSICurrentFP.

On the other hand if a value, of the corresponding pair, already arrived:

- The KalmanFilterObj contained is fed with the newly arrived RSSI value, so the algorithm can be run.
- The value calculated with the Kalman filter is retrieved and stored in the tag-gateway cell of dfRSSICurrentFP.

After the saving of data from Kafka has been defined, you have to calculate the positions.In the algorithm when a position is calculated, two fields are used for each tag in the dfTagStatus dataframe. The first is position_tracker, a list that contains the last positions calculated. Among these positions are also possible outliers, therefore non-validated positions. The function of this list is to support the detection of movements, detection of invalid positions and to help refine the position in case of close measurements. The length of this list will be equal to the parameter POSITION_TRACKER_BUFFER_LEN. The second field is position_history. That simply keeps track of all valid positions calculated. This allows you to have a clear idea of all the movements carried out by the tag. A position is calculated in these cases:

- It is the first time that the position for a particular tag is calculated, that is, that the position_history relative to that tag in the dfTagStatus dataframe is empty. In this case the position is saved both in the position_tracker, and in the position_history, it is then considered a valid position;
- The position is too old so it must be recalculated;
- The position_tracker has reached its maximum length so you have to check if there has been a movement or it is needed a correction of the current position. The check is performed using the function shown in Listing 3.3. How is it possible to see the control is done on the last valid position (current_pos), that is the last one present in the position_history and all position_tracker positions (track_history). In addition you need an input parameter called move_tresh was also passed to determine a unit of measurement to detect movement, which refers to MOVEMENT_THRESHOLD_PIXEL. As final parameter validity_thresh for distinguishing outliers from the positions to be taken into account, namely the constant NEAR_POS_VALIDITY_PIXEL.

This function returns two flags to report if a movement (isMovement is True) has been detected, or if the position needs to be corrected (adjustPosition is True), or none of the two operations. If a movement is detected then the returned position will be equal to the average of the position values in the position_tracker. If a position correction is needed, then the average will be made between the position tracker positions and the last valid position.

```
def detect_movement(current_pos,track_history,move_thresh,validity_thresh):
    isMovement=False
    adjustPosition=False
    # Create a boolean list that calculates the euclidean distances in pixel and
    # then check if this overcome the designed movement threshold
    movement_bool_list=[(euclidean_distance(current_pos[0],current_pos[1],pos[0],
    pos[1])>move_thresh) for pos in track_history]
    # If all the distances of the new position w.r.t. the ones of the track
    # history overcome the threshold it is a probable movement
    if movement_bool_list.count(True) == len(track_history):
        # The flag globalValidity makes possible to detect possible outliers.
        # If the new position is far from tracker position doesn't mean that it
        # is a movement. It is a movement if the previous position are near to
        # each others, so if respect the validity threshold of proximity
        globalValidity=True
        for item in track_history:
            itemCheck=([(euclidean_distance(item[0], item[1], pos[0], pos[1])<=
    validity_thresh) for pos in track_history].count(True)== len(track_history))
            globalValidity=globalValidity and itemCheck
        if globalValidity:
            isMovement=True
    # If all the distances of the new position w.r.t. the ones of the track
    history
    # stay beyond the threshold it is a probable that position need to be adjusted
    elif movement_bool_list.count(False) == len(track_history):
        # The flag itemCheckGlobal makes possible to detect possible outliers
        # If the tracker positions are near to each others the position adjust
        # could be performed.
        # Otherwise it is not performed because could be influenced by some
    outlier.
        itemCheckGlobal=True
        for item in track_history:
            itemCheck=([(euclidean_distance(item[0], item[1], pos[0], pos[1])<=
    validity_thresh) for pos in track_history].count(True)== len(track_history))
            itemCheckGlobal = itemCheckGlobal and itemCheck
        if itemCheckGlobal:
            adjustPosition=True
    return  isMovement,adjustPosition
```

Listing 3.3: Movement detection function

After being clear on the modalities and calculation context of the location, it is important to draw attention to an essential distinction in the type of computation that the algorithm performs. This check has to be made in connection to the value of the RSSI before calculating a position and checking the situations studied above. In general the types of localization offered are two, the occupation of the rooms and the localization in the open space, calculated by trilateration. The first type of localization is priority, so an initial check is done in this regard. The control over

the occupancy of the room is performed while looking at what kind of gateway the signal comes from. Information about gateway types and zone characterization is contained in the file *maps_features.json*, that is a geoJSON. Each zone is described as follows:

```
{   "type": "Feature",
    "id": "01",
    "properties": {
        "name": "ZONE_NAME",
        "occupation": 0,
        "assigned_station": "ap_XXX",
        "open_zone": 0
    },
    "geometry": {
        "type": "Polygon",
        "coordinates": [[[1108, 18],
                [1108, 198],
                [903, 198],
                [903, 18],
                [1108, 18]]]
            }
}
```

Listing 3.4: GeoJSON fragment

The file also contains a list of all the gateways that record you and the corresponding locations. Considering the description presented above, if the signal comes from a gateway assigned to a zone where the *open_zone* flag is set to 0, then you have to consider the scenario that the placement of the tag in question is inside the same room. To understand if this signal refers to the occupancy of a room, you have to consider the *ROOM_THRESH* parameter. If the value of RSSI is higher than this threshold, you can assume that the tag is very close to the gateway in question, then inside the room. This constant was determined after several tests, and therefore valid in the context of the office. In a different environment, with different conditions, this value may be different. The position will then be given by the known position of the gateway, plus the addition of a random range useful to drop the position inside the geo fences of the room, and not allow the overlap in the case of multiple surveys of occupation in that room. It is possible to see that the controls mentioned above are applied at line 40 of the algorithm 1. The first relates to the position_tracker's buffer's size. In this instance, simply the detection of a hypothetical movement is carried out. First off, it's crucial to remember that if the buffer has become too long, you must erase the oldest item (line 48). In the event that a movement is discovered, you must first empty the

position_tracker list, restart it with the newly determined position value, then save the actual record in the position_history. Another thing to check is whether the calculated location is the first for this tag; in that case, you need to save the location directly in the position_history. Another significant instance is out there. It is pointless to recalculate the location if the tag is stationary in the room and a position is already existing in the history and no movement has been observed. The most recent one found in position_history is maintained.

The second type of localization is that concerning the open-space part of the office. In this case, the calculation is more complicated. First, you must have at least 3 valid values referring to the tag in question in order to be able to calculate the trilateration. Valid value refers to those measurements whose timestamp minus the current timestamp (time.now()) is lower than the threshold given in input, TIME_BOUNDARY_S. After proving that you have the necessary information to locate the tag in question then, you proceed to the next steps, i.e. see if this is the first time that the position is calculated or if it is a re-calculation because the position has been lost. When calculating a position for the first time, the dfRSSIBackup dataframe needs to be examined. If a location has already been determined, it signifies that the dataframe line corresponding to the tag under investigation already has at least 3 values. The control will return in this situation with flag noBackup equal to True. The position_history is examined when a position is lost. Every time a position is lost, a standard value, that's (-1, -1), is put to the bottom of the list to signal that the position needs to be recalculated and that the previously saved position is no longer valid. This temporary value is deleted from the list prior to recalculation, and the new value is then stored. In this case the value returned by the check isPositionLost equal to True.

At this point begin the procedures of calculation of the distances described in the listing 3.5. In this fragment you can see that given the list of values RSSI, and note the of RSSI to 1 meter and the coefficient N, it is possible to calculate the distance between the tag and the gateway that recorded the RSSI value. The relationship between the RSSI value and the distance is described below [63][64][65]:

$$RSSI_d = -10 * n * \log(\frac{d}{d_0}) + RSSI_{d_0} \tag{3.1}$$

$$d = d_0 * 10^{\frac{RSSI_{d_0} - RSSI_d}{10*N}} \tag{3.2}$$

You can see that the signal strength of the BLE Beacons decreases exponentially. The strength of the signal is affected by the signal strength that the BLE beacon has, and that is quantified at a distance of one meter. In the above equation this value refers to the value of RSSI at d0, which is then 1 meter in this case. So this becomes a fundamental parameter in calculating distance, and it's important to calibrate it correctly. Its value obviously depends on the environment in which

the signal diffusion takes place and therefore you cannot assign a standard value a priori. Finally, the coefficient N represents the path-loss exponent defining the rate at which the power falls. Multiple values in the range of 2 to 3 were evaluated. The distances got dilated when the values were too close to 2, which increased the errors in the trilateration that resulted. With values near 3, the distances were underestimated making it challenging to calculate the trilateration because the algorithm's convergence took longer and the positions discovered were actually more erroneous.The values that have provided a more accurate solution are around 2.4. The final value chosen is 2.4.

```
def distanceCalculationWithN(RSSI, RSSI1m, N):
    # Retrieve the distance of the TAG using the classical propagation model of
    the signal in RSSI
    return 10**((RSSI1m - RSSI) / (10 * N))

def getDistances(RSSI_values,RSSI1m,N):
    #Calculate distances of the list of RSSI in input
    distances=[]
    for RSSI in RSSI_values:
        distance = (distanceCalculationWithN(RSSI, RSSI1m,N)* 1564) / 51.83
        distances.append(distance)
    return distances
```

Listing 3.5: RSSI distance calculation

The position can be trilaterated using the distances once they have been computed. A clear assumption must be made before the description can move forward. The z-axis distances are treated as constants in this calculation. The stations are all located at the same height for two reasons, and localized tags can typically be assumed to be at the same altitude as well. Localization tags can be placed on a table, in a pocket, or through a tie that is fastened around the neck. This lowers the cost of computing for a size that is difficult to estimate and is meaningless in practical applications. The function takes a list of calculated distances and an identifier of the plane to which it corresponds in order to load the information for the same, as shown in the X fragment of code. The Cartesian coordinates of the plan's stated input gateways are first loaded and saved in a list. We can say that every point in the Cartesian plane, whether the calculated position or the position of a gateway can be indicated as $P = (x, y)$. To have an idea on how the trilateration works it can be helpful analyze the geometrical meaning. A point $P = (x, y)$ on the Cartesian plane lies on a circle of radius d centred at (x_i, y_i) if and only if is a solution to this equation:

$$(x - x_i)^2 + (y - y_i)^2 = d^2 \tag{3.3}$$

With only one station available, we cannot identify the exact position of P. What we know, however, is how close it is. Each point that is at distance d from L is a potential candidate for P. This means that with only one beacon, our guess of P is

limited to a circle of radius d around P (Fig. 3.21). By employing not one, but two station, L_1 and L_2 ,the situation is better. Only the area inside the red circle can contain our object P. However, for the same reason, it can only be around the green circle's perimeter. This implies that it must be at the points where the two circles converge. Our hypothesis is immediately constrained to just two potential sites (in grey)(Fig. 3.22).

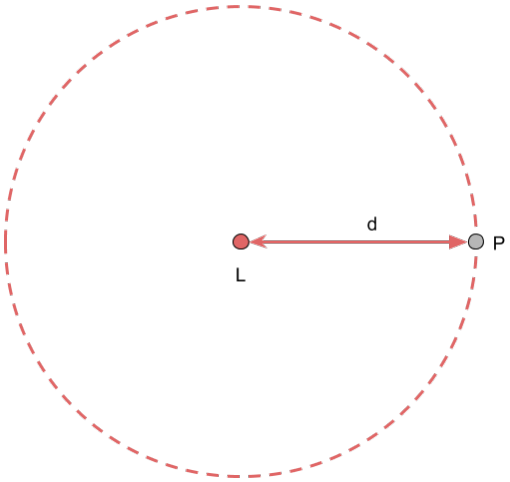


Figure 3.21: One station trilateration

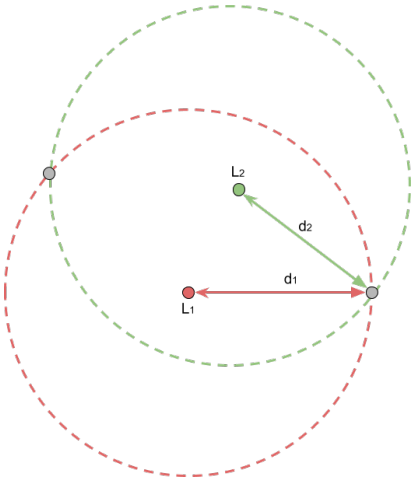


Figure 3.22: Two stations trilateration

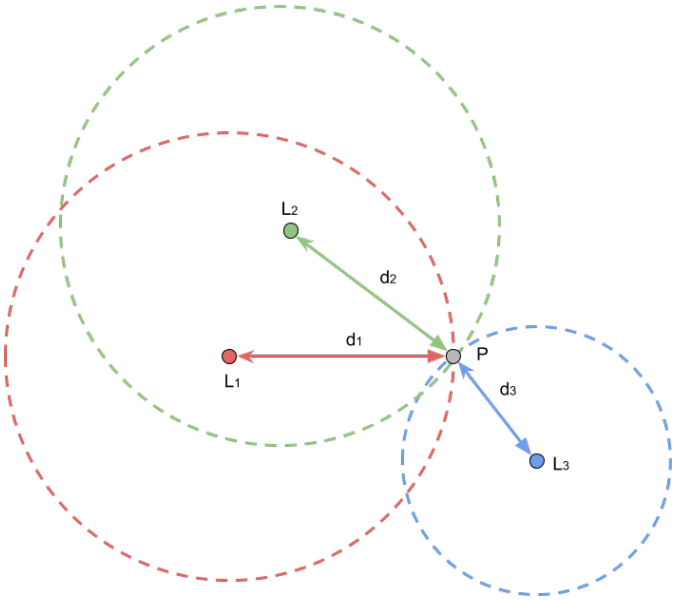


Figure 3.23: Geometrical representation of trilateration

The actual geometrical representation of the trilateration is shown in Fig. 3.23. In this figure it is observed because there is the need to know the distance from at least three known points. The problem of trilateration is solved mathematically by finding the point $P = (x, y)$ that simultaneously satisfies the equations of these three circles.

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = d_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = d_2^2 \\ (x - x_3)^2 + (y - y_3)^2 = d_3^2 \end{cases} \tag{3.4}$$

The goal of the initial position calculation method was to address the geometric issue. Trilateration can undoubtedly be seen (and solved) as a geometrical problem, but this is frequently not feasible. In order to use this mathematical modeling, measurements must be extremely accurate. In the worst situation, the set of equations will not have a solution if the circles do not converge to a single point. You haven't calculated a position in this situation. Even if we do have perfect accuracy, the mathematical method does not scale well. In fact, taking into account more than three stations makes the computations extremely complicated. So, it is possible to approach the trilateration problem from an optimisation perspective. Without considering circles and intersections, it can be viewed as a minimisation problem. Given a point $\hat{P}$, it is possible to estimate the point that minimizes this problem and best approximates the true P. This is easily accomplished by figuring out its distance from each station, L_i. If such distances exactly match the corresponding distances d_i (as determined by RSSI), then $\hat{P}$ is in fact P. It is presumed that $\hat{P}$ is further away from P the more it deviates from these distances. With this new formulation, it is necessary to define and minimize a specific error function while taking into account one source of error for each station. So the idea it is the following, i.e. define a contribution error for each station:

$$\begin{aligned} e_1 &= d_1 - dist(\hat{P}, L_1) \\ e_2 &= d_2 - dist(\hat{P}, L_2) \\ &\dots \\ e_n &= d_n - dist(\hat{P}, L_n) \end{aligned} \tag{3.5}$$

To combine them you can use the average of these contributions' squares. Due to the fact that squares are always positive, the prospect of negative and positive errors canceling each other out is eliminated. The result is the so called mean squared error (MSE), that is presented below:

$$\frac{\sum [d_i - dist(\hat{P}, L_i)]^2}{N} \tag{3.6}$$

In the code snippet 3.6 these calculations described can be seen in the first lines of code(1-18) in which the functions *euclidean_distance* and *___mse* are defined, and in the function *trilaterate* (42-49). In addition, it can be noticed that first value assigned to $\hat{P}$ is (0,0). Focusing on the rows 42-49 you can see that a *minimize*[5] function is used that is part of the Scipy library. The parameters set are respectively:

- *ftol*: aim for accuracy while determining the stop criterion's value. It is set at 10^{-2} because it is a reasonable value related to the desired outcome and allows for completion in a limited number of iterations.
- *maxiter*: it is the maximum number of iterations. It is set to 10^4 because it allows for the exclusion of erroneous calculations, given that during this calculation, convergence must occur fairly quickly. Iterating for a very high number of iterations, and consequently for a very high amount of time, becomes counterproductive and can result in the calculation of incorrect positions.

```
1  # This method retrieve the Euclidean distance given two points
2  def euclidean_distance(x1, y1, x2, y2):
3      p1 = np.array((x1 ,y1))
4      p2 = np.array((x2, y2))
5      return np.linalg.norm(p1 - p2)
6
7  # Mean Square Error
8  # locations: [ (x1, y1),      ]
9  # distances: [ d1,      ]
10 def ___mse(x, locations, distances):
11     mse = 0.0
12     # Given the position of the gateways(locations) and a given point (x) it
       calculates the
13     # euclidean distances between al these points. Then it calculates the MSE
       between these distances
14     # and the ones calculated with rhe values of RSSI
15     for location, distance in zip(locations, distances):
16         distance_calculated = euclidean_distance(x[0], x[1], location[0], location
       [1])
17         mse += math.pow(distance_calculated - distance,2.0)
18     return mse / len(distances)
19
20
21 def trilaterate(distancesTotal,FLOOR):
22     # distancesTotal ==> [12,56,nan,567,...]
23     # These value is ordered, so first value is referred to ap_1,
24     # second to ap_2 and so on. The order respect the way in which is written
25     # in usefulFiles/maps_feature.json under the key gateways
26     locations=[]
27     distances=[]
28     gatewayPos=load_gateway_positions(FLOOR=FLOOR)
29     initial_location=(0,0)
30     locationsTotal= [(gatewayPos[item][0],gatewayPos[item][1]) for item in
       gatewayPos]
```

[5]Scipy minimization function with SLSQP[66]

```
31      # Prepare the two lists that will be used in the minimization process.
32      # Append only the gateway positions(locations) of the one that has  a
        correspondant
33      # value of distance given by the RSSI calculation(value of distancesTotal in
        not NaN)
34      for i,dist in enumerate(distancesTotal):
35          if not pd.isna(dist):
36              locations.append(locationsTotal[i])
37              distances.append(dist)
38
39      # initial_location: (x, y)
40      # locations: [ (x1, y1), ... ]
41      # distances: [ d1,      ... ]
42      result = minimize(
43          __mse,                          # The error function
44          initial_location,              # The initial guess
45          args=(locations, distances), # Additional parameters for mse
46          method='SLSQP',             # The optimisation algorithm
47          options={
48              'ftol':1e-2,          # Tolerance
49              'maxiter': 1e+4       # Maximum iterations
50          })
51      return (int(result.x[0]),int(result.x[1]))
```

Listing 3.6: Trilateration core function

Therefore, the function *trilaterate* will return a correct position or, in the unlikely event of non-convergence, a value of none. At this point, a check is made (*getConsistency()* line 78 pseudo-code 1) to ensure that the calculated position is consistent given the data for the calculated position and the Cartesian boundary of the floor plan under analysis. To be more clear, given Cartesian boundary, X_{max} and Y_{max}, all points included in the area delimited by the points (0,0), (0,X_{max}), (Y_{max},X_{max}), (Y_{max},0) are considered. Where X_{max} and Y_{max} refer to the numerical values for the length (from 0 to Y_{max}) and width (from 0 to X_{max}) of the floor plan under examination.

If the validity check is positive, the position is valid. If the check fails to pass, the position is calculated iteratively, increasing of one the initial RSSI value at 1m each time, until either the consistency condition is satisfied or the time limit provided in the input is exceeded. This last condition was added to prevent the flow of operations getting blocked in an infinite loop. As previously discussed, the value of RSSI at 1 meter must be calibrated based on the environment in which the tag emits the signal. As a result, it is impossible to determine an initial value that is equal across all tags. Every time the position is calculated for the first time, or every time it is lost and recalculated, the check and, in the case of a negative result, this iterative correction, is done. It is important to note that when the condition is satisfied, before exiting the loop, the RSSI at 1 m value, at which convergence occurs after iterations, is saved in the dataframe dfTagStatus and replaces the previously saved value. All of these considerations are based on positions that have been calculated in the office's open space. Consequently, additional examinations

must be made to determine whether the calculated position is in a room, where it must be calculated using the previously described procedure, or it's in a location on the map where no localization is planned (such as the bathrooms). The control is carried out within the loop if the consistency requirement is satisfied, because, this kind of check is meaningless if the position is inconsistent, due to exceeding the calculation time limit. If the position is within a room or an area that is not authorized, the value taken into consideration is the most recent one that is still valid in the position_history of dfTagStatus. A null value is assigned if a valid position is not available. Following this, if the position's calculation did not exceed the time limit, the same checks described before for calculating the occupancy of the room are performed to detect any movement or the need for a position adjust.

When the position calculation is complete, the algorithm's 1 line 134 is reached. At this point, all the columns in the dataframe dfTagStatus are analyzed, specifically the field pos_history, to find all the tags with validation positions by examining the timestamps in each column. All viable positions are saved in positionListToProduce. The positions are not published each time they are calculated but rather accumulate and are published at regular intervals determined by the constant MAP_PRODUCTION_TIME. This ensures that production does not delay the calculation of positions excessively. The position list is converted to JSON at the time of publication and take the form seen in Fig. 3.24. The publication of the Kafka message will take place on the topic "bleLocalization," and each message will include as its key the name of the floor to which the location calculation is connected. After publication, the described process is repeated in an iterative fashion for Kafka's subsequent package to arrive. If there is no fault related to the connection to Kafka or if the algorithm is not manually stopped, the process will run endlessly. In the event of error in computation, calculation, or overall algorithm with runtime errors, the process is restarted, allowing calculations to continue without the application crashing. It should be noticed that each time the algorithm is permanently stopped, the customer is also disconnected from the group of consumers.


```
Out[7]: '[{"floor": 1, "x": 1526, "y": 205, "id": "10UN2JY", "name": "Test 2", "ts": "2023-04-01T18:46:51.425587"}, {"floor": 1, "x": 1532, "y": 130, "id":
 "10Uy1pe", "name": "Test 5 ", "ts": "2023-04-01T18:46:51.435706"}, {"floor": 1, "x": 1513, "y": 95, "id": "10UG2JS", "name": "Test 1", "ts":
 "2023-04-01T18:46:51.445980"}, {"floor": 1, "x": 1518, "y": 155, "id": "10UF2GT", "name": "Juan Sebasti\\u00e1n Ochoa", "ts": "2023-04-01T18:46:50.993217"},
 {"floor": 1, "x": 1544, "y": 295, "id": "10UP1r2", "name": "Test 6", "ts": "2023-04-01T18:46:51.003545"}, {"floor": 1, "x": 1414, "y": 505, "id": "10UG1SK",
 "name": "Riccardo Nicolicchia", "ts": "2023-04-01T18:46:50.965376"}]'
```


Figure 3.24: Localization engine JSON packet structure

Kalman filter

Given the very variable and unstable nature of the RSSI values, it was decided to adopt a correction and filtering of these values using a mathematical support. The

options you could use were different. Before the choice of the Kalman filter, several tests were carried out with the particle filter and the fast fourier transform. But the performance and flexibility provided by the Kalman filter made the decision hang on it [67]. In wireless communication systems, RSSI Kalman filtering has various advantages [68][69], including:

- Noise reduction: environmental factors, interference, and noise frequently affect RSSI measurements. The noisy RSSI observations can be filtered using a Kalman filter, producing an estimate of the solid RSSI value that is smoother and more precise.
- Adaptability: in wireless contexts, interference, multipath fading, and signal intensity attenuation can all affect RSSI measurements. By revising its estimations in light of newer measurements, the Kalman filter is able to adjust to these variances.
- Real-time estimation: as newer data become available, the recursive Kalman filter updates the state estimate and error covariance in real-time. This characteristic makes it ideal for real-time systems where prompt or almost prompt updates are needed.

The Kalman filter [70][71][72] is a state estimator that uses noisy observations to estimate some unobserved variable. As it takes into consideration the past measurements, it is a recursive algorithm. In the scope of this project the filter is used to estimate the actual RSSI. There are different variation of this algorithm, but it was chose the classical one, that relies on linear models. In other words, the change from one state to the next and from measurement to state should both be linear. The transition model's general form is as follows:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{3.7}$$

The current state vector x_t is affected by the noise contribution ϵ. This contribution ϵ is known as *process noise*. It refers to the ambiguity or variability in the system's evolution that is being modelled. It quantifies the difference between the expected and actual state transitions while taking into account variables that the model does not explicitly account for, such as outside disruptions, unmodeled dynamics, or innate system variability. To account for the innate unpredictability in RSSI observations that the model cannot fully capture, the Kalman filter for RSSI filtering integrates process noise. Process noise is represented as a random variable with Q as its covariance matrix (Equation 3.8).

$$p(\epsilon) \sim \mathcal{N}(0, Q) \tag{3.8}$$

It is assumed that all estimates have been performed considering that there are no movements occurring within the system during a estimate. As a result, during

each calculator the filter take a snapshot of the real environment and calculates the new RSSI measurement. In the proposed arrangement, component u_t, i.e. the control input (includes knowledge or information about the dynamics of the system or outside factors that have an impact on the evolution of the state), is regarded as zero.In accordance with the choice to take into account the static system, the matrix A, i.e. state transition matrix (captures the linear relationship between the present state and the following state),is regarded as an identity matrix. A relatively straightforward model is produced by these two changes:

$$x_t \approx x_{t-1} + \epsilon_t \tag{3.9}$$

Following the definition of the transition model, the observation model must be defined in the manner described below.

$$z_t = C_t x_t + \delta_t \approx x_t + \delta_t \tag{3.10}$$

The z_t stands for the measurements vector, and as shown is affected by the noise contribution δ. The δ is the *measurement noise*. It describes the uncertainty or error underlying measurements obtained from real-world systems. When measuring RSSI, noise takes into consideration numerous kinds of interference and represents the difference between the real signal strength and the measured RSSI value. The filter can balance the projected RSSI value and the measured RSSI value by taking into account the measurement uncertainty in order to get a more precise estimation of the real signal strength. Measurement noise is modeled as a random variable, precisely a normal random variable (Equation 3.11) characterized by its covariance matrix R.

$$p(\delta) \sim \mathcal{N}(0, R) \tag{3.11}$$

Both the error covariance matrix and the estimation of the true signal strength are updated using it. By taking measurement noise into account can reduce the impact of measurement fluctuations and improving the overall accuracy of RSSI-based localization or tracking applications. It is expected that the state variables can be directly measured, the matrix C is regarded as an identity matrix.In practice, each component of the state vector x_t immediately correlates to the corresponding component of the measurement vector z_t. The algorithm splits into two phases **prediction** and **update**.Taking into account that the previously stated conditions, namely $A = I$, $u = 0$, and $C = I$, are still valid and in addition the value of Q and R are assumed constant.The following operations define the **prediction** operations:

State Prediction

$$\bar{\mu_t} = \mu_{t-1} \tag{3.12}$$

Error Covariance Prediction

$$\bar{\Sigma_t} = \Sigma_{t-1} + Q \tag{3.13}$$

It is crucial to keep in mind that x_t represents the true value of the state and $\bar{\mu_t}$ is the actual prediction. So, based on the prior state estimate, at time step t-1, the $\bar{\mu_t}$ represents the expected state estimate at time step t. The matrix $\bar{\Sigma_t}$ represents the estimated error covariance matrix at time step t based on the actual error covariance matrix and the uncertainty introduced by the process noise covariance matrix Q.

Then the **update** steps:

Kalman gain

$$K_t = \bar{\Sigma_t}(\bar{\Sigma_t} + R)^{-1} \tag{3.14}$$

The Kalman gain is calculated using the covariance prediction estimate while also assuming the contribution of measurement noise R. The Kalman gain is then applied as a weight on the estimated state and measurement residual. The measurement residual is the difference between the actual measurement, z_t and the predicted measurement, $\bar{\mu_t}$. Gain is crucial since it measures the accuracy of the measurements and modifies the measurement residual's contribution to the state update. A lower Kalman gain emphasizes the anticipated state estimate more while a higher Kalman gain lends more weight to the measurement residual, indicating greater confidence in the measurements.

State update

$$\mu_t = \bar{\mu_t} + K_t(z_t - \bar{\mu_t}) \tag{3.15}$$

Error Covariance Update

$$\Sigma_t = \bar{\Sigma_t} - (K_t\bar{\Sigma_t}) \tag{3.16}$$

Finally, after adding the measurement data, the updated error covariance embodies the reduction in uncertainty regarding the state estimate. The metrics that will be used as filtered value in the calculation of position is the μ_t of Equation 3.15. In the context of this project, the algorithm's steps are completed in real

time. In other words, each time a new measurement comes along, an algorithm step is run while keeping the parameters and historical data from previous steps saved, and a new value of μ_t to use in the calculation is retrieved.

Algorithm 1 Localization engine algorithm: procedure to calculate the positions given the raw RSSI values

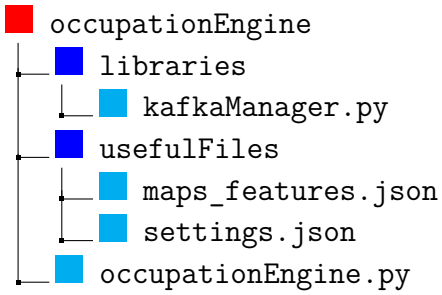
```
1: procedure Localization engine()
2:     programStarter()                                  ▷ Initialize all global variables
3:     ▷ Initialize kafkaManager obj
4:     dfTagInfo ← Excel+Kontakt data
5:     dfTagStatus, dfKalmanFilters,dfRSSICurrentFP,dfRSSIBackupFP ← [ ]
6:     producer ← configProducer
7:     while True do
8:         consumer ← configConsumer
9:         kafkaManager.assignConsumerToPartition(consumer)
10:        try
11:            while True do
12:                consumer.poll(0.1)
13:                if msg is None then
14:                    continue
15:                else if msg is Error then
16:                    raise Exception
17:                else
18:                    for packet in msg do
19:                        try
20:                            if MAC is Valid then
21:                                if MAC first time record then
22:                                    dfTagStatus[MAC] ← packet
23:                                else
24:                                    continue
25:                                end if
26:                                if (MAC,AP) first time record then
27:                                    KalmanFilterObj(packet[RSSI], A, H, Q, R)
28:                                    ▷ Related to (MAC,AP) cell
29:                                    dfKalmanFilters ←KalmanFilterObj
30:                                    ▷ Related to (MAC,AP) cell
31:                                    dfRSSICurrentFP ← packet[RSSI]
32:                                else
33:                                    dfKalmanFilters[MAC,AP].run_algorithm(packet[RSSI])
34:                                    ▷ Related to (MAC,AP) cell
35:                                    dfRSSICurrentFP← dfKalmanFilters[RSSI]
36:                                end if
37:                                roomOccupied = check_room_occupancy()
38:                                if roomOccupied  then
39:                                    ▷ Calculate pos
40:                                    dfTagStatus[MAC,ts] ← time.now
41:                                    dfTagStatus[MAC,pos_track] ← pos
42:                                    if dfTagStatus[MAC,pos_track] = MAX then
43:                                        isMovement = detect_movement()
44:                                        if isMovement then
45:                                            ▷ Calculate pos by average
46:                                            ▷ all position in tracker
47:                                        end if
48:                                        dfTagStatus[pos_track].pop(0)
49:                                    end if
50:                                    if isMovement then
51:                                        dfTagStatus[MAC,pos_track].clear()
52:                                        dfTagStatus[MAC,pos_track] ← pos
53:                                        dfTagStatus[MAC,pos_his] ← pos
54:                                    else if len(dfTagStatus[MAC,pos_his])==0 then
55:                                        dfTagStatus[MAC,pos_his] ← pos
```

```
56:                                 else
57:                                     pos=dfTagStatus[MAC,pos_his][-1]
58:                                 end if
59:                             else
60:                                 if trilateration is feasible then
61:                                     RSSITmp← dfRSSICurrentFP[MAC]
62:                                     ▷ Get value isPositionLost
63:                                     ▷ Get value noBackup
64:                                     if noBackup or isPositionLost then
65:                                         d=getDistances(RSSITmp,RSSI1m,N)
66:                                         pos = trilaterate(d)
67:                                         ▷ If the RSSI at 1m makes possible to
68:                                         ▷ have a consistent position, given X-Y limits
69:                                         ▷ in input, otherwise adjust the RSSI
70:                                         ▷ till a consistent result is reached
71:                                         consistent=getConsistency(pos,lim)
72:                                         ▷ In addition to consistency also a
73:                                         ▷ timer is set to avoid infinite loop
74:                                         while not consistent & TIME_LIM do
75:                                             RSSI1m = RSSI1m - 1
76:                                             d=getDistances(RSSITmp,RSSI1m,N)
77:                                             pos = trilaterate(d)
78:                                             consistent=getConsistency(pos,lim)
79:                                             if consistent is True  then
80:                                                 dfTagStatus[MAC,RSSI1m]=RSSI1m
81:                                                 ▷ Check the zone and return
82:                                                 ▷ flags closedRoom
83:                                                 ▷ and noTrack
84:                                                 checkZone(pos,map)
85:                                             end if
86:                                         end while
87:                                         if closedRoom or noTrack  then
88:                                             pos=dfTagStatus[MAC,pos_his][-1]
89:                                             ▷ If position history is empty
90:                                             pos= None
91:                                         end if
92:                                         if pos not None TIME_LIM then
93:                                             dfTagStatus[MAC,ts] ← time.now
94:                                             dfTagStatus[MAC,pos_track] ← pos
95:                                             if dfTagStatus[MAC,pos_track]= MAX then
96:                                                 ▷ Calculate isMovement
97:                                                 ▷ adjustPosition with
98:                                                 detect_movement()
99:                                                 if adjustPosition then
100:                                                    ▷ Calculate pos by average
101:                                                    ▷ between last position in history
102:                                                    ▷ and all position in tracker
103:                                                end if
104:                                                if isMovement then
105:                                                    ▷ Calculate pos by average
106:                                                    ▷ all position in tracker
107:                                                end if
108:                                                dfTagStatus[MAC,pos_track].pop(0)
109:                                            end if
```

```
                                        if isMovement or adjustPosition then
                                            dfTagStatus[MAC,pos_track].clear()
                                            dfTagStatus[MAC,pos_track] ← pos
                                            dfTagStatus[MAC,pos_his] ← pos
                                        else if noBackup then
                                            dfTagStatus[MAC,pos_his] ← pos
                                        else
                                            pos = dfTagStatus[MAC,pos_his][-1]
                                            ▷ If position history is empty
                                            pos= None
                                        end if
                                        dfRSSIBackupFP.at[MAC,AP]← RSSITmp
                                    else
                                        dfTagStatus[MAC,ts] ← time.now
                                        pos = dfTagStatus[MAC,pos_his][-1]
                                        ▷ If position history is empty
                                        pos= None
                                    end if
                                else
                                    dfTagStatus[MAC,ts] ← time.now
                                    pos= dfTagStatus[MAC,pos_his][-1]
                                end if
                            end if
                        end if
                        ▷ Search in dfTagStatus[pos_his] the position associated to
                        ▷ each tag that respect the condition:
                        ▷ dfTagStatus[ts]-time. now < TIME_VALIDITY_TO_PRODUCE
                        ▷ and save then as JSON in positionListToProduce
                        cond1=len(positionListToProduce)>0
                        cond2=time.now-REFRESH_POSITION_TIMER>MAP_PRODUCTION_TIME
                        if cond1 and cond2 then
                            for item in positionListToProduce do
                                msg ← JSON(floor,x,y,id,name,ts)
                            end for
                        end if
                        producer.produce(topic,value = message,key = floor)
                        producer.flush(0.5)
                        REFRESH_POSITION_TIMER = time.time()
                        kafkaManager.counter += 1
                    else
                        continue
                    end if
                catch Exception
                    MESSAGE PROCESSING ERROR!
                    exit(104)
                end try
            end for
        end if
    end while
catch KeyboardInterrupt
    PROGRAM MANUALLY STOPPED
    exit(0)
catch Exception as whatever_it_is
    KAFKA PROCESS ERROR
finally
    consumer.close()
    kafkaManager.updateConsumerGroup()
end try
end while
end procedure
```

3.3.4 Occupation calculator engine

Another part of the framework is the occupation engine. It is built on top of the services previously mentioned. It particularly depends on the message generation of the localization engine provides. The directory demonstrates that the service manages the connection to the Kafka broker using the KafkaManager library. The script also makes use of the same geoJSON description of the map that is found in maps_feaure.json.



This module's settings are less complicated. It only requires the default Kafka settings the default Kafka settings to publish to the topic *bleOccupation* and to read from the topic *bleLocalization* . The snippet of code 3.7 explains the behavior. All connection options to the Kafka broker are first loaded. The connection to Kafka is then established, allowing for both consumption and production. Just after those initial operations, an infinite loop that periodically polls the Kafka broker to receive and process packets is started. Upon arrival, a valid message has the structure shown below:

```
1 [{"floor": 1, "x": 1530, "y": 230,"id": "XXXX", "name":
     "XXXX", "ts": "1998-03-02T00:00"},
2 {"floor": 1, "x": 1545, "y": 233, "id": "YYYY", "name":
     "YYYY", "ts":"1998-03-02T00:00"}, ...]
```

The message is a list of precise coordinates referring to a particular floor, as you can see. Thus, the algorithm begins by checking the floor number. The following step involves iterating through the various floor zones that are represented in the floor features object, which was defined in line 17. The data is taken from the mapsFeature.json file (structure highlighted in the listing 3.4). Because the information published on the broker is a modified version of the original geoJSON obtained from the aforementioned JSON file, the data is kept in an object called geoFenceManager. In order to restart with the initial data when another message

arrives, you must in practice have two operational versions of these data that you can modify at runtime. As you can see from the snippet, the geoFenceManager is a very simplistic Python class. You have two attributes: backupCopy, which is private, and features, which is public. The first is initialized at the start of the algorithm and is never changed after that. The second one is the one that undergoes runtime changes and is refreshed with the help of the class method refreshValues. This class employs the deepcopy technique. Normally, a shallow copy of the data is made when you copy or initialize a variable. An object that contains the reference to the original elements is created by a shallow copy. A shallow copy merely copies the reference to nested objects; it does not actually copy the nested objects themselves. This indicates that the reference to the same nested objects is shared by the original variable and the copy. A deep copy, on the other hand, creates a new object and adds copies of nested objects that were present in the original elements in a recursive manner. This indicates that the copy and the original variable are independent.

To summarize, you move on to the next step when one of the items on the list in the Kafka message matches the floor number of a floor specified in the geoJSON (always same structure of listing 3.4) saved in the geoFenceManager. You repeat this process for all of the floor's zones that are defined by geofences and other features. They outline the building's perimeter based on a predetermined reference system. You use the library Shapely that is described in Section 2.2.1 to determine whether a position coming from Kafka fits inside a particular region. The coordinates of a point are used to define a point object. You must be aware that this point is made using the Figure 3.25's pixel map reference system. Applying a correction factor to the y coordinate will make it a valid point in the live map's reference system. The new point will be equal to (x_p,$C_p - y_p$) given a point (x_p,y_p), where C_p is the length in pixels of the short side of the map.

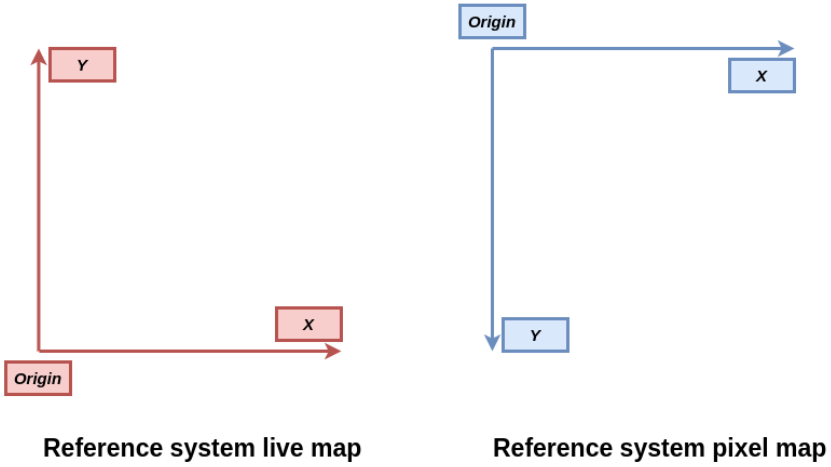


Figure 3.25: Reference systems of developement maps

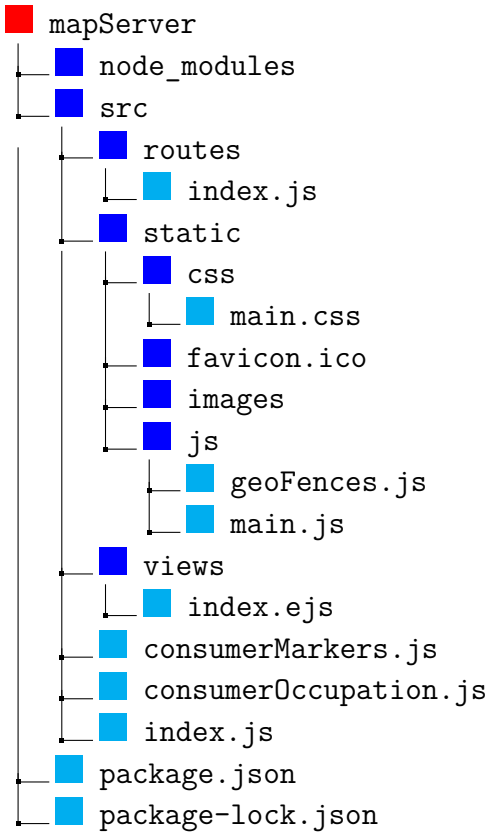
You now define a Polygon object and use the method *contains()* to determine whether the point is contained within the polygon. When the true condition is discovered, you add one to the geoFanceManger object's correspondent occupation field. Finally, after iterating through each position in the Kafka packet's list, the entire geoJSON is produced using the floor number as a key, and the runtime modified geoJSON is reset.

```
from copy import deepcopy
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
import libraries.kafkaManager as kM
from confluent_kafka import Consumer, Producer
import json
class geoFenceManager:
    def __init__(self,FLOORS_FEATURES):
        self.__backupCopy=deepcopy(FLOORS_FEATURES)
        self.features=FLOORS_FEATURES
    def refreshValues(self):
        self.features=deepcopy(self.__backupCopy)
if __name__ == '__main__':
    programStarter()
    floor_features = geoFenceManager(FLOORS_FEATURES=FLOORS_FEATURES)
    kafkaManager = kM.kafkaManager(
        BROKER_LINK,
        CONSUMER_TOPICS,
        INITIAL_CONSUMER_TOPIC,
        PRODUCER_TOPICS,
        INITIAL_PRODUCER_TOPIC)
    configConsumer = kafkaManager.chargeConsumerSettings()
    configProducer = kafkaManager.chargeProducerSettings()
```

```
    producer = Producer(configProducer)
    consumer = Consumer(configConsumer)
    consumer.subscribe(kafkaManager.consumerTopicList)
    kafkaManager.assignConsumerToPartition(consumer)
    try:
        while True:
            msg = consumer.poll(1)
            if msg is None:
                print("Waiting...")
            elif msg.error():
                print("ERROR: %s".format(msg.error()))
            else:
                for item in json.loads(msg.value().decode()):
                    LOCALIZED = False
                    for floor in floor_features.features:
                        if(item['floor']==floor['floor_number']):
                            for zone in floor['areasGeoFences']:
                                point = Point(item['x'],
                                FLOOR_Y_ADJUST - item['y'])
                                polygon = Polygon(zone['shape'])
                                if polygon.contains(point):
                                    zone['occupation']+=1
                                    LOCALIZED=True
                                if LOCALIZED:
                                    break
                        if LOCALIZED:
                            break
                for floor in floor_features.features:
                    message = json.dumps(floor['areasGeoFences'])
                    producer.produce(
                        topic=kafkaManager.actualProducerTopic,
                        value=message,
                        key=f'floor_{floor["floor_number"]}',
                        callback=delivery_callback)
                producer.flush(0.5)
                floor_features.refreshValues()

    except KeyboardInterrupt:
        exit(0)
    except Exception:
        exit(103)
```

Listing 3.7: Occupation calculator engine

3.3.5 Real time map



The system's real-time map module enables a live display of the obtained outcomes. The system is web-based and the map has been defined using the JavaScript library Leaflet.js (described in section 2.3.2). The map defined is the non-geographical category, according to the definition on Leaflet.js. The map is non-geographical because, the library in question is used, usually, to display and customize real maps that take advantage of those provided by the major providers of this type of content such as Google Maps, OpenStreetMap etc. In the case of this project, the potential of this library has been exploited to manage an indoor plant of a building. The map on which you are based will then be the loaded building plant, which will then present a custom reference system, different from the classical latitude and longitude used in the usual applications of this library. This reference system in practice is defined by a system of cartesian axes that originate from the lower left side of the building plant. Each pixel of the image will be mapped as coordinates in the order and the abscissa. This allows you to easily view the results calculated by the other modules. The results are the calculated

individual positions and occupation levels of the office areas and rooms. These findings will be displayed as in Fig. 3.28, where you can see that the individual places are represented by the blue icons on the map. If you select one of these icons, the timing and person with whom the location is related will be shown. The occupation is represented by the various colored zones, which become more intense as the number of individuals identified rises. At the upper right of the map, it also displays the precise number of people found out. This system module is composed of many components and is directly connected to the broker Kafka, which is the source of the displayed data. This module's central point is the Express.js server. This manages all of the static resources that will be made available to users, when they use the service, and communicates with the two Javascript modules which aim to capture the information provided by the Kafka broker. The relationships between the various components are depicted in a general way in Fig. 3.26.

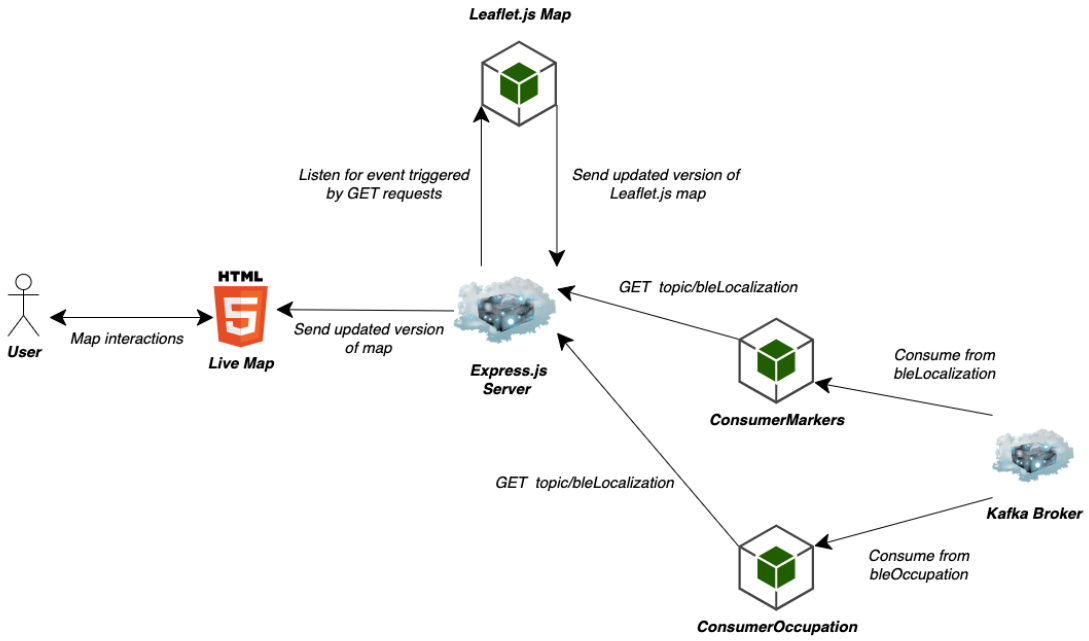


Figure 3.26: Architectural overview of real-time map

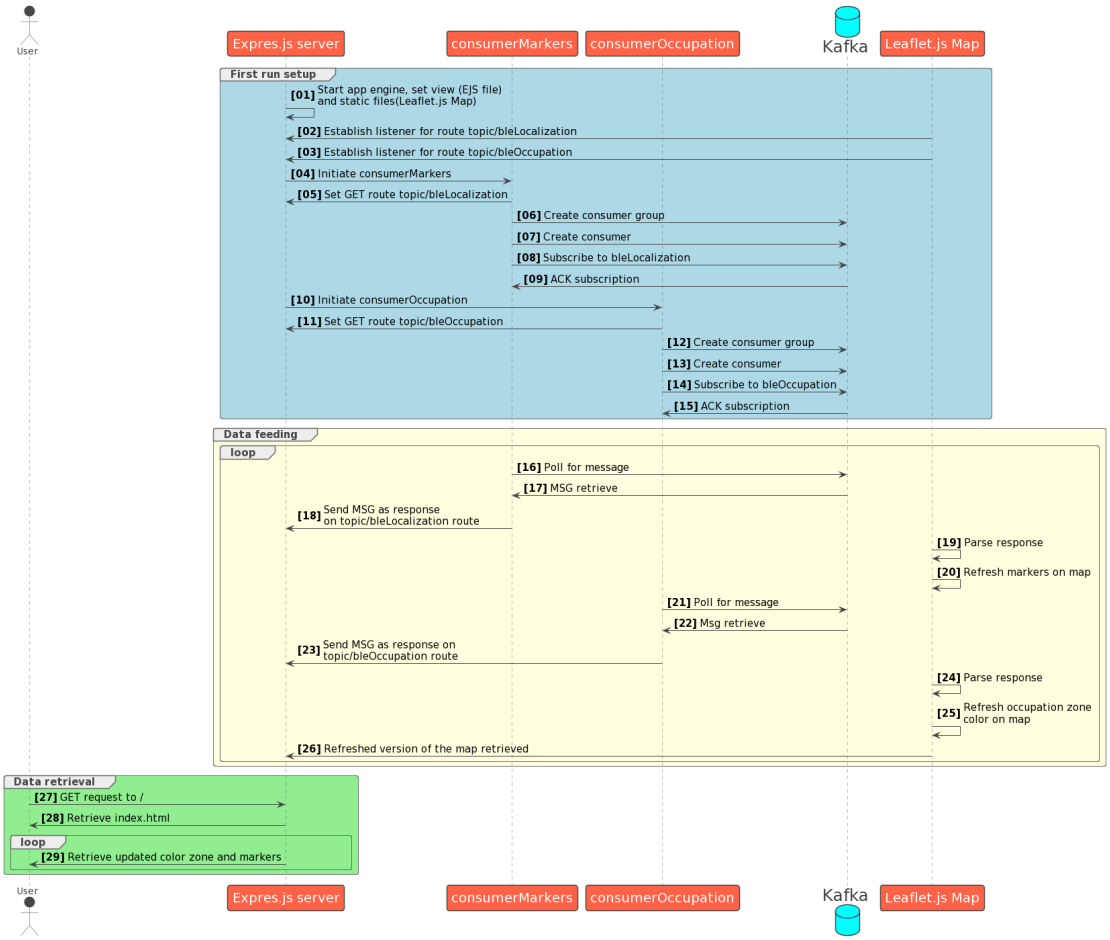


Figure 3.27: Process diagram of real-time map

Two stages—the initial setting and the functioning after first boot—can be seen if you pay close attention to Fig. 3.27. When the system is begun, it will run indefinitely, independent of the rest of the system, even if the other modules fail. The following stages are taken during the first setup:

- At the beginning, the engine of the Express.js app, which works as core server in this module, is started. You must define links to static files and routes of the server. In terms of static files, they merely have a file that serve as display engine and a Javascript file pertaining to the map itself.The first, is EJS file. EJS is a JavaScript templating engine that allows you to insert JavaScript code within HTML templates to generate dynamic information on the server side. It is commonly used to render dynamic data and to create reusable components. So this will format the interface and provide a dynamically enabled interface. The second is the JavaScript file that is written using the

Leaflet.js library, which allows you to define and customize the interactive map as defined previously. With regard to the routes what is defined are the one that redirect to the main page (the root '/'), and also the two routes, i.e. 'topic/bleLocalization' and 'topik/bleOccupation', to which listeners are associated, fundamental for the operation of the system, whose role will be described in the next steps.

- In the subsequent stage, Kafka consumers defined in the consumerMarkers.js and consumerOccupation.js files using KafkaJs, a package that allows you to manage Kafka connections using JavaScript (described in Section 2.3.3), are initialized. The processes show that the first phases of these components include, above all, the linkage of a function with the matching GET route. This recursive code, given in the 3.8 sample, creates a route that sends periodic Server-Sent Events (SSE) to the client every 5 seconds until the session is terminated. The events are sent in text format and can be processed on the client side with JavaScript to handle real-time server updates. SSE is a web technology that allows servers to provide real-time updates to clients across a long-lived HTTP connection, providing real-time communication without the need for frequent client requests.

```
router.get('ROUTE',jsonParser,  function (req,res){
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Access-Control-Allow-Origin', '*');
    res.setHeader('Connection', 'keep-alive');
    res.flushHeaders();
    let interValID = setInterval(() => {
        res.write('data: '+body+'\n\n');
        clearInterval(interValID);
        res.end(); // terminates SSE session
        return;
        }, 5000);

});
```

Listing 3.8: Function related to GET route

At this point the system is settled and functioning, it will remain static until it receives information from Kafka. In short, loading the web page will only load the map, with no marker and with the zones set to a zero occupation level. At the same time, the two consumers will continue to be in a poll status. When a message of interest is published,, respectively, in the topic bleLocalization, for markers, and bleOccupation for occupation levels, with intervals of 5 seconds, this will be transformed into an action that will modify the map appropriately. Updated map that is instantly returned to the user. If a bleLocalization message arrives, a marker

is generated or moved, in the latter case the color level and the number relating to the occupation of the affected areas will be increased/reduced. The instant trigger of these actions is made possible by using Eventsource in the index.js related JavaScript code written using Leaflet.js library. In JavaScript, the EventSource interface is used to connect to a server that supports SSE. It enables the client to receive real-time updates or events from the server without having to poll the server continuously. It makes it easier to create real-time applications. The system in any case will remain in this state of infinite loop in the background, ensuring that every user access has an updated version of the environment under analysis.

Figure 3.28: Views on real-time map

3.4 Exploratory data analysis

This section will outline the methods followed for a consistent system configuration, with a focus on the analysis of the tag signal and how the filtering was improved using Kalman filter. The system, as described in the preceding section, is extensively parametrized. Other parameters unrelated to the RSSI signal and the linked filtering have been described in previous sections. For them, a on-field calibration based on real-world proof, in relation to the development scenario, was performed. This was because statistically assessing every component of the system would have been impossible because the development scenario involving fading, shadowing, and other phenomena would have been too difficult to model. The selection of these parameters was primarily based on practical tuning, but on the other hand, the signal processing was given greater consideration because it affects every other system behavior. Regarding the raw RSSI signal of the tags, a methodical analysis was performed in order to understand its evolution and time behavior, and thus to discover an appropriate filtering solution. Because the development scenario includes numerous tags and a quite large number of gateways, a smaller context will be examined as an example to demonstrate the methods used. More specifically, a single tag in relation to four gateways for signal behavior analysis and a tag's behavior in relation to a single gateway for Kalman filter parameters analysis applied to the RSSI signal. The above context is shown in Fig. 3.29, and the corresponding key information is presented in Table 3.3.

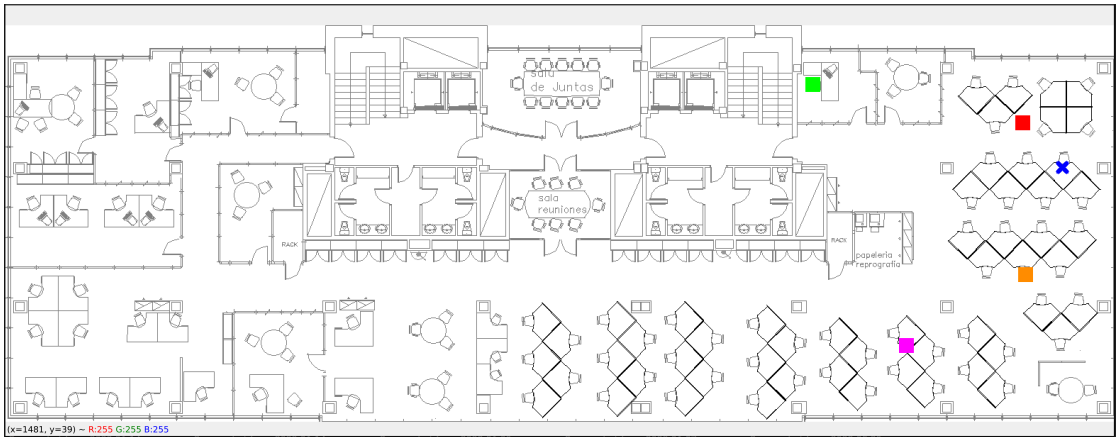


Figure 3.29: EDA context

Gateway name	Color	Figure
ap_194	Red	3.30
ap_196	Green	3.31
ap_199	Orange	3.32
ap_206	Magenta	3.33

Table 3.3: Gateway reference information

The signal received from the tag by each gateway is rather unsteady, as seen in Figs 3.30 to 3.33. It should be noted that the signal associated with this tag pertains to a static context, i.e., the tag is left standing in a known location (the blue cross in Fig. 3.29), free to broadcast to any gateway in its line of sight. As you can see, by left-side images, the signal varies by 15 dBm points and also drastically in the majority of the data being analyzed. Even the KDE distribution [6] , as shown by the figures to the right of each picture, exhibits very irregular behavior, as evidenced by the presence of many peaks, a non-uniform distribution, and a rather broad range of values. This is due to physical impediments, interferences, or changes in the environment. As a result, a mathematical model capable of filtering the signal while also being flexible and reactive is required. The Kalman filter, whose algorithm is explained in Section 3.3.3, is the natural choice. To go deeper in the motivations of the choices made regarding the parameters of the Kalman filter only a signal is evaluated. The one chosen is the received one by the bluetooth gateway ap_194.

[6]Kernel density estimate: Smoothly estimate the shape of a distribution by placing smooth curves (kernels) on data points and summing them up.

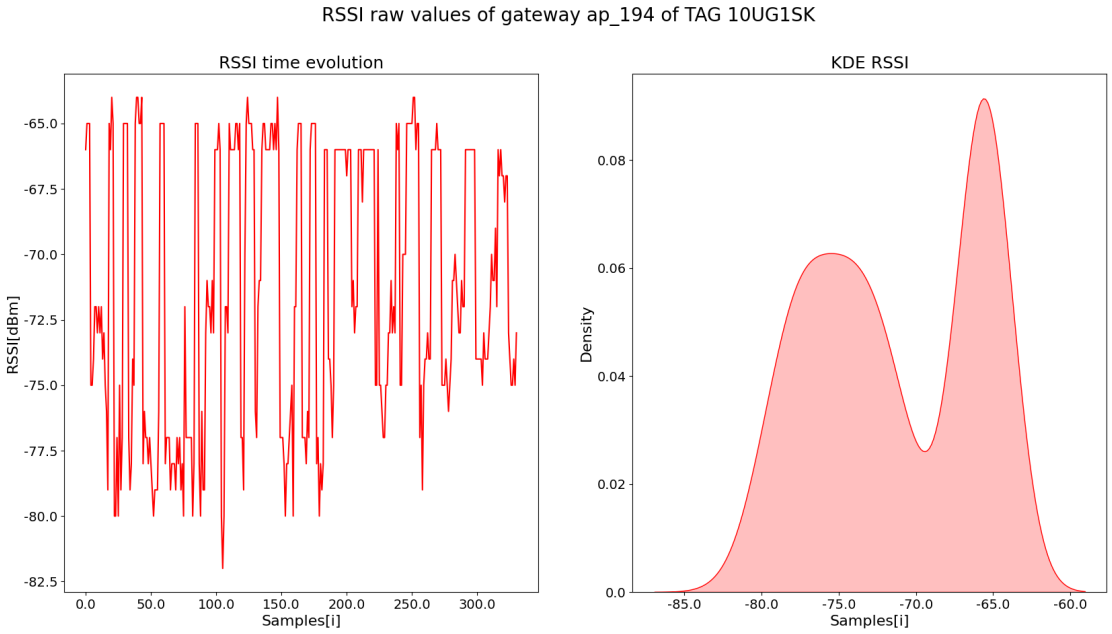


Figure 3.30: Signal evolution of ap_194

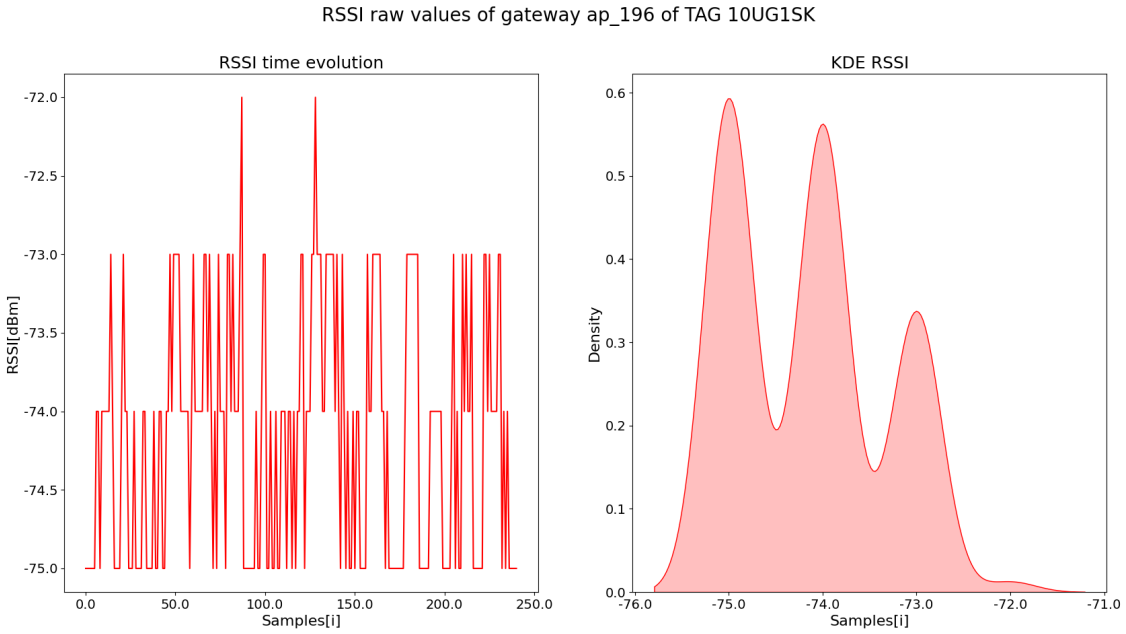


Figure 3.31: Signal evolution of ap_196

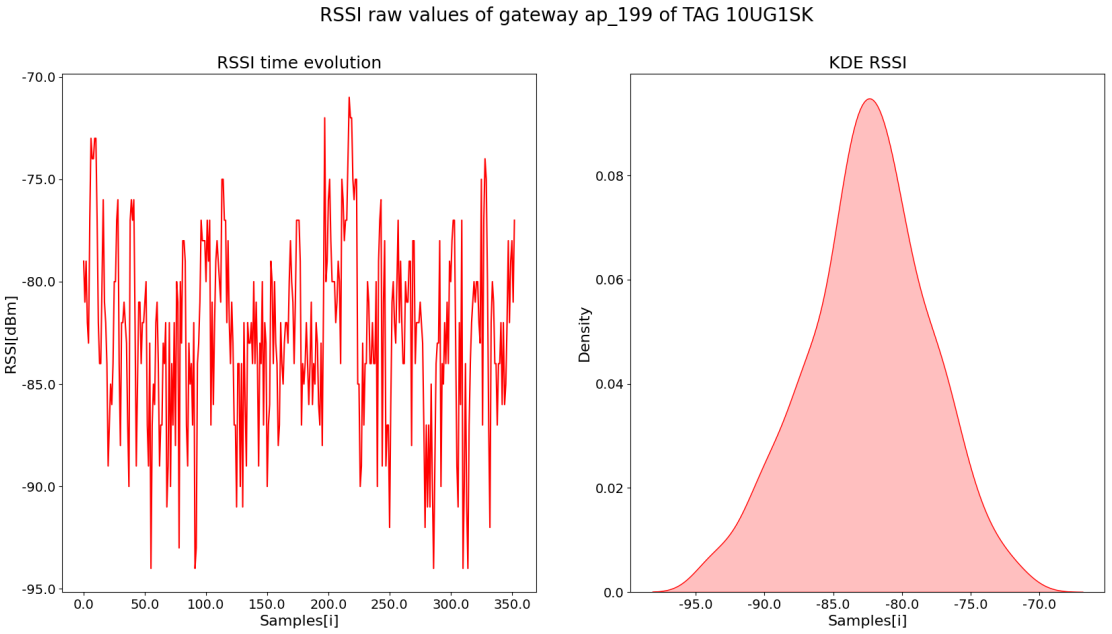


Figure 3.32: Signal evolution of ap_199

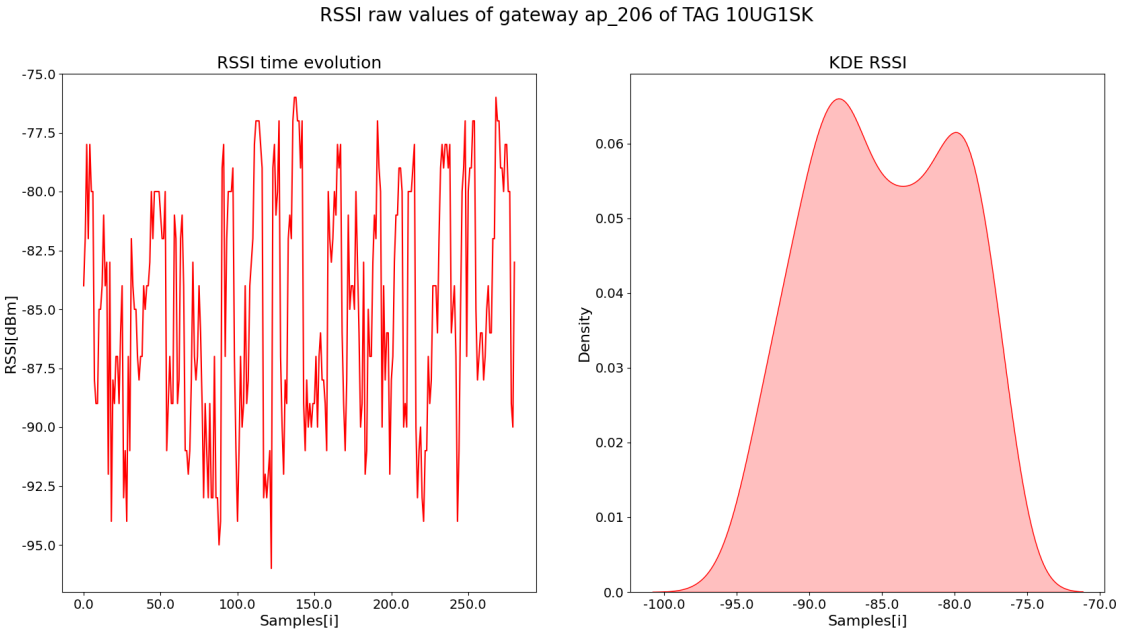


Figure 3.33: Signal evolution of ap_206

Considering the simplifications and assumptions to linearize filter modeling, described in section 3.3.3, the only two parameters to be set to change filter behavior are Q, related to process noise, and R, linked to measurement noise. Table 3.4 describes the various tests that are taken as examples to guide the final choice made, i.e. the one described by the test number 7 of the table. Before analysing the tests carried out, it is necessary to analyze the meaning of these two parameters. For the explanation, the values that have been defined as optimal for this system will be taken into account.

When the process noise in a Kalman filter for RSSI filtering is increased, it signifies that additional uncertainty or variability is introduced into the presumed signal propagation model. Process noise in RSSI filtering represents errors or variations in RSSI readings that are not accounted for by the filter's prediction model. By increasing the process noise, you are effectively implying that you have less confidence in the model's precision and reliability. The increase in process noise indicates that you believe signal propagation characteristics such signal attenuation, multipath fading, or interference are less predictable or more likely to alter over time. It means that variations in RSSI readings may be influenced by elements not explicitly considered or modeled by the filter. The Kalman filter gets increasingly responsive to measured RSSI values as the process noise is increased. It gives more weight to current measurements than to forecasts from the inner model. This greater responsiveness enables the filter to respond faster to changes in the signal propagation environment, such as abrupt interference or signal strength shifts. A variation of 0.1 in an RSSI tracking system, generally, is considered to be high. A fluctuation of 0.1 dBm may be perceived excessive if the tracking system is designed for fine-grained precision or is utilized in applications where minor changes in signal strength are crucial, such as localization or proximity-based systems. In such instances, it is critical to reduce noise, calibrate the system, and assure high-quality signal measurements in order to achieve consistent tracking performance. If the system is operating in a high-noise environment or the needed precision is not excessive, a difference of 0.1 dBm may be reasonable and deemed within an acceptable range. This is because there may be many types of interference and signal attenuation owing to ambient conditions in a tracking system that uses RSSI data for approximate localisation. In this case, the RSSI tracking system's primary purpose is to convey a general sense of location or closeness rather than pinpoint accuracy. Small changes in RSSI values are unlikely to have a substantial impact on overall tracking performance. To adjust for 0.1 dB changes as a natural component of system operation, techniques such as filtering or averaging could be used. However, it is crucial to highlight that overly raising process noise can present the risk of overreacting to measurement noise or outliers. Over time, the filter's ability to reliably track the true signal strength may deteriorate.

By purposefully adding extra uncertainty or inaccuracy to the RSSI measurements the Kalman filter uses, the measurement noise in the filter can be increased. This can be done to test the filter's robustness, evaluate its performance in challenging circumstances, account for unknown factors influencing measurements, and assess worst-case scenarios. Given a variation range of 20 units more or less (82 - 62), as seen in Fig. 3.30 in a measurement noise value of 5 could indicates a large fraction of the system's dynamics. It accounts for 25% of the overall range, indicating that the measurements are subject to significant ambiguity or error. It is normal for measurements to have a higher level of uncertainty or variability in an unstable system with considerable signal changes.Setting the measurement noise to 5 acknowledges and accounts for the system's inherent instability and fluctuation. A larger measurement noise value can assist the Kalman filter in adapting to the system's rapid changes and providing a more responsive assessment of the state. It enables the filter to be less reliant on measurements and more impacted by the expected state, which can be useful in dealing with signal instability and dramatic changes. It is crucial to remember, however, that increasing the measurement noise generally results in lower estimating accuracy and a smoother, but potentially less accurate, estimation of the genuine RSSI values. The filter's responsiveness may also be reduced, resulting in slower updates and a more filtered output.

Given these broad considerations, it is reasonable to presume that the values selected are adequate. But, to understand why these were picked, look at the visuals connected with the various tests. The R value was fixed in the first three, thus create variation of Q. If you select a lower number, at 0.1 as you can see, the signal is over-filtered and may provide an issue in the event of movement since the convergence to the new signal may be too delayed (Figs. 3.34 and 3.35). A number larger than 0.1 implies that the system will react excessively to measurement noise or outliers (Fig. 3.36).

In the second set of tests, on the contrary, Q is fixed at 0.1 and R is varied. In this case, taking values below 5 means that you give too much confidence to the measurements, and therefore the filtration will be very reduced, making this passage almost useless (Figs. 3.37 and 3.38). In the case of higher values, however, the signal is simplified too much and can cause problems in dynamic contexts (Fig. 3.39).

As previously stated, the test 7 illustrated in Fig. 3.40, with Q=0.1 and R=5, was chosen as the optimum combination to track the system's evolution. This option allows you to a compromise of precision for practical limits.

Test N.	Q	R	Figure N.
1	0.01	5	3.34
2	0.05	5	3.35
3	0.5	5	3.36
4	0.1	0.5	3.37
5	0.1	1	3.38
6	0.1	10	3.39
7	0.1	5	3.40

Table 3.4: Kalman filter use cases

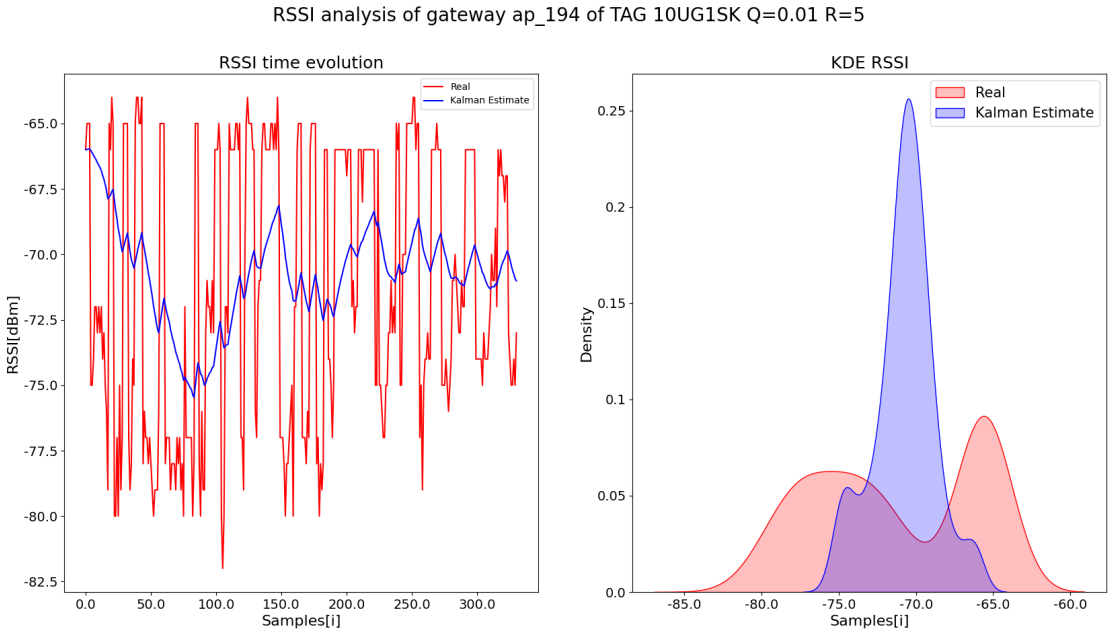


Figure 3.34: Signal evolution with Q=0.01 and R=5

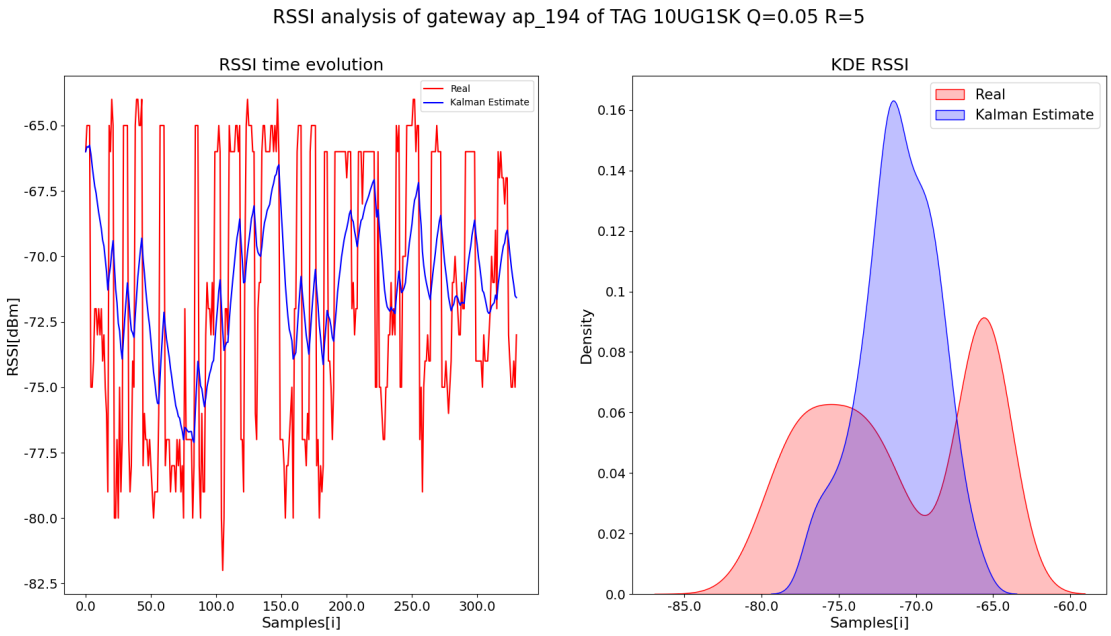


Figure 3.35: Signal evolution with Q=0.05 and R=5

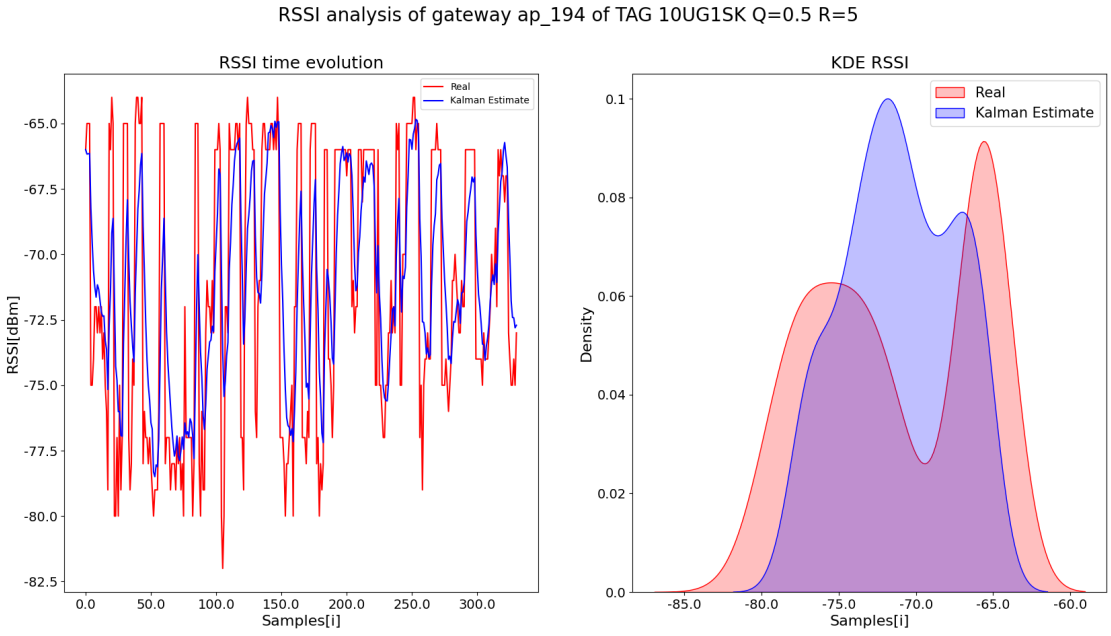


Figure 3.36: Signal evolution with Q=0.5 and R=5

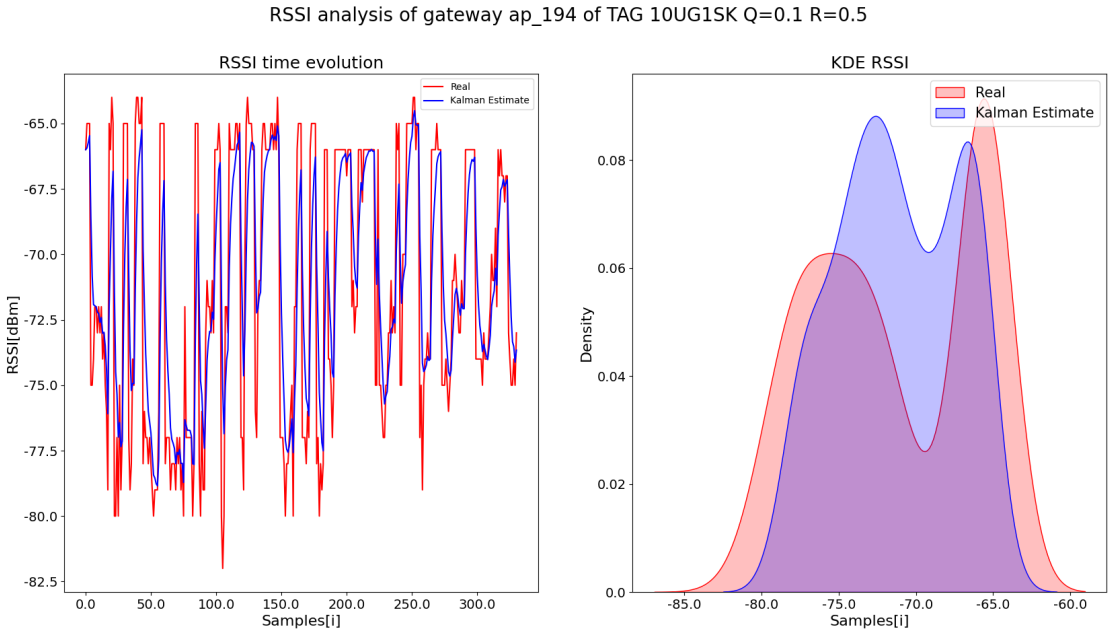


Figure 3.37: Signal evolution with Q=0.1 and R=0.5

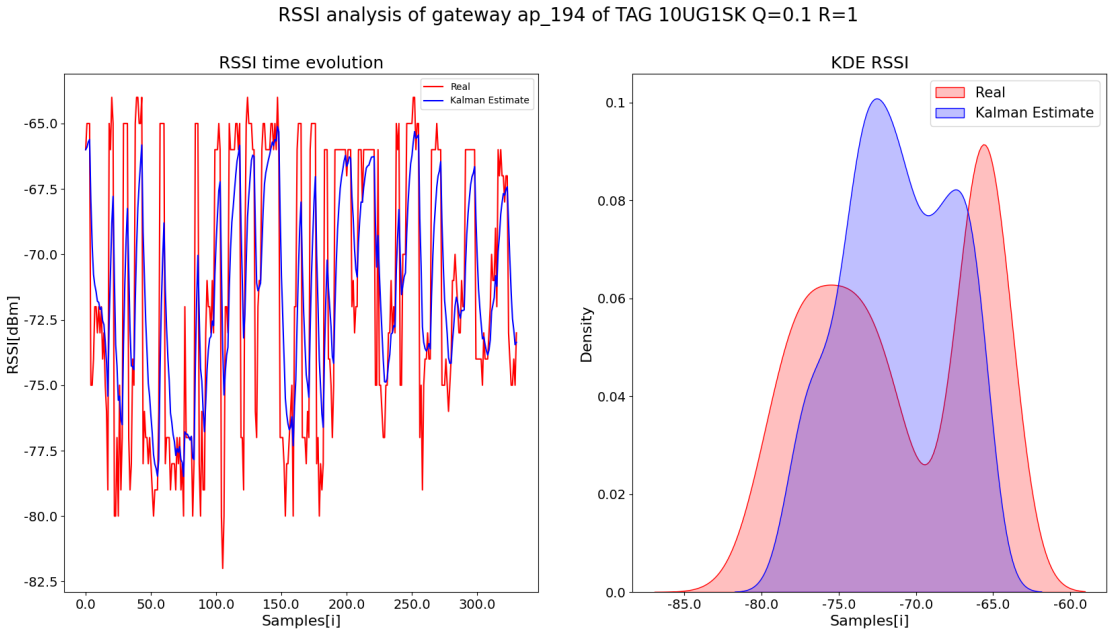


Figure 3.38: Signal evolution with Q=0.1 and R=1

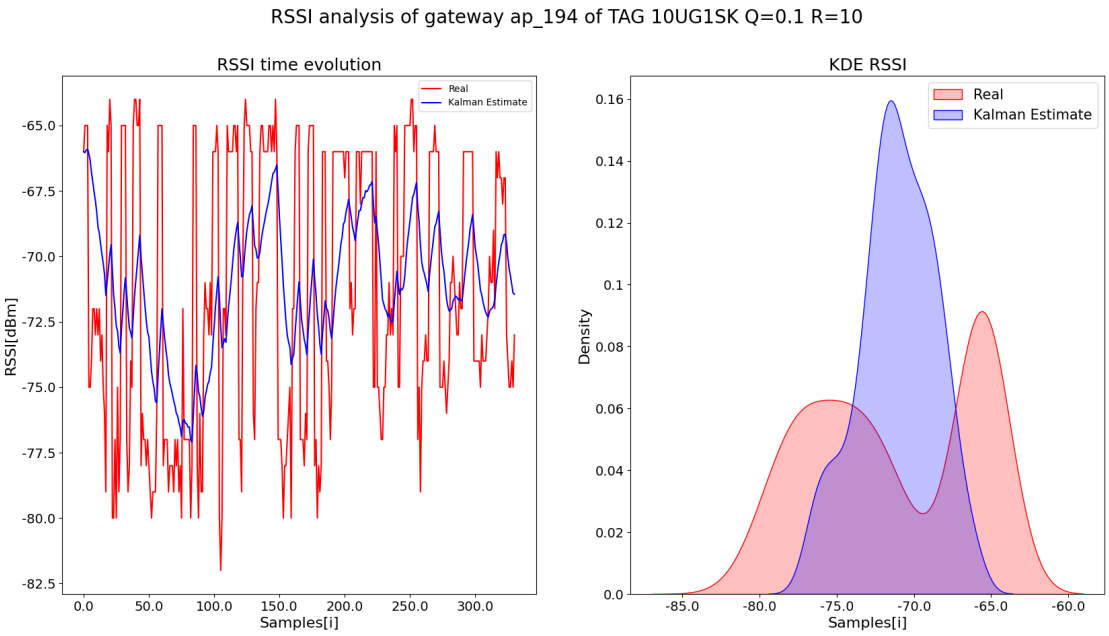


Figure 3.39: Signal evolution with Q=0.1 and R=10

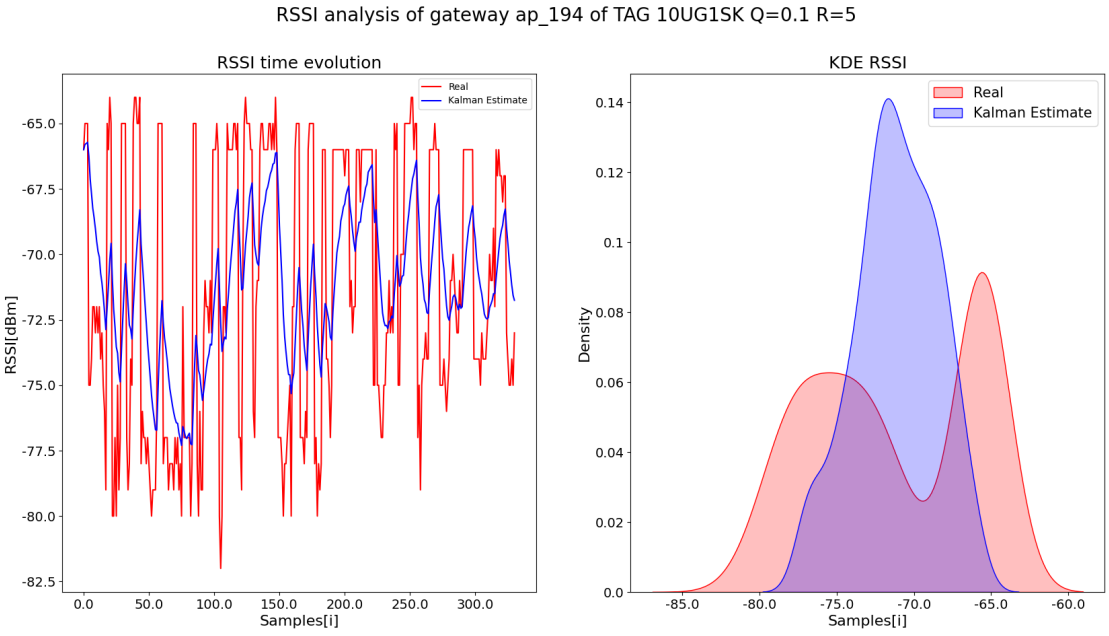


Figure 3.40: Signal evolution with Q=0.1 and R=5

Chapter 4

Results and conclusions

4.1 Results

This part will review all of the project's findings, both numerical and the ensuing visual interface to support the user experience.

4.1.1 Localization engine results

The localization engine as described in section 3.3.3 as a result gives an always-running system capable of determining positions from RSSI raw data in a few milliseconds. The end result is a system that, in line with the project's goal, can calculate the positions statically, e.g., whether the tag stays in place. Given the non-critical nature of the system's application, the specifications actually do not call for a fine-grained location but rather a system that provides basic directions on occupation of the building. Given the physical restrictions imposed by the stability of the tag's signal, dynamic tracking is possible, but the computations' convergence is not immediate. Fig. 4.1 depicts a day's worth of activity of the system. As you can see, despite the noisy nature of the tag signals, the estimated positions generally have a good quality and do not vary significantly over the day. The average accuracy level, as well as the minimum and maximum values, are shown in Fig. 4.2. Generally speaking, it should be noted that the system can accurately determine a position between 1 and 5 meters away. The analysis also includes information on tags that moved, for which the correctness of the calculation of the positions cannot be precisely verified because there are no knowledge about their real position. Fig. 4.4 depicts the tracking of a single tag, which demonstrates the system's convergent behavior. In fact, the first unstable position is quickly rectified and maintained, as seen in Fig. 4.3. This phenomenon is caused by the application of the Kalman filter, which, despite being initially unconditioned, quickly converges to the right value.

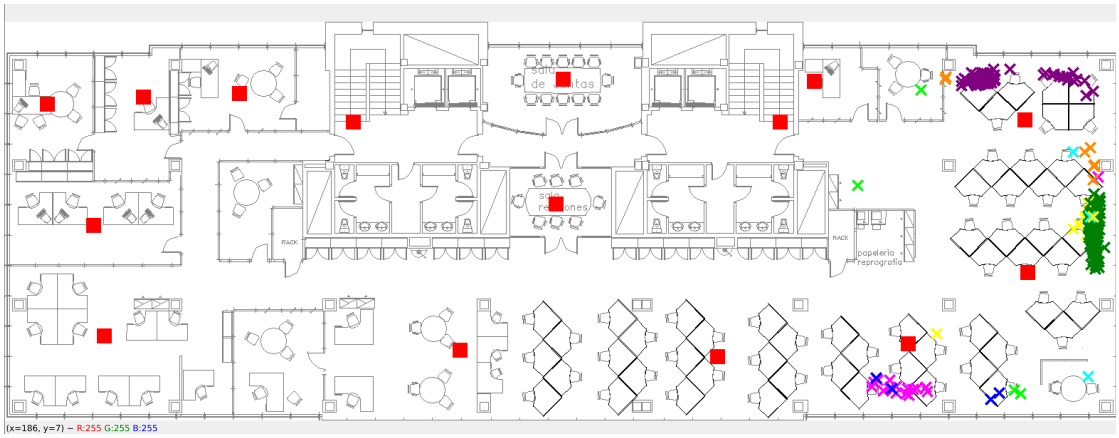


Figure 4.1: One day position tracking

```
GREEN_CROSS ==> [redacted]-10UG1SK
CYAN_CROSS ==> [redacted]-10UN2JY
MAGENTA_CROSS ==> Test 5 -10Uy1pe
BLUE_CROSS ==> Test 6-10UP1r2
YELLOW_CROSS ==> Test 3-10UN2aG
PURPLE_CROSS ==> [redacted]-10UF2GT
ORANGE_CROSS ==> [redacted]-10UZ1Sc
DARK_GREEN_CROSS ==> Test 1-10UG2JS
10Uy1pe
ID:10Uy1pe, MAX:10.918323022670208m, MIN:2.043415654039681m, MEAN:2.8350734111633384m
10UG2JS
ID:10UG2JS, MAX:5.0000402855010675m, MIN:1.9048299272531424m, MEAN:3.3998338203889142m
10UP1r2
ID:10UP1r2, MAX:6.796722346762676m, MIN:1.5218512663663586m, MEAN:2.2272533147500515m
10UF2GT
ID:10UF2GT, MAX:5.4842601032657905m, MIN:1.5282203532049012m, MEAN:4.212746719498804m
```

Figure 4.2: Error overview of the trace

```
1==> (1537, 217) ACCURACY:10.918323022670208
2==> (1267, 521) ACCURACY:2.5720984546139043
3==> (1253, 520) ACCURACY:2.769884170279095
4==> (1244, 502) ACCURACY:2.510520515877261
5==> (1222, 505) ACCURACY:3.135398182398463
6==> (1227, 502) ACCURACY:2.944396105918666
7==> (1219, 511) ACCURACY:3.3311650473864653
8==> (1235, 517) ACCURACY:3.066663534155022
9==> (1253, 511) ACCURACY:2.5297913186041154
10==> (1248, 516) ACCURACY:2.7603217135598164
11==> (1248, 515) ACCURACY:2.7345925851348447
12==> (1245, 512) ACCURACY:2.722727071028554
13==> (1269, 514) ACCURACY:2.336166693914797
14==> (1271, 523) ACCURACY:2.5804766476633216
15==> (1297, 518) ACCURACY:2.2360859937235684
16==> (1282, 514) ACCURACY:2.184079717956506
17==> (1295, 512) ACCURACY:2.043415654039681
18==> (1277, 515) ACCURACY:2.265097704244769
ID:10Uy1pe, MAX:10.918323022670208m, MIN:2.043415654039681m, MEAN:2.8350734111633384m
```

Figure 4.3: Temporal improvements of position

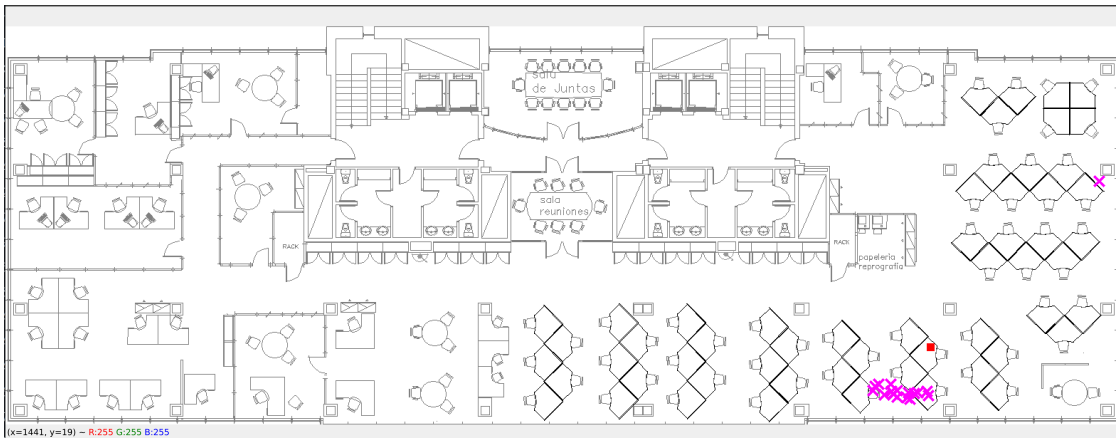


Figure 4.4: Single tag position track

4.1.2 Occupation engine results

The occupation calculator, as detailed in section 3.3.4, allows you to compute building occupation levels very accurately and fast using simple geometric calculations. From an organizational standpoint, the module is autonomous from the rest of the system, and its operational precision is extremely high. It should be mentioned that it operates on the basis of data supplied by the localization engine. As a result, if the aforementioned module functions well, the overall accuracy of the occupation computation achieves high levels.

4.1.3 User Interface

Another significant final achievement in this project is the resultant user interface for monitoring real-time data, which will assist facility management staff in preserving the quality of the environment, as described in Section 3.3.5. The final result is a web app that allows you to see the location and occupation of the building in real time. The user can zoom to distinguish the different markers, click on the markers to see the information (Fig. 4.6), select a specific floor or data layer (Fig. 4.5). In fact, you can choose to show only the layer that highlight the occupation (Fig.4.8), or that one layer which shows only the markers (Fig. 4.7), and obviously both (Fig. 3.28).

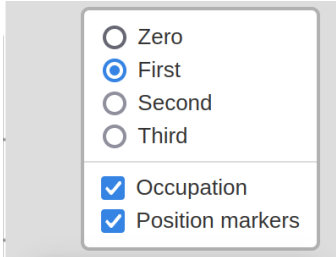


Figure 4.5: Layer control panel

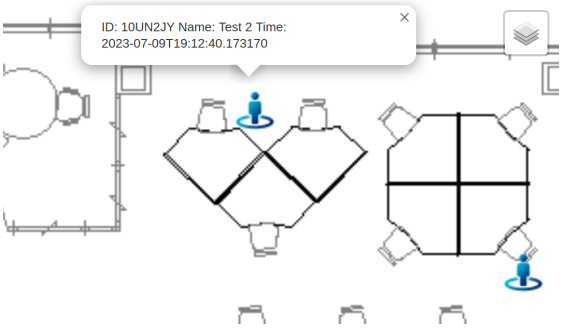


Figure 4.6: Marker information showed

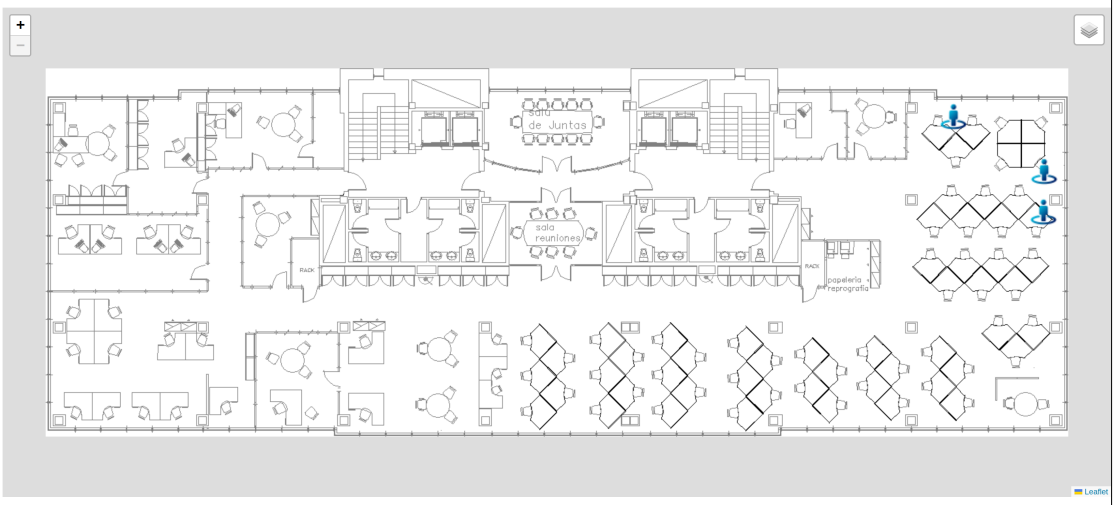


Figure 4.7: Only markers view

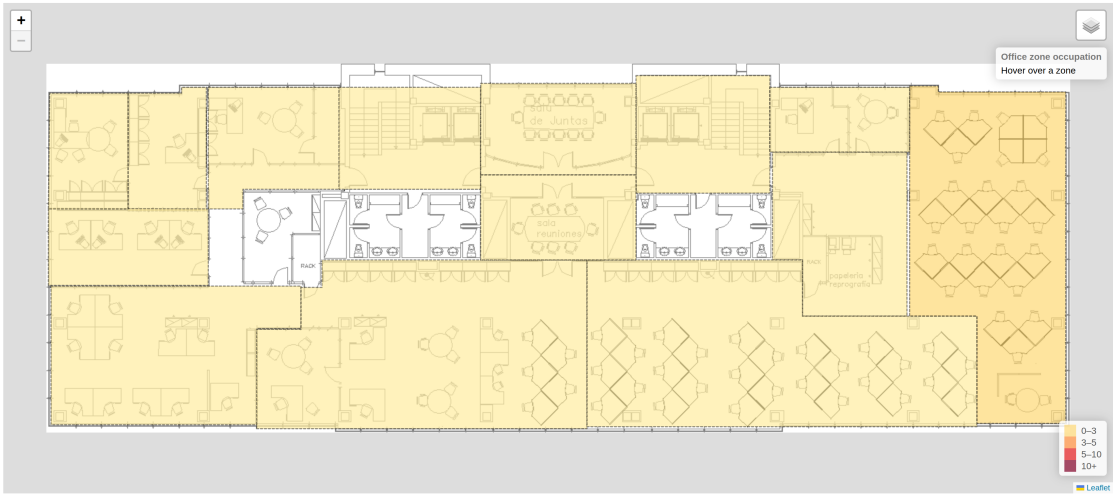


Figure 4.8: Only occupation view

4.2 Conclusions

To conclude the analysis of this project, it is good to reanalyze the goals that were defined at the beginning of this paper in Section 1.5. More precisely:

- After careful analysis of the available hardware, it was possible to take action on the calibration of the various components. The field trials to understand the correct settings, of both the stations and the tags, have been crucial in

order to achieve reasonable results. In the specific scenario considered, the levels of transmission power of tags and the filter level of received power of gateways were analyzed. In addition, a reasonable timing of transmission has been found, in order to proceed to the following steps of localization. Not taking into account these factors would lead to a poor accuracy given by wrong signal setting, and in the latter case of timing to a bad convergence of the trilateration.

- A real-time localization application needs a solid backend to function. First off, the flexibility and responsiveness of the lightweight HTTP API used for data sinking allowed for the millisecond-accurate data redirecting from gateways. To meet the necessary delivery and time constraints, it was essential to use stream processing to connect the computing core. The entire architecture is the result of multiple revisions and efforts to discover an appropriate structure for developing a non-monolithic tool. Even on less capable hardware (such as an edge solution), this structure, helped by the virtualization offered by the containers, allows a high level of scalability and portability. Additionally, opting to use the CI/CD approach for micro-service-oriented development enabled for the creation of a system that was always up to date, simple to change, and error-free. Each module was created as a distinct connected entity.The components of the architecture were each independently created and are formally autonomous. They communicate effectively, in part because of Docker's networking capability but also because of Apache Kafka's great potential;

- The localization system has been completely parametrized in order to allow for varying QoS levels and application adaptation on different deployment scenarios. The parameters for the system strongly depend on the circumstance in which it is being utilized, as was mentioned when examining the system. This is a result of the BLE technology's physical constraints, which were utilised in the creation of this system. To give you complete control over the system customisation, a highly extensive parametrification was used (Section 3.3.3). These adjustments cover everything from specifications for tag transmission, Kalman filter settings, needed spatial and temporal accuracy requirements, through consumption management and publication on the Kafka broker criteria;

- The developed algorithm effectively and almost instantly manages the considerable amount of data gathered and forwarded by the HTTP server. The objective has always been to reduce the amount of time required to perform the position calculation after receiving the raw RSSI data from a tag. As

previously stated, a first bottleneck was removed by improving the data collection server. The second time barrier has been broken by making the most of Kafka's capabilities, utilizing topic and partition division, as well as the outstanding reactivity provided by its request pull behavior. The algorithm itself, which rapidly calculates using a direct and quick approach, minimizes unnecessary and redundant calculations, is the latest improvement. In this approach, a valid position can be calculated quickly and with fair precision thanks to the Kalman filter's operation. Additionally, you can observe the location associated with a tag in a matter of seconds by constantly combining the strength of Kafka with the responsiveness offered by the actual real time mapping module;

- It is not sufficient on its own to have strong results and a supported system that measures metrics in a reliable way. For this type of monitoring program, having a distinct and well-defined user interface is crucial. This unlock the possibility to examine the office system in real-time comprehensive viewpoint.

It should be noted that possible performance limitations in terms of timing or accuracy are closely related to the technology used. The devices used are of entry-level quality and therefore do not provide the best possible performance. The aim of this project, however, was not to create a performance solution at the top, but rather to demonstrate and analyze the feasibility of such a system even with non-advanced hardware. The current system from a software point of view is barely portable. To adjust it, you only need to reason in terms of setting with regard to the above mentioned parameters. With hardware improvements, i.e. more performing transmitters, better results could be achieved, both in terms of speed in returning positions, as well as accuracy, and adjustment to change of location. In addition, although the entire system developed refers to a single floor of the building, the structure of the system has been thought to be scaled to more elaborate scenarios.

In conclusion, all of the planned milestones were met in the manner intended. This allows for the return of a comprehensive solution that allows for an office environment monitoring system, as well as assistance in improving working environment space management.

As stated before, the current solution rather than a finished product is a proof of concept of a possible applicable solution, so, may be improvable in its current condition, and the next and last part will outline possible future enhancements that can be done.

4.2.1 Future works

The developed solution is still in its development because there is still much space for improvements.The first natural progression is surely the upgrade of part of

the hardware used. The receivers utilized are adequate, however the transmitters provide a highly unstable signal that is difficult to track in the case of very quick movements. There are more efficient options available on the market. One option is to use cellphones' Bluetooth Low Energy (BLE) transmission. In advertisement mode, the smart phone can be used as a tag. Both Android [73] and iOS [74] support the use of BLE low energy in the background, ensuring low battery use. The problem in this situation is more than technical; it is basically "human" because it deals with concerns linked to convincing individuals to download an app and, in some ways, handling personal data. You must ensure that the individual utilizing the app is someone known to the company. However, by providing lower latency and higher transmission quality, this sort of solution may enable further applications, such as automating the work badge check for entering or departing from work using a position-type solution.

All of these operations should be defined in as lightweight and portable a manner as possible. By keeping the amount of memory to a minimum, all of these procedures might be performed at the edge, decreasing the amount of data bulk handled by the server. This would allow the solution to be extended to cover more instances than it now does.This could be accomplished by investigating new communication bus technologies. NATS.io [75] might be used to make the system lighter, or Apache Pulsar [76] could be used to include serverless computing functionality directly into the context of service communication (e.g. on-the-fly filtering).

The user interface is the portion of the system that needs to be completely updated, because the one established is only a showcase that should be integrated into a more comprehensive graphic solution, preferably with an authentication system to protect the information.

The final value you may include is data saving with the integration of a database. At the time, this is not established because the data is handled as ephemeral data, with a life of a few seconds, to prevent overloading corporate systems, without a solid plan for optimally saving all or part of the data. This could open up data analytics scenarios providing a more detailed historical view of this type of monitoring.

All of these long-term objectives were established with future integration of two SATEC wider products in mind, namely SATEC Health.Track/Monitor and SATEC Location System/Monitor (SALOS). As noted above, all the methods currently specified in this project will be analyzed and then incorporated while also looking at potential upgrades that might enhance performance.

Appendix A

Kafka broker settings

```
---
version: '3.6'
volumes:
  zookeeper-data:
    driver: local
  zookeeper-log:
    driver: local
  kafka-data:
    driver: local
networks:
  kafka_net:
    name: kafka_network

services:
# ==============================
# ========KAFKA ENVIRONMENT=========
# ==============================
  zookeeper:
    image: confluentinc/cp-zookeeper
    container_name: zookeeper
    restart: always
    networks:
      - kafka_net
    volumes:
      - zookeeper-data:/var/lib/zookeeper/data:Z
      - zookeeper-log:/var/lib/zookeeper/log:Z
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
  broker:
    image: confluentinc/cp-kafka:latest
    container_name: broker
    restart: always
    networks:
      - kafka_net

    volumes:
      - kafka-data:/var/lib/kafka/data:Z
    ports:
      - "9092:9092"
      - "9101:9101"
```

```
42      expose:
43        - "9093"
44      depends_on:
45        - zookeeper
46      environment:
47        KAFKA_BROKER_ID: 1
48        KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
49        KAFKA_LISTENERS: INSIDE_VIEW://0.0.0.0:9093,OUTSIDE_VIEW://0.0.0.0:9092
50        KAFKA_ADVERTISED_LISTENERS: INSIDE_VIEW://broker:9093,OUTSIDE_VIEW://
      COMPANY_HOST:9092
51        KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INSIDE_VIEW:PLAINTEXT,OUTSIDE_VIEW:
      PLAINTEXT
52        KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE_VIEW
53        KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
54        KAFKA_OFFSETS_RETENTION_MINUTES: 5
55        KAFKA_OFFSETS_RETENTION_CHECK_INTERVAL_MS: 60000
56        KAFKA_LOG_RETENTION_MS: 30000
57        KAFKA_LOG_RETENTION_CHECK_INTERVAL_MS: 5000
58        KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
59        KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
60        KAFKA_CONFLUENT_BALANCER_TOPIC_REPLICATION_FACTOR: 1
61        KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
62        KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
63        KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
64        KAFKA_JMX_PORT: 9101
65        KAFKA_JMX_HOSTNAME: localhost
66        KAFKA_CONFLUENT_SCHEMA_REGISTRY_URL: http://schema-registry:8081
67        CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: broker:9093
68        CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 1
69        CONFLUENT_METRICS_ENABLE: 'true'
70        CONFLUENT_SUPPORT_CUSTOMER_ID: 'anonymous'
71        KAFKA_AUTHORIZER_CLASS_NAME: 'kafka.security.authorizer.AclAuthorizer'
72        KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: 'true'
73        KAFKA_CONFLUENT_SUPPORT_METRICS_ENABLE: 'false'
74      links:
75        - zookeeper
76    init-kafka:
77      image: confluentinc/cp-kafka:latest
78      container_name: initial_bootstrap
79      networks:
80        - kafka_net
81      depends_on:
82        - broker
83      entrypoint: [ '/bin/sh', '-c' ]
84      command: |
85        "
86        # blocks until kafka is reachable
87        sleep 15
88        echo -e 'SHOW PRE-EXISTENT TOPICS'
89        kafka-topics --bootstrap-server broker:9093 --list
90
91        echo -e 'Creating kafka topics'
92        sleep 5
93        kafka-topics --bootstrap-server broker:9093 --create --if-not-exists --topic
      bleRSSI --replication-factor 1 --partitions 5
94        kafka-topics --bootstrap-server broker:9093 --create --if-not-exists --topic
      bleRSSISupport --replication-factor 1 --partitions 5
95        kafka-topics --bootstrap-server broker:9093 --create --if-not-exists --topic
      bleLocalization --replication-factor 1 --partitions 1
96        kafka-topics --bootstrap-server broker:9093 --create --if-not-exists --topic
      bleOccupation --replication-factor 1 --partitions 1
```

```
97
98        echo -e 'Configuring kafka topics'
99        sleep 5
100       kafka-configs --bootstrap-server broker:9093 --entity-type topics --entity-
      name bleRSSI --alter --add-config retention.ms=10000
101       kafka-configs --bootstrap-server broker:9093 --entity-type topics --entity-
      name bleRSSISupport --alter --add-config retention.ms=10000
102       kafka-configs --bootstrap-server broker:9093 --entity-type topics --entity-
      name bleLocalization --alter --add-config retention.ms=10000
103       kafka-configs --bootstrap-server broker:9093 --entity-type topics --entity-
      name bleOccupation --alter --add-config retention.ms=10000
104
105       echo -e 'Successfully created the following topics:'
106       kafka-topics --bootstrap-server broker:9093 --list
107       "
108   kafka-ui:
109     container_name: kafka-ui
110     restart: always
111     image: provectuslabs/kafka-ui
112     ports:
113       - 8994:8080
114     networks:
115       - kafka_net
116     depends_on:
117       - init-kafka
118     environment:
119       KAFKA_CLUSTERS_0_NAME: localfirst
120       KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: broker:9093
121       KAFKA_CLUSTERS_0_METRICS_PORT: 9101
122       KAFKA_CLUSTERS_0_SCHEMAREGISTRY: http://schema-registry:8990
123       KAFKA_CLUSTERS_0_KAFKACONNECT_0_NAME: first
124       KAFKA_CLUSTERS_0_KAFKACONNECT_0_ADDRESS: http://connect:8992
125 ...
```

Listing A.1: Apache Kafka services stack YAML file

Appendix B

Localization services settings

```
version: '3.6'
networks:
  internal_net:
    name: kafka_network
services:
    bjoern_server:
      image: COMPANY_CONTAINER_REGISTRY/bjoern_server
      container_name: bjoern_server
      restart: always
      networks:
        - internal_net
      ports:
       - "8989:8989"
    map_server:
      image: COMPANY_CONTAINER_REGISTRY/map_server
      container_name: map_server
      restart: always
      volumes:
        - ./mapServer:/app/
      networks:
        - internal_net
      ports:
       - "8995:8995"
    localization_engine:
      image: COMPANY_CONTAINER_REGISTRY/localization_engine
      container_name: localization_engine
      restart: always
      depends_on :
        - bjoern_server
      networks:
        - internal_net
    occupation_engine:
      image: COMPANY_CONTAINER_REGISTRY/occupation_engine
      container_name: occupation_engine
      restart: always
      depends_on :
       - bjoern_server
      networks:
        - internal_net
```

Listing B.1: Localization services stack YAML file

Bibliography

[1] Shadi Al-Sarawi, Mohammed Anbar, Rosni Abdullah, and Ahmad B Al Hawari. «Internet of things market analysis forecasts, 2020–2030». In: *2020 Fourth World Conference on smart trends in systems, security and sustainability (WorldS4)*. IEEE. 2020, pp. 449–453 (cit. on p. 1).

[2] In Lee and Kyoochun Lee. «The Internet of Things (IoT): Applications, investments, and challenges for enterprises». In: *Business horizons* 58.4 (2015), pp. 431–440 (cit. on p. 1).

[3] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. «Internet of Things (IoT): A vision, architectural elements, and future directions». In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660 (cit. on p. 1).

[4] *ISO/IEC 30141:2018 Internet of Things (IoT) — Reference Architecture*. International Organization for Standardization (ISO), 2018 (cit. on p. 1).

[5] Somayya Madakam, Vihar Lake, Vihar Lake, Vihar Lake, et al. «Internet of Things (IoT): A literature review». In: *Journal of Computer and Communications* 3.5 (2015), p. 164 (cit. on p. 2).

[6] *ISO/IEC 21823-1:2019, Internet of things (IoT) — Interoperability for IoT systems — Part 1: Framework*. International Organization for Standardization (ISO), 2019 (cit. on p. 2).

[7] *Internet of things (IoT) — Industrial IoT*. International Organization for Standardization (ISO), 2020 (cit. on p. 2).

[8] *ISO/IEC 30162:2022, Internet of Things (IoT) — Compatibility requirements and model for devices within industrial IoT systems*. International Organization for Standardization (ISO), 2022 (cit. on p. 2).

[9] *ISO/IEC 30165:2021, Internet of Things (IoT) — Real-time IoT framework*. International Organization for Standardization (ISO), 2021 (cit. on p. 2).

[10] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. «Internet of things: A survey on enabling technologies, protocols, and applications». In: *IEEE communications surveys & tutorials* 17.4 (2015), pp. 2347–2376 (cit. on p. 2).

[11] Ulises Carrasco, Pedro Daniel Urbina Coronado, Mahmoud Parto, and Thomas Kurfess. «Indoor location service in support of a smart manufacturing facility». In: *Computers in Industry* 103 (2018), pp. 132–140 (cit. on p. 2).

[12] A. Brooks B. Atkin. *Total Facilities Management (Third edition)*. Oxford: Wiley-Blackwell, 2009 (cit. on p. 2).

[13] Keith Alexander. «Facilities management practice». In: *Facilities* 10.5 (1992), pp. 11–18 (cit. on p. 2).

[14] *ISO 41001:2018, Facility Management — Management systems — Requirements with guidance for use*. International Organization for Standardization (ISO), 2018 (cit. on p. 2).

[15] *ISO 41011:2017, Facility Management — Vocabulary*. International Organization for Standardization (ISO), 2017 (cit. on p. 2).

[16] *ISO 41012:2017, Facility management — Guidance on strategic sourcing and the development of agreements*. International Organization for Standardization (ISO), 2017 (cit. on p. 2).

[17] *ISO/TR 41013:2017, Facility management — Scope, number concepts and benefits*. International Organization for Standardization (ISO), 2017 (cit. on p. 2).

[18] *ISO 41014:2020, Facility management — Development of facility management strategy*. International Organization for Standardization (ISO), 2020 (cit. on p. 2).

[19] *ISO/FDIS 41015, Facility management — Influencing behaviours for improved facility outcomes and user experience*. International Organization for Standardization (ISO), 2018 (cit. on p. 2).

[20] M Tscherkassky-Aleksić. «Internet of Things for Facility Management». In: *Journal for Facility Management* 1.16 (2018) (cit. on p. 3).

[21] *How IoT can benefit a facility*. Nov. 2018. URL: `https://www.facilitiesnet.com/maintenanceoperations/article/How-IoT-Can-Benefit-a-Facility--18125` (cit. on p. 3).

[22] *Energy sources and power management in IoT sensors and edge devices*. URL: `https://devm.io/iot/energy-sources-power-management-iot-sensors-edge-devices-145006` (cit. on p. 3).

[23] Shanmugavelayutham Muthukrishnan et al. «Data streams: Algorithms and applications». In: *Foundations and Trends® in Theoretical Computer Science* 1.2 (2005), pp. 117–236 (cit. on p. 3).

[24] Charu C Aggarwal. *Data streams: models and algorithms*. Vol. 31. Springer, 2007 (cit. on p. 3).

[25] Paul Le Noac'H, Alexandru Costan, and Luc Bougé. «A performance evaluation of Apache Kafka in support of big data streaming applications». In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 4803–4806 (cit. on p. 3).

[26] Guenter Hesse, Christoph Matthies, and Matthias Uflacker. «How fast can we insert? an empirical performance evaluation of apache kafka». In: *2020 IEEE 26th international conference on parallel and distributed systems (ICPADS)*. IEEE. 2020, pp. 641–648 (cit. on p. 3).

[27] Shubham Vyas, Rajesh Kumar Tyagi, Charu Jain, and Shashank Sahu. «Literature Review: A Comparative Study of Real Time Streaming Technologies and Apache Kafka». In: *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*. IEEE. 2021, pp. 146–153 (cit. on p. 3).

[28] Keiichi Yasumoto, Hirozumi Yamaguchi, and Hiroshi Shigeno. «Survey of real-time processing technologies of iot data streams». In: *Journal of Information Processing* 24.2 (2016), pp. 195–202 (cit. on p. 3).

[29] Adnan Akbar, Abdullah Khan, Francois Carrez, and Klaus Moessner. «Predictive analytics for complex IoT data streams». In: *IEEE Internet of Things Journal* 4.5 (2017), pp. 1571–1582 (cit. on p. 3).

[30] Mark Baillie, Saskia le Cessie, Carsten Oliver Schmidt, Lara Lusa, Marianne Huebner, and Topic Group "Initial Data Analysis" of the STRATOS Initiative. *Ten simple rules for initial data analysis*. 2022 (cit. on p. 4).

[31] Frederick Hartwig and Brian E Dearing. *Exploratory data analysis*. 16. Sage, 1979 (cit. on p. 4).

[32] Stephan Morgenthaler. «Exploratory data analysis». In: *Wiley Interdisciplinary Reviews: Computational Statistics* 1.1 (2009), pp. 33–44 (cit. on p. 4).

[33] *BLE/WiFi Gateway iGS01S User Guide*. URL: `https://www.ingics.com/doc/Gateway/GW0002_BLE_WiFi_Gateway_iGS01S_User_Manual.pdf` (cit. on p. 6).

[34] *kontakt.io KHWPO400F001 Lanyard Tag User Manual*. URL: `https://manuals.plus/kontakt-io/khwpo400f001-lanyard-tag-manual#axzz7u8XXg0ak` (cit. on p. 7).

[35] *iBeacon specifications*. URL: https://developer.apple.com/ibeacon/ (cit. on p. 7).

[36] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 8).

[37] *NumPy: The fundamental package for scientific computing with Python*. URL: https://numpy.org/doc/stable/ (cit. on p. 8).

[38] *SciPy documentation*. URL: https://docs.scipy.org/doc/ (cit. on p. 8).

[39] *Pandas documentation*. URL: https://pandas.pydata.org/docs/ (cit. on p. 9).

[40] Wes McKinney et al. «Data structures for statistical computing in python». In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. 1. Austin, TX. 2010, pp. 51–56 (cit. on p. 9).

[41] *Matplotlib documentation*. URL: https://matplotlib.org/stable/index.html (cit. on p. 9).

[42] *Seaborn documentation*. URL: https://seaborn.pydata.org/api.html (cit. on p. 9).

[43] *Bjoern repository*. URL: https://github.com/jonashaag/bjoern (cit. on p. 10).

[44] *Falcon documentation*. URL: https://falcon.readthedocs.io/en/stable/ (cit. on p. 10).

[45] *Confluent Kafka Python client documentation*. URL: https://docs.confluent.io/kafka-clients/python/current/overview.html#python-demo-code (cit. on p. 10).

[46] *Docker website*. URL: https://www.docker.com/resources/what-container/ (cit. on p. 11).

[47] *Docker Compose documentation*. URL: https://docs.docker.com/get-started/08_using_compose/ (cit. on p. 11).

[48] *Apache Kafka documentation*. URL: https://kafka.apache.org/documentation/ (cit. on p. 11).

[49] Nishant Garg. *Apache kafka*. Packt Publishing Birmingham, UK, 2013 (cit. on p. 11).

[50] *Express.js API reference*. URL: https://expressjs.com/en/4x/api.html (cit. on p. 13).

[51] *Node.js documentation*. URL: https://nodejs.org/dist/latest-v18.x/docs/api/ (cit. on p. 13).

[52] *JavaScript website*. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (cit. on p. 13).

[53] *EJS template engine documentation*. URL: https://ejs.co/#docs (cit. on p. 13).

[54] *Leaflet.js documentation*. URL: https://leafletjs.com/reference.html (cit. on p. 14).

[55] *KafkaJS documentation*. URL: https://kafka.js.org/docs/getting-started (cit. on p. 14).

[56] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybyłek. «Monolithic vs. microservice architecture: A performance and scalability evaluation». In: *IEEE Access* 10 (2022), pp. 20357–20374 (cit. on p. 16).

[57] Konrad Gos and Wojciech Zabierowski. «The comparison of microservice and monolithic architecture». In: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. IEEE. 2020, pp. 150–153 (cit. on p. 16).

[58] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. «Microservices: How to make your application scale». In: *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers 11*. Springer. 2018, pp. 95–104 (cit. on p. 16).

[59] Omar Al-Debagy and Peter Martinek. «A comparative review of microservices and monolithic architectures». In: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE. 2018, pp. 000149–000154 (cit. on p. 16).

[60] Ralph Droms. *RFC2131: Dynamic Host Configuration Protocol*. 1997 (cit. on p. 19).

[61] *Apache Kafka broker - Confluent documentation*. URL: https://docs.confluent.io/kafka/overview.html (cit. on p. 24).

[62] *UI for Apache Kafka - Provectus documentation*. URL: https://docs.kafka-ui.provectus.io/overview/readme (cit. on p. 24).

[63] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and Laurie Cuthbert. «Bluetooth positioning using RSSI and triangulation methods». In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. IEEE. 2013, pp. 837–842 (cit. on p. 39).

[64] *KW38/KW36 Localization base on RSSI ranging*. URL: https://www.nxp.com/docs/en/application-note/AN12977.pdf (cit. on p. 39).

[65] ChihKun Ke, MeiYu Wu, YuWei Chan, and KeCheng Lu. «Developing a BLE beacon-based location system using location fingerprint positioning for smart home power management». In: *Energies* 11.12 (2018), p. 3464 (cit. on p. 39).

[66] *Sequential Least SQuares Programming algorithm*. URL: https://github.com/scipy/scipy/blob/main/scipy/optimize/slsqp/slsqp_optmz.f (cit. on p. 43).

[67] Paolo Bellavista, Antonio Corradi, and Carlo Giannelli. «Evaluating filtering strategies for decentralized handover prediction in the wireless internet». In: *11th IEEE Symposium on Computers and Communications (ISCC'06)*. IEEE. 2006, pp. 167–174 (cit. on p. 46).

[68] *Kalman filters explained: Removing noise from RSSI signals*. URL: https://www.wouterbulten.nl/posts/kalman-filters-explained-removing-noise-from-rssi-signals/ (cit. on p. 46).

[69] Guoquan Li, Enxu Geng, Zhouyang Ye, Yongjun Xu, Jinzhao Lin, and Yu Pang. «Indoor positioning algorithm based on the improved RSSI distance model». In: *Sensors* 18.9 (2018), p. 2820 (cit. on p. 46).

[70] Qiang Li, Ranyang Li, Kaifan Ji, and Wei Dai. «Kalman filter and its application». In: *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE. 2015, pp. 74–77 (cit. on p. 46).

[71] Dan Simon. «Kalman filtering». In: *Embedded systems programming* 14.6 (2001), pp. 72–79 (cit. on p. 46).

[72] Greg Welch, Gary Bishop, et al. «An introduction to the Kalman filter». In: (1995) (cit. on p. 46).

[73] *Android API: Bluetooth Low Energy Advertising*. URL: https://source.android.com/docs/core/connect/bluetooth/ble_advertising (cit. on p. 79).

[74] *Apple iOS API: Advertising Data*. URL: https://developer.apple.com/documentation/corebluetooth/cbperipheralmanager/advertising_data (cit. on p. 79).

[75] *NATS.io official website*. URL: https://nats.io/ (cit. on p. 79).

[76] *Apache Pulsar documentation*. URL: https://pulsar.apache.org/docs/3.0.x/ (cit. on p. 79).