VRIJE UNIVERSITEIT BRUSSEL

Proef ingediend met het oog op het behalen
van de graad van Master in de Toegepaste Informatica

INDOOR NAVIGATION

By Centralized Tracking

MAXIM VAN DE WYNCKEL
2018-2019

Promotor: Prof. Dr. Olga De Troyer
Wetenschappen en Bio-ingenieurswetenschappen

Thesis submitted for the purpose of obtaining the degree of Master in the Applied Informatics

INDOOR NAVIGATION

By Centralized Tracking

MAXIM VAN DE WYNCKEL

2018-2019

Promoter: Prof. Dr. Olga De Troyer
Science and Bio-engineering Sciences

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix. I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

Firstly, I want to express my sincere gratitude to my promoter Prof. Olga De Troyer for supporting and motivating my thesis idea. Her guidance helped me bring structure to the thesis, not only on paper – but in the implementation as well.

I also gratefully acknowledge the help of Werner Van de Wynckel who provided various insights and opinions about positioning techniques, Emmelien De Roock who helped testing and setting up navigation tests.

This thesis idea was sparked during an internship at UZ Brussel. I would therefore thank Michel Vervoort and other employees at UZ Brussel for providing the required information needed to design this system with a hospital environment in mind.

Abstract (EN)

The main goal of this thesis is to create an indoor positioning system that guides a patient or visitor through a hospital building. Using only the smartphone of a person, the system should be able to track and guide the user to his/her destination. Unlike other possible solutions that focus on the smartphone application to determine the position, the proposed solution uses a centralized server with Bluetooth scanners.

These scanners are distributed throughout the hospital and report their scan results back to the centralized server for position calculation and navigation instruction generation. Finally, these instructions are sent to the smartphone application to guide the user to his/her destination. Apart from starting the transmission of a Bluetooth signal, the task of the smartphone application is limited to receiving and displaying these navigation instructions.

The centralized server approach provides companies that implement this system, complete control over which other functionalities to implement related to the determined location of the end-user (i.e. the patient or visitor). For instance, companies can trigger actions based on the end-user's location such as automatic registration when a user arrives at their destination or use it to track assets.

Starting from a comparison of existing positioning techniques, such as beacons or Wi-Fi positioning, this thesis gradually explains how the system can track a smartphone; how both visual and textual navigation instructions are created and the pros and cons of using this system in a hospital environment.

To achieve the goal of the thesis, first theoretical research was needed to come to a possible solution. Next, the solution was implemented, i.e. software was created. The software implementation shows the feasibility of the positioning approach as well as the navigation approach.

Abstract (NL)

Het doel van deze thesis is om een navigatiesysteem te maken binnen een gebouw, dat de patient of bezoeker doorheen een ziekenhuisgebouw kan leiden. Het systeem moet de persoon kunnen traceren en begeleiden naar zijn/haar bestemming met enkel het gebruik van een smartphone. Waar andere oplossingen vaak focussen op de smartphone applicatie om de positie te bepalen zal de voorgestelde oplossing een centrale server gebruiken met Bluetooth scanners.

Deze scanners zijn verdeeld over het ziekenhuis en versturen hun scanresultaten terug naar de centrale server die op zijn beurt de positie bepaalt en navigatie-instructies aanmaakt. Vervolgens worden deze instructies verzonden naar de smartphone applicatie om zo de gebruiker tot zijn bestemming te begeleiden. Buiten het versturen van een Bluetooth signaal is de taak van de applicatie gelimiteerd tot het ontvangen en versturen van de navigatie-instructies.

Door het gebruik van een centrale server voor het bepalen van de posities, krijgen bedrijven door dit systeem meer controle over het gebruik van de locatie van de eindgebruiker (m.n. de patiënt of bezoeker). Bijvoorbeeld kunnen bedrijven bepaalde acties uitvoeren op basis van de gebruiker zijn locatie, zoals een automatische registratie bij aankomst, of kunnen ze het systeem gebruiken om belangrijke voorwerpen/machines te traceren.

De thesis begint met een onderzoek naar de verschillende beschikbare positioneringsmethode binnen een gebouw, zoals ‘beacons’ of Wi-Fi positionering. Hierop gebaseerd zal deze thesis uitleggen hoe het systeem een smartphone kan traceren; hoe visuele en tekstuele instructies gemaakt worden en wat de voor- en nadelen van het gebruik van het systeem in een ziekenhuisomgeving zijn.

Om het doel van de thesis te bereiken werd eerst een theoretische oplossing uitgewerkt. Deze oplossing werd nadien geïmplementeerd in de vorm van een software implementatie. De software implementatie is gebruikt om de haalbaarheid van de positioneringsmethode en de navigatie aanpak aan te tonen.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programmable Interface
AoA	Angle of Arrival
BR/EDR	Bluetooth Basic Rate/Enhanced Data Rate
BT Classic	Bluetooth Classic
BLE	Bluetooth Low Energy
CI	Cell Identification
ECG	Electrocardiogram
eFuse	Electronic Fuse
EIR	Extended Inquiry Response
EM	Electromagnetic
GPS	Global Positioning System
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
ISM band	Industrial Scientific Medical radio band
JS	JavaScript
JSON	JavaScript Object Notation
KWS	Klinisch Werk Station (Dutch abbreviation for Clinical Work Station)
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport

OTA	Over the Air
PCB	Printed Circuit Board
REST	Representational State Transfer
RF	Radio Frequency
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SLAM	Simultaneous Localization and Mapping
TDoA	Time Difference of Arrival
ToA	Time of Arrival
TX	Transmitter
UUID	Unique User Identification
WORA	Write Once, Run Anywhere
QoS	Quality of Service

I. Thesis Overview

1. Introduction

Navigation is defined by the English Oxford Dictionary as “the process or activity of accurately ascertaining one's position and planning and following a route” (Oxford University Press, 2019). People use navigation all the time: to find a place to eat, to get to appointments or simply to find a bench to sit.

While in the past the position of stars and the sun were used to help with navigation (Taylor, 1971), we now have technology to support our navigation. Outdoors we use GPS technology that uses satellites in a direct line of sight to know our position. However, once we step inside a building, the walls and ceilings block the signals and result in an inaccurate position (Zandbergen & Barbeau, 2011).

Complex building such as hospitals can benefit from the possibility of indoor positioning and navigation to make it easier for the visitors to find their way throughout the building. Just like outside, indoor navigation could allow them to find a place to eat, get to their appointment or find a place to sit. The academic hospital of Brussels (UZ Brussel, 2018) is such a building where navigation is complex. Like in most hospitals they use a system of routes. When the patient knows the route number for the department he or she want to go to, he or she can follow this number using the reference signs which are placed all over the hospital. These routes are laid out in a way that they do not guide everyone through the same hallways to avoid crowds. This results in longer paths that often guide you to a central point before taking you to the actual destination. Next to route numbers, colors are used for the routes. Although, this looks like a simple and easy system, a lot of patients still have problems to find their destination. This has different reasons: Some patients don’t know where to find the route number; sometimes they lose track of the route because of lack of attention or because the signals are not always unambiguous, especially when elevators should be taken. Also, using colors may not be possible for people that suffer from color blindness.

Therefore, the UZ Brussel could benefit from an indoor navigation system and was prepared to investigate the potential of such an indoor navigation system. This resulted into the main objective of this thesis: to develop a system that improves the indoor navigation.

During the research, many approaches and companies, offering solutions to the indoor positioning problem were found. This was surprising because, as far as me and my mentor at UZ Brussel were aware of, they are hardly used in practice. It made us eager to discover the reason behind this lack of adoption. After my first meeting with UZ Brussel it became clear that the most important reason for hospitals not implementing the technology is because of the price and maintenance cost. Therefore, the challenge of this master thesis was to create a solution that would be reasonably priced and would not require much maintenance over time.

After further investigation, the main objective of this thesis was refined as follows: to create a system that improves the indoor navigation and uses the patient's estimated location for location-based actions. The acquisition and maintenance cost should not be higher than existing solutions. Furthermore, patients or visitors should not be forced to use any other devices or hardware other than their own smartphone.

To achieve this objective, both theoretical research and the development of a solution, i.e. a software implementation, were done. The implementation allowed evaluating the theoretical solution. It was used to verify that the developed positioning and navigation methods were accurate and beneficial to guide visitors inside a complex building.

Originally, the idea for the solution started in 2016 with the simple idea to reverse the way in which beacons contribute to indoor positioning. During the first stages of the research, this idea was tried out to make sure it would be a viable positioning method.

After having the certainty that the idea would work, the hardware for the scanners was selected (section 7.3.1). This hardware was used to gradually develop an indoor positioning method consisting of the scanner hardware, tracking server and navigation application.

The result of this idea is presented in this thesis. We start by providing the requirements formulated for the system in chapter 2. Next, in chapter 3, we discuss the different positioning techniques that are available to track people with a smartphone inside a building and provide background information on Bluetooth (chapter 4) and on how RF-based positioning techniques work (chapter 5).

The implementation part of the thesis starts with chapter 6 that describes the chosen positioning technique and a general architectural overview of all system components (scanners, tracking server and navigation app) is given in chapter 7. These components are explained in more detail in chapters 8, 9 and 10 respectively.

The thesis ends with a conclusion, limitations and further work (chapter 11).

2. System Requirements

In this chapter, we present the different requirements formulated for our solution. We start with the non-functional requirements, followed by the functional requirements and the usability requirements. To leave room for different technical solutions, the requirements are formulated at a high level.

2.1. Non-functional Requirements

Because our system is intended for UZ Brussel, which is a hospital, our project has to be considered as an IT healthcare project, the technical possibilities can be limited due to government restrictions such as the General Data Protection Regulation (GDPR) in Europe (European Union Agency for Network and Information Security, 2018) or the Health Insurance Portability and Accountability Act (HIPAA) in the US (European Union Agency for Network and Information Security, 2009). Therefore, an obvious requirement for a healthcare system is the security of the system. It should not be possible for an unauthorized user to gain access to the real-time location of the patients in the hospital or worse their personal information.

Next there was the concern over using Bluetooth or any other radio frequency signal in a hospital environment. The signals could potentially cause issues with medical equipment such as ECG's (Electrocardiogram) or patients and infants that are sensitive to EM fields (Wikipedia, 2019b).

Finally, price and resource cost were the most important requirement as it was the goal of this thesis to reduce the cost related to the smartphone application. Existing systems usually have a high development and installation cost, because they require a lot of calibration and usually focus a lot on the smartphone application to do all the work.

2.2. Functional Requirements

The following baseline requirements had to be fulfilled by the system:

- Ability to **track** a person inside a hospital building
  - Multiple floors with multiple different connected buildings
  - The tracked object is the smartphone of this person (i.e. the visitor or patient)
- Ability to guide a person to **navigate** from anywhere in the hospital to a predefined set of possible destinations. Note that some routes or hallways are not accessible without a badge.
  - Guiding must be done through the smartphone of the visitor or patient
- Ability to **configure** new routes and new floors
  - Without these changes requiring programming or updating of the mobile application
- Ability to be combined with the (appointment) system of UZ Brussel
  - Developers must be able to link the navigation to the appointment system to allow quick navigation to the appointment destination

2.3. Usability Requirements

Existing systems that were researched in this thesis often focus on the usability of the end-user, i.e. the user that must be guided. However, the system also needs to be installed and maintained. If this requires trained people, than this would result in an additional cost (see e.g. the duration of Gatwick Airport installation mentioned in section 3.2.1).

For this reason, the focus of our usability concerns will be equally shared between both the end-users that require to be guided to their destination and the users that will have to install and maintain the system in the hospital.

In general, we can state that end-users should be able to use the smartphone application directly, without any training or external help. For the users that need to do the installation and maintenance, the installation needs to be quick without the need for extensive testing and fine-tuning.

II. Background and Related Work

3. Indoor Positioning

To navigate inside a building to a certain location, we need to know the position of the user inside the building. The more accurate and real-time this location is, the more accurate the instructions can be that are provided to the user.

Outside, phones can use GPS satellites to determine their position. Each connected satellite is in a geostationary orbit, meaning they remain at the same location relative to earth (The Office of the Secretary of Defense, 2008). Using multiple connected satellites and a GPS receiver, this receiver can trilaterate the location relative to those stationary satellites.

If there are only few connected satellites, the location will be less accurate. Disruptions occur when the satellites are no longer in a direct line of sight. Research by Zandbergen and Barbeau (2011) shows that the positioning using GPS is possible indoors, but due to the various angles of reception this accuracy can change depending on how close you are to an outside wall. This means that GPS is not suitable for getting an indoor location. In the next sections, we investigate the most frequently implemented methods of indoor positioning.

3.1. Visual Positioning

Visual positioning can be compared to how the human memory works. Humans match locations based on previous knowledge. The first thing we do to position ourselves is look for landmarks (Ruhr-Universitaet-Bochum, 2015). Normal (1999) also showed that the type and size of landmarks could help us to determine our position.

An example of this method is called **SLAM** (Simultaneous Localization and Mapping).

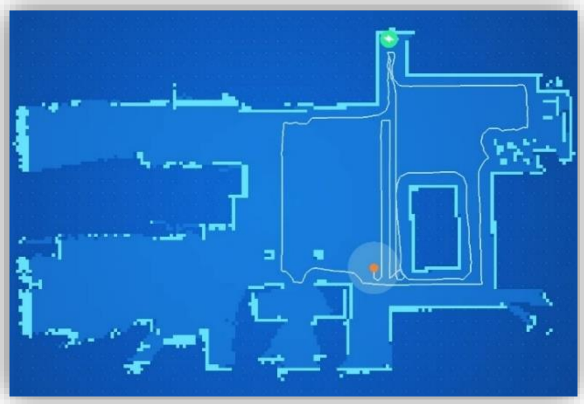


Figure 1 - Lasers used in SLAM positioning on a Xiaomi robot vacuum. (Roberts, 2017)

This technology creates a map of its surroundings and meanwhile uses this map to determine and follow its own location. This means that the map is continuously updating and improving. The mapping is done with 2D or 3D sensors such as cameras or lasers (Dissanayake, 2001).

This technology could be used in combination with augmented reality to show the directions on the screen of the smartphone. While this technology has many advantages in the field of robotics, it was abandoned for this thesis. The only sensor that a phone could use to map a location would be the camera. Walking around with the camera aimed forward is not ethically accepted in a hospital.

3.2. Radio Frequency Positioning

It is possible to perform positioning using radio frequencies (RF). Just like the visual positioning method we can use the RF receivers or transmitters to act like landmarks.

For this thesis, we considered three RF positioning methods that are commonly used and were feasible inside a hospital environment: beacon positioning, Wi-Fi positioning and Bluetooth tracking. Every technology has their own advantages and disadvantages which will be discussed respectively in sections 3.2.1, 0 and 0.

3.2.1. Beacon Positioning

Both Google with Eddystone (Google LLC, 2018a) and Apple with iBeacon (Apple Inc, 2018) invested a lot of resources in researching the use of beacons for indoor positioning.

Beacons are often advertised as easily installable things that you just attach to your ceiling or wall every 5-10 meters. They are in most cases sold as battery-powered devices for a reasonable bulk price.

As standalone devices, beacons do not offer any significant use. In combination with a mobile device they can be used for applications ranging from localized advertisements to indoor positioning such as our use case.

Gatwick Airport (UK) used 2000 beacons to allow their visitors to use augmented navigation. Their reasoning for this approach was the low logistical complexity and cost of the battery powered devices (Gatwick Airport Press Office, 2017).

However, when you have a large building like a hospital or an airport, you will need to change all these batteries every one or two years depending on the manufacturer. This results in both additional work and battery cost.

Because beacons do not have any logic apart from being able to broadcast information for a long duration, an application is required to process this broadcasted information. In the case of indoor positioning this requires the creation of an app for calculating the position made for each mobile operating system.

Other implementations such as infsoft’s LocAware platform® (infsoft GmbH, 2018) use a server to calculate the position, but the actual receiving happens at the user’s smartphone. Because the mathematical aspect of the implementation is moved to a server, it lowers the complexity of the smartphone application. However, it does cause the latency to increase because the receiving of the Bluetooth signals transmitted by beacons is still done on the smartphone.

3.2.2. Wi-Fi Positioning

Wi-Fi is a form of RF signal; it can be used to determine the relative distance between a mobile and stationary device. Buildings like UZ Brussel have several Wi-Fi access points on stationary locations throughout the hospital. Experiments by Lindemann, Schnor, Sohre and Vogel (2016) show that a reasonable accuracy can be received from the trilateration of access points.

Just like for beacons, these signal readings need to be translated into a location by means of a Wi-Fi scanner application or sent to a positioning server like done in the Wi-Fi positioning case study described by Pathak et al (2014).

While Wi-Fi positioning offers many benefits because it is built upon existing infrastructure with a much higher range than Bluetooth, it is not always the best solution.

To determine an accurate location, multiple access points are required in range. If this is not the case, Wi-Fi access points that have a similar transmitting strength than the others need to be added to fill these ‘blind spots. Adding Wi-Fi access points is not always the best solution. It can be expensive because the added hardware needs to be the same as the existing hardware in the building, for more professional hardware this can quickly be more than hundred euros. It also increases the saturation of the 13 available channels in Belgium due to their close proximity to each other (Test-Aankoop, 2018).

3.2.3. Bluetooth Tracking

The chosen method for this thesis was to reverse the whole idea of beacons. Instead of having devices throughout the hospital that broadcast a signal and having a scanning smartphone application on the smartphone, it was decided to use the smartphone as the broadcasting device (using Bluetooth – see chapter 4) and to have scanning devices looking for these smartphones instead. The positioning techniques remain the same with the only difference that the phone of the user is now the "broadcasting beacon".

In this case, the application will only need to function as a user interface for the navigation instructions, rather than as a beacon scanner that calculates the location. Apart from having the anticipated advantage of spending less time and resources on the technical aspects of a smartphone application it also has a lot more room for expansion.

With beacon-based positioning the application needs to share the location with a server if the company or building manager wants to use this location for server-side actions. These actions could allow for automatic registration when the patient enters a waiting area or for tracking purposes to navigate users through a less crowded hallway. Our proposed reversed method removes this requirement of the application to constantly share and update its location because the location is determined by the device-scanners in the hospital.

Because this method can look for regular Bluetooth devices without special protocol or application, it allows for other advantages such as inexpensive asset tracking. Instead of using the inexpensive beacons to determine a position, they can be placed on (portable) expensive medical tools; in this way their location can always be found. While this is a bonus of Bluetooth tracking, it will not be explored throughout this thesis. The goal remains to achieve indoor navigation rather than indoor tracking.

Tracking can be done with either Wi-Fi or Bluetooth. For our approach, the choice was made to use the same RF communication as beacons. Wi-Fi tracking would be possible, but it would require a constant broadcast of the mobile devices. This would saturate the radio band and use a lot of smartphone battery. In addition, iOS 8 devices perform MAC address rotation what will prevent the tracking of the device (Apple Inc, 2017). Section 8.2.3 of this thesis will explain what Bluetooth can do to circumvent this MAC rotation.

Positioning by tracking a Bluetooth device is mostly used for proximity marketing and customer tracking purposes. Shops can use it to visualize the routes customers take throughout the shop such as used in Accuware's Beacon tracker (Accuware Inc., 2018). Because this approach does not require an application, the customers do not need to install an application for being tracked.

Systems utilizing this method for tracking purposes were used as reference points for the implementation created for this thesis. While beacons are very simply (and dumb) devices that only broadcast a configurated signal, these tracking nodes such as the ones used by Accuware (see Figure 2) are more sophisticated. This sophistication allows the scanners to not only scan, but also report the scan results to a server which prevents them from being battery powered.

The lack of a battery prevents them from simply being glued to any wall or ceiling without additional cabling. However, their installation is permanent and does not need to be easily accessible to change the battery.

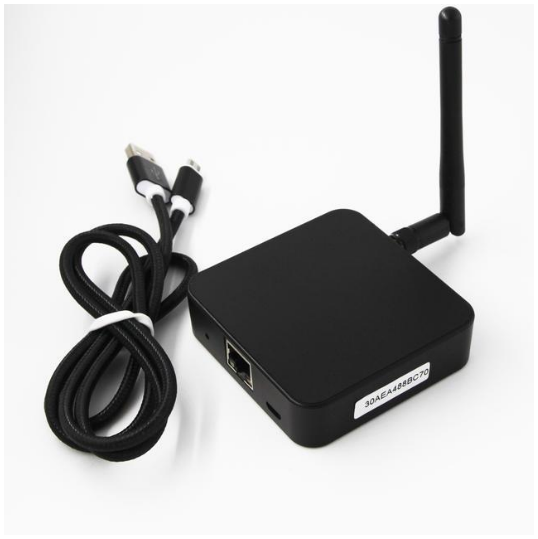


Figure 2 - Accuware's BLE v3 tracking node with ethernet and Wi-Fi connection (Accuware Inc., 2018)

4. Bluetooth

Founded in 1998, Bluetooth is a short-range wireless communication technology standard operating on the unlicensed ISM (Industrial, Scientific and Medical purposes) band. The radio frequency (RF) operates within the 2.4GHz and 2.4835 GHz range (Bluetooth Special Interest Group inc., 2016) and is divided into 79 channels of 1 MHz that are switched 1600 times per second to avoid interference (Bluetooth Special Interest Group inc., 2016, pp. 257-261).

The Bluetooth Special Interest Group oversees the development of the technology (Bluetooth Special Interest Group inc., 2016). They are registered as a non-profit organization meaning it can be commercially used for free. They do however ask for qualification fees when you want your product officially listed (Bluetooth Special Interest Group Inc, 2018b).

There are multiple classifications of Bluetooth devices based on the transmission power expressed in dBm. These are the decibels send with 1 milliwatts over a 50 Ohm load (Bigelow, Carr, & Winder, 2001). The load is the antenna used for communication what in the case of Bluetooth is 50 Ohms.

Table 1 - Bluetooth classification

	dBm	***Range***
Class 3	0 dBm	1 meter
Class 2	4 dBm	10 meters
Class 1	20 dBm	100 meters

(Bluetooth Special Interest Group inc., 2016)

The sensitivity of a receiver is also expressed in dBm. Sensitivity is the indication of how weak a traveling signal can be before it can no longer be received.

4.1. Classic and Low Energy Bluetooth

Bluetooth Low Energy (BLE) was released with the 4.0 specification (Bluetooth Special Interest Group inc., 2016, p. 286). It is very similar to classic Bluetooth (BT), but uses less power. This makes it more ideal for small battery powered devices such as beacons as discussed in section 3.2.1.

Apart from technical differences, the protocol for discovering these devices differs. Due to the choice for the reverse beacon positioning method that relies on discovery, it is important to investigate these differences.

4.1.1. Bluetooth (BT) Classic Discovery

Scanning for classic Bluetooth devices is on-demand. The initiating device sends an **inquiry** packet to everyone listening. Those who receive the inquiry will respond with their device information inside an inquiry response (Bluetooth Special Interest Group inc., 2016, pp. 371-372).

This response was changed in Bluetooth 2.1 with the addition of information inside the newly renamed *Extended Inquiry Response*. This EIR packet adds information about services and manufacturer information that can be used to identify devices running a specific application or feature.

4.1.2. Bluetooth Low Energy (BLE) Discovery

BLE discovery is very different from the classic discovery. BLE devices **advertise** themselves in pre-defined intervals. The device that wants to scan for other BLE devices needs to have a broad enough scan window to receive these advertise packets when performing a passive scan (Bluetooth Special Interest Group inc., 2016, pp. 169-172).

Active scanning is also supported, this works similar as BT Classic and contains additional information not included in the passive advertise packets.

Unlike classic Bluetooth, BLE only uses three of the 79 available channels to reduce battery usage. This increases the chance of advertise conflicts by other BLE transmitters within the same range, but this is minimized due to the relatively long time between advertise broadcasts.

5. RF-Based Positioning Methods

When using fixed points to determine a position it is required to have some estimate of distance between a transmitter and receiver. If a receiving device would only be able to know when a signal is “near enough” without an indication of “how near”, it would only know a rough estimate of the location based on the receiving range (see section 5.1).

With the following different types of positioning between transmitters and receivers we call the ‘mobile device’ the transmitting device we need to determine the position of. The ‘stationary devices’, called the receivers, have fixed positions with known locations.

Two general types of positioning methods will be discussed in this section: calibration-based positioning methods (section 5.1 and the “Calibration positioning” in section 5.4) and mathematical methods (sections 5.2, 5.3, and the “Mathematical positioning” in section 5.4).

Calibration-based positioning uses heavy training that links measurements to various locations inside the building. This is a good solution because buildings are mathematically unpredictable. Due to the enormous amount of obstructions such as walls in various materials, the measurements may fluctuate.

However, because the accuracy of calibration-based positioning depends a lot on the distance between calibrations, this becomes a very long process. The beacon-based positioning in Gatwick Airport took two months of calibration (Gatwick Airport Press Office, 2017). Additionally, calibration must be redone when a receiver is moved or no longer operational.

5.1. CI (Cell Identification)

Positioning using cell identification is very simple but inaccurate. The location of the mobile device is based on the known location of the stationary device in range (Trevisani & Vitaletti, 2004).

This type of positioning can be used when the exact location is not as important such as in the example given by Aalto, Göthlin, Korhonen and Ojala (2004) where it was used behind shop windows to provide advertisements for people walking past it. In the example given in the aforementioned paper, the goal was to rapidly detect proximity rather than determining a location.

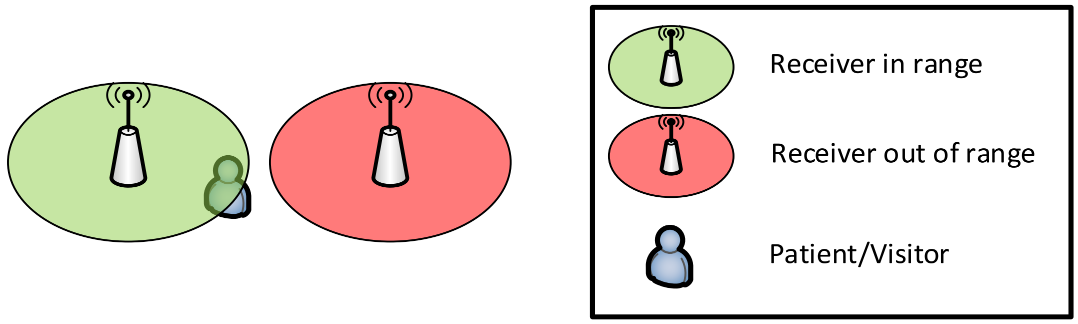


Figure 3 – Cell identification positioning example

5.2. ToA (Time of Arrival)

With this method of positioning, a very accurate clock synchronization between all devices is made. Packets sent between devices include a timestamp. The receiver will compare the timestamp inside the packet with its own clock to determine the distance using trilateration (The Institute of Electrical and Electronic Engineers, 2010). Trilateration or multilateration is a technique to determine the unknown location of a user (patient/visitor in Figure 4) using the distance or transmission time to three or more known locations (the receivers in Figure 4) (Schaefer, 2018).

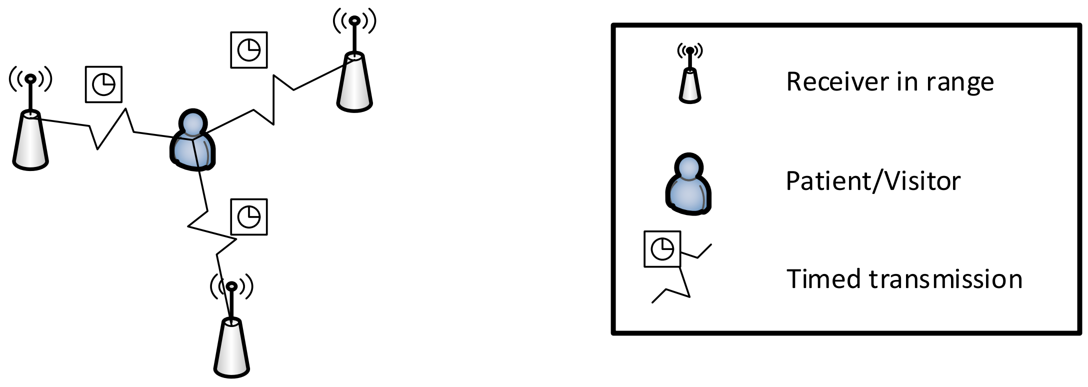


Figure 4 – Time of arrival positioning example

However, trilateration with timed transmissions is not possible with standard Bluetooth hardware. The official Bluetooth specifications allow for a minimal clock accuracy of $2\,\mu s$ (Bluetooth Special Interest Group inc., 2016, p. 2604). While $2\,\mu s$ seems like an accurate resolution, for using it to determine the position it is very inaccurate. RF waves are electromagnetic waves that travel with the speed of light as proven in the special relativity theory by Einstein (1905). With this knowledge we can calculate the exact impact $1\,\mu s$ has on the calculated distance. Since we are interested in the distance, 299 792 478 meters per second was used for the speed of light in the calculation as this is the exact value used to express the metric system (Bureau International des Poids et Mesures, 1983). This shows the impact being 299.792 meters per μs. For the 'short-range' Bluetooth wireless communication and our goal to achieve indoor positioning this is way too inaccurate. The calculation is confirmed by the more rounded value in the paper of Jami, Ali and Ormondroyd (1999).

Also shown in this paper are multiple variations on this positioning method such as TDoA (Time Difference of Arrival), OTD (Observed Time Difference) and TA (Timing advance) which all require a clock synchronization between either the stationary or mobile devices.

5.3. AoA (Angle of Arrival)

Positioning is determined by the angle of the received signal of at least two stationary devices using triangulation. Unlike trilateration that uses the range between receivers and the user, triangulation determines the position of a user using the angle of the reception (Schaefer, 2018).

For a receiver to determine this angle, it needs several Bluetooth antenna's facing in different directions (Lehtimaki, 2018). Silicon Labs has made a prototype array of antenna's that shows this method being used in practice. As shown by Lehtimaki (2018) this is not yet reliable due to the short range of Bluetooth. It would also require a direct line of sight between the mobile device and at least two arrays of receivers (Gauer, 2015).

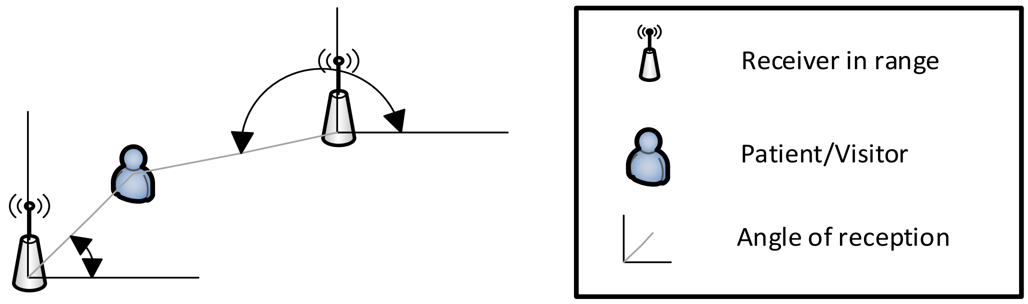


Figure 5 – Angle of arrival positioning example

5.4. RSS (Received Signal Strength)

When a transmitter sends a signal whose power is expressed in dBm (decibels per 1 mW over a 50 Ohm load) (Bigelow, Carr, & Winder, 2001), it weakens over distance or when obstructed by the medium it travels through (Bose & Foh, 2007). As demonstrated in this last paper, receivers can use the received signal strength indicator (RSSI) to determine the distance between two devices. According to the latest Bluetooth Specifications (Bluetooth Special Interest Group inc., 2016), manufacturers can choose their own transmission power. This makes the RSSI not only sensitive to distance but also by the hardware that is being used (Gao, 2015).

Mentioned specifically in the official blogpost by Gao (2015), RSSI should be used to indicate if a signal is getting closer or stronger, or to trilaterate it as shown by Wang et al (2013) where they narrow down a location by using multiple RSSI sources.

5.4.1. Mathematical RSS Positioning

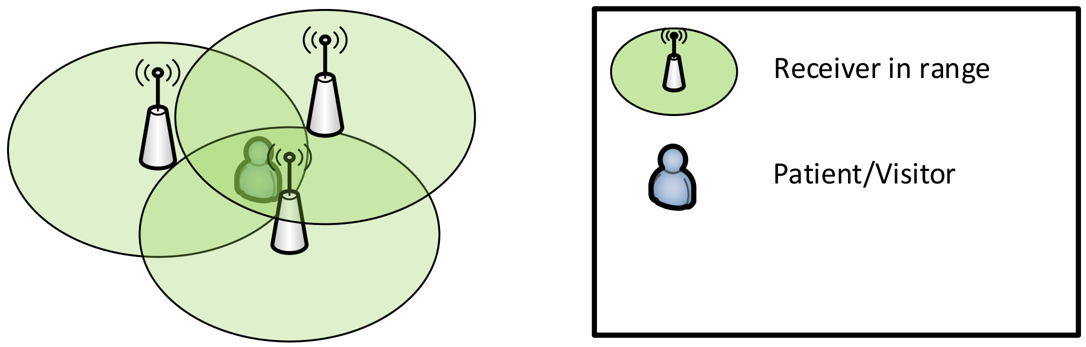


Figure 6 – Received signal strength positioning

To perform trilateration, the signal strength needs to be converted to an estimated distance. A later experiment in this thesis will show that the signal strength does not change linear to the distance. Because RF signals have a very predictable propagation through air, the distance can be estimated using a signal propagation formula (Pathak, Palaskar, Palkar, & Tawari, 2014).

5.4.2. Calibration-based RSS Positioning

Calibration-based RSS positioning will use the received signal strength from one or more receivers to determine a possible location. Fingerprinting is used to create a database of reference RSS values. After the calibration, this database is used to try and match a set of RSS values to a possible 'fingerprinted' location (Faragher & Harle, 2015).

The paper by Torres-Sospedra et al. (2015) uses this type of positioning for its campus navigation system. In that paper, it is described how they obtained more than 20 thousand different fingerprints, each linked to a specific location on campus.

Other approaches to the fingerprinting include machine learning such as described in the thesis by Ugave (2004). In this thesis, the calibration is done by following routes. The RSS readings that are captured during these routes train a neural network to output a location.

5.5. Related Work

This section of the thesis discusses related work of other RF-Based positioning implementations in relation to the goal of this thesis.

The main goal of this thesis was to 'improve' the indoor navigation of an end-user and to provide location-based actions. This improvement relates to both the traditional methods of navigating in a hospital (i.e. following route numbers or colors) and existing applications or technologies that the end-user may be familiar with (i.e. Google Maps or smartphone usage).

The journal article by Hansen, Wind, Jensen, & Thomsen (2003) discusses a seamless transition from GPS to indoor positioning. Instead of forcing the user to use a different application or technology for in-and outdoors, the paper explains how the navigation and tracking aspect can be handed over as soon as it detects known indoor landmarks. They achieve this by having a centralized server that stores fingerprint data of Wi-Fi access points. The smartphone application still needs to perform the scanning and suffers the limitations discussed in section 3.2.2. However, with some modifications such as the use of Beacons (section 3.2.1) and using the fingerprint server to log back the location similar to the infsoft's LocAware platform® (infsoft GmbH, 2018), this method could provide the end-user with a single (familiar) application for all types of navigation.

A similar approach was given in the paper by Kha Hoang, et al (2013), the paper does not specifically discuss the use or transition from GPS to indoor navigation but calculates the position of a user on the server using various sensor readings. These sensor readings can include accelerometers or (despite not being specifically mentioned) even a GPS location. Together, these sensor readings can determine a more accurate location while still allowing the central server to use the location for other purposes. Other than the chosen implementation in this thesis, that uses scanners that transmit scan results to a tracking server, the paper lets the smartphone scan for Wi-Fi access points. Just as the scanners, the smartphone will provide these 'raw' scan results together with other information to the tracking server in order to determine a location. Section 8.2.5 briefly discusses how additional sensor data can help get a more accurate location.

Like this thesis, that uses scanner nodes, is the experiment discussed in the paper by Oosterlinck, Benoit, Baecke, & Van de Weghe (2017). Both systems use Bluetooth scanners that scan for smartphone devices. While the goal of this thesis is to navigate the end-user to their destination, the goal of the paper is to investigate the applications of Bluetooth tracking for marketing purposes. Unlike the approach that will be discussed in this thesis, the system that is used in the paper is not real-time and does not use a centralized server. The retrieved 'tracking data' is stored on each scanner devices and needs to be manually transferred and computed to determine a location. Still, the technical aspects and results of the paper closely resemble the findings in this thesis such as the reduced cost, limitations and the possibility of moving away tasks from the smartphone application. More practical implementations of this tracking technology include the Beacon tracker by Accuware Inc. (2018) as discussed in section 3.2.3.

III. Implementation

6. Positioning Method

The choice of positioning using the Bluetooth tracking concept has several consequences. A hospital consists of hallways, elevators and waiting rooms. It is important that the location is accurate at times where the user must make a choice where to go.

With this knowledge a cell ID based positioning as shown in the image below (where the red areas (representing the range of the receiver) indicate a critical point where a choice must be made), would suffice for the most basic type of navigation.

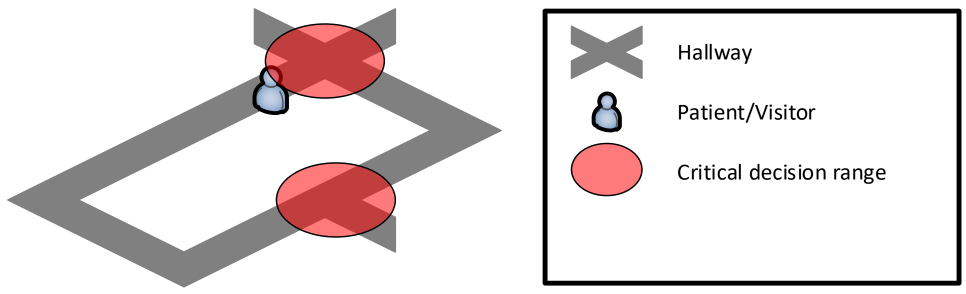


Figure 7 – Cell identification inside a hallway

While this offers a good indication of a user reaching a corner or other critical point, it is not scalable when you want to give the user an estimation of their position inside a hallway. Receivers may only cover certain critical points and may not cover the entire hospital. If a new destination has to be added in an uncovered location, new receivers need to be installed and configured.

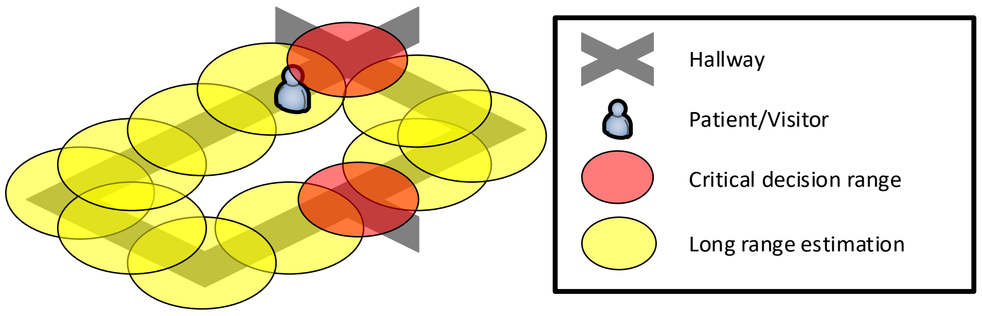


Figure 8 – Cell identification in combination with RSS trilateration inside a hallway

To solve this, the system will be a combination of RSS trilateration and cell identification. The critical points will have a higher positioning priority than the RSS trilateration by filtering out transmitters that are ‘possibly’ too far.

Note that hallways are more forgiving in accuracy than ‘large space’ positioning where the exact 2-dimensional location in this space is required. With just two receivers in range, the position can be determined over just a single dimension because we can assume the user follows the direction of the hallway.

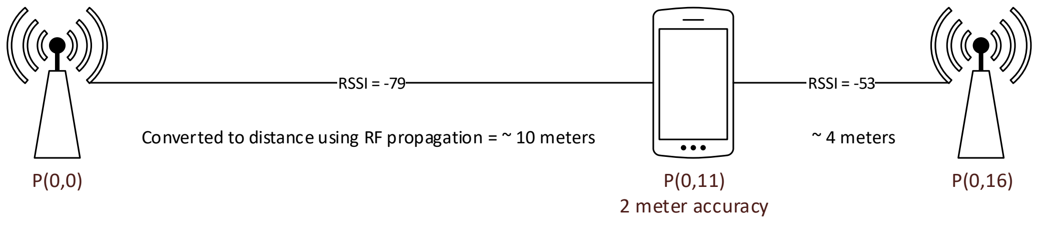


Figure 9 – Position estimation between two stationary receivers in the same X dimension

In Figure 9, point P(0,0) is representing receiver 1 and point P(0,16) is representing receiver 2 with their distance being 16 meters. Because we can assume the X position of the phone remains the same, its location is approximately at point P(0,11). The implementation of this using the latitude and longitude of the two receivers is explained in section 9.3.2.

By keeping the chosen positioning method cell ID-based with trilateration as an option, it is possible to create a scalable system where accuracy depends on the number of installed receivers. These receivers will from now on in the thesis be referred to as the *scanner nodes*.

7. System Design

The complete system consists of three main components. Most importantly are the *scanner nodes* as mentioned at the end of the previous section, they scan for devices within their range and are placed throughout the hospital. These scan results are sent to the *tracking server* that handles the tracking and navigation aspect of the system.

Lastly there is the publicly available application that registers itself to a public endpoint of the server and initiates the navigation request to a specific destination. This *navigation app*, used by the patients and visitors, maintains a constant connection to the tracking server during the navigation, so it can receive up-to-date instructions.

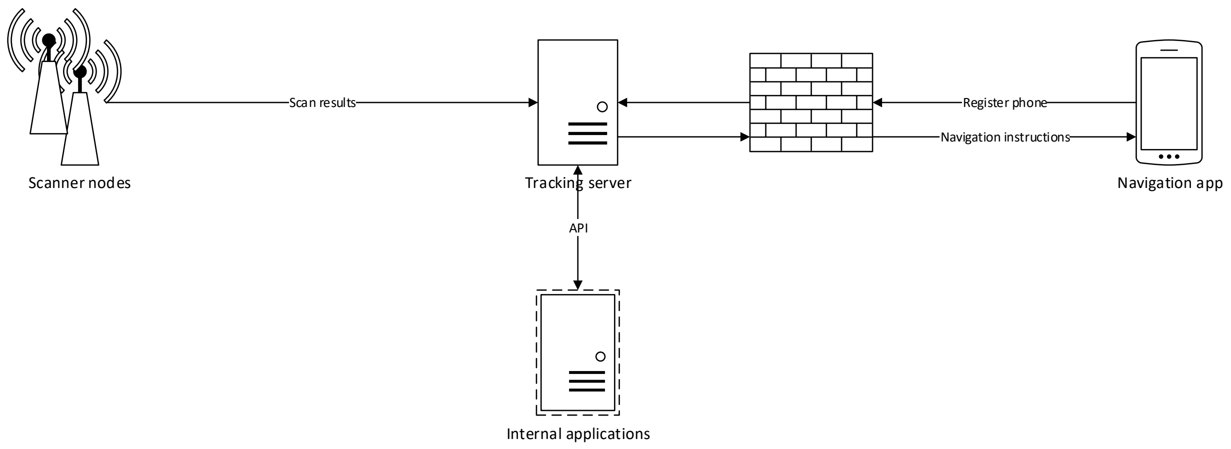


Figure 10 – General system architecture with scanners, tracking server and navigation app

Internal applications such as the KWS system of UZ Brussel (Dutch abbreviation for Clinical Work Station) can connect with the tracking server to use the user's location for its own location-based actions (Partezis, 2010). An example of such action could be automatic registration when the patient enters the waiting room of his appointment.

For internal applications to hook into this system, an application interface (API) must be made that can provide the required information to these applications. Because this tracking information is considered sensitive in a hospital environment, the decision was made to have the tracking server on-site in the hospital. This not only increases the security but also lowers the latency between the scanner nodes and the server.

This chapter of the thesis will provide typical use cases for different types of users that interact with different components of the system (section 7.1). The communication between these components and their architecture will be explained in the following sections respectively: 7.2, 7.3, 7.4 and 7.5.

7.1. Use Cases

The system needs to be designed with different types of users in mind, each having their own reasons for interacting with the system. Patients and visitors, i.e. the *end-users*, use the navigation application to navigate throughout the building, the *tracking server configurator* needs to set up the scanner nodes and create all the possible routes. In addition, there are *third party developers* who need to interface with the system through the use of API's. In this section, we outline typical scenarios for each user type.

End-user

Scenario: *A patient has an appointment to get an x-ray scan of his broken arm. Afterwards, he must wait for his physician in the waiting room to get the results of this scan.*

The patient can use a link on the UZ Brussel's patient portal to start navigation to his appointment. This link opens the application and automatically starts the navigation towards radiology.

Navigation is based on instructions that shows the turns and elevators the patient needs to take to get to his appointment in time. He does not need to register himself once he arrives at the radiology; this is done automatically after the patient arrives in the waiting room.

Once the x-ray scan is done, the application starts the navigation to the waiting room of the physician. This navigation works the same as before and will again notify the physician when the patient arrives in the waiting room.

Tracking server configurator

Scenario: *The configurator of the system configures the routes and destinations in his office. He then needs to position several hundred scanner nodes throughout the hospital.*

The configurator starts by configuring routes and destinations on the tracking server. He can upload and align floor plans to help him with the positioning.

Afterwards, he needs to physically position and configure the scanner nodes throughout the hospital. Because it is impractical to bring and use a laptop while installing these scanners, the user can use his smartphone to configure the scanners on the tracking server. This is because the scanner nodes could be placed at any convenient location. False ceilings or next to already installed Wi-Fi routers. Most of these installations will require a ladder, meaning the use of a (small) laptop would be impractical and unsafe.

KWS system developer

Scenario: *The developers of the Clinical Work Station (internal application abbreviated as KWS) want to add a new feature where visitors of a patient can find the patient, they are looking for is not in his/her room.*

KWS developers need to be able to hook into the tracking server to track the location of certain users, i.e. their smartphone or asset tracker. This tracking server is not on the same machine as the KWS server and written in a different programming language. Due to the anonymity of the devices on the tracking server, it is the developer's responsibility to track and link devices to patients.

7.2. Software Communication

The system has a separate communication strategy for internal and external communication. Scanners and internal applications need a secure way of communication to avoid unwanted tracking of patients while the navigation app needs a public endpoint to receive navigation instructions.

Because internal applications need a scalable and language independent programmable interface, the choice was made to use socket-based communication between the server and internal applications.

This section describes the strategy used for the internal communication, i.e. MQTT communication (section 7.2.1), as well as the strategy for the external communication, i.e. WebSocket communication (section 7.2.2).

7.2.1. Message Queuing Telemetry Transport (MQTT)

MQTT is a ‘publish and subscribe' TCP protocol. It consists of a broker that allows clients to connect and publish their payload to a specific topic or channel (OASIS, 2014). Clients can also subscribe to specific topics to receive payloads send by other connected clients.

Used in many internet of things (IoT) devices or even other tracking systems like Accuware’s beacon tracker (Accuware Inc., 2018), MQTT allows for a simple communication protocol with minimal overhead. The protocol has very small headers and can be implemented on top of already existing socket connections such as WebSocket’s. In this thesis it will be implemented as a standalone TCP broker on our tracking server.

Allowing both the scanners and internal applications to use the same communication channel allows for direct communication between those internal applications and scanners as well. Internal applications can subscribe to topics that are normally used for communication between scanner nodes and the tracking server. This allows for an API interface without having to implement a new endpoint.

Subject filtering

The topics that can be subscribed on can contain wildcards. This gives clients that connect to the broker more freedom in specifying the topic they want to listen to.

MQTT topics are usually formatted with forward slashes separating the topic subject from the actor. This provides a hierarchical separation of subjects.

Quality of service

As documented on HiveMQ (2015), a framework for developing IoT devices, MQTT offers three tiers of quality of service (QoS). This is an agreement between client and broker to guarantee the delivery of published payloads.

- **At most once (QoS 0):** A message is sent without confirmation. It will arrive at most once.
- **At least once (QoS 1):** A message is sent and waits for confirmation. If the confirmation is not received by the sender within a certain time, it will be sent again.
- **Exactly once (QoS 2):** A message is exactly sent a single time. This is achieved by acknowledging the transmission twice using a separated PUBREC and PUBCOMP packet.

Latency

Publish-Subscribe patterns are usually not real-time. Payloads send to topics have to be broadcasted to all subscribed clients. The tracking server should always have a higher priority on the results published by the scanner nodes (clients) than the internal applications.

This problem was solved by intercepting the payloads on the MQTT broker. The tracking server is not really a client of the broker but hooks into the included broker to handle the messages before they are being broadcasted to other clients.

Protocol

The MQTT protocol gives information about the raw information originating from the scanners and the computed location of the devices based on multiple scanners.

Table 2 - MQTT scanner API protocol

Scanner topic	***Description***
scanner/<SCANNER_MAC>/scan	Scan results are posted in JSON format to this topic
scanner/<SCANNER_MAC>/ota	Scanner nodes use this topic to listen for OTA updates

Table 3 - MQTT device API protocol

Device topic	***Description***
device/<DEVICE_ID>/loc/raw	Raw location updates (JSON with coordinates and floor number)
device/<DEVICE_ID>/loc/route	Computed location based on nearby routes
device/<DEVICE_ID>/nav/destination	Navigation destination changes
device/<DEVICE_ID>/nav/update	Navigation update event sent to the mobile device

Shown in Table 2 and Table 3 are the topics that are publicly available to internal applications. The scanner topics contain the raw scan results that are used by the tracking server. The device topics are sent by the tracking server with the following information:

- **Raw location**: This is the calculated location without any optimizations or route snapping (see section 9.3.6).
- **Route location:** This is the location projected on the route. This is the actual position that will be used for the navigation.

- **Navigation destination:** The destination of the device, updated when the user changes this destination or terminates the navigation.
- **Navigation update:** Similar data that is sent to the mobile application. It contains formatted instructions.

The tracking server has no information on the link between a “DEVICE_ID” and patient information. Internal applications have full control in how they handle the linking for privacy reasons. If for some reason the MQTT broker communication is intercepted by unauthorized users, they have no means to link this to patient information.

7.2.2. WebSocket Communication

Communication between the server and navigation app is done using WebSockets. These allow for live and low latency messages from server to client. This server to client connection is required because the *tracking server* is the actor that computes the location of the user. It will generate navigation instructions based on these locations and send it to the navigation app.

In addition, the mobile device will use this live connection to send the compass heading of the smartphone’s sensor. This heading can be used by the tracking server to provide more precise instructions or noise filtering. An example would be to instruct the user to turn around, so he/she is facing in the correct direction.

WebSocket connections are made on top of the HTTP application layer allowing for easy implementation of the secured (SSL/TLS) connection between client and server (Fette & Melnikov, 2011).

While it would be possible to use the MQTT broker to get navigation updates, it would pose a security issue because this broker is used to communicate the whereabouts of all patients in the hospital. An intruder that manages to find the login credentials of the API or an exploit in the broker can access this tracking information from outside UZ Brussel.

7.3. Scanner Node Architecture

Scanner nodes have an important task, but their architecture widely depends on the hardware. These scanners will continuously scan for Bluetooth devices and transmit the information of these devices to the tracking server. In order to achieve accurate navigation, the scanner hardware and software must meet the following criteria:

> **Bluetooth sensitivity**
>
> The sensitivity of the Bluetooth signal must be as high as possible to be able to receive transmissions from longer distances. A longer distance allows for multiple overlapping scanners that can be used to increase the accuracy of the trilateration between the smartphone and scanner nodes or the ability to have fewer scanners in the hospital.
>
> **Discovery results**
>
> Bluetooth remains a short-range communication, meaning a moving device might only remain in its range for several seconds. The amount of discovery results of a same device needs to be as many as possible to track the movement of the device and filter out inaccurate readings (see section 8.2.5).
>
> **Server connection**
>
> Results received from the discovery are computed on the server. These results need to be sent to the server as fast as possible without blocking the scanning (as this would limit the amount of discovery results). This throughput needs to be equal to the speed of the discovery results to prevent any data from getting lost.
>
> **Rapid (prototype) installation**
>
> With the project being a master thesis that is still in development, the hardware needed to be easily installable in various locations. This meant that the devices would have to connect to the server through Wi-Fi to limit the amount of cabling and network configuration.

Apart from these technical criteria, the cost is also an important issue as many of them are required. This added the additional criterion for a price that should be lower or equal to beacons. As mentioned in section 3.2.1, beacons are currently the most commonly used method of creating indoor positioning systems. The price of beacons was used as a reference for deciding on the hardware.

7.3.1. Hardware

With the technical and price related criteria in mind, the choice was made to use ESP32 development boards for this thesis. Created by the Chinese company Espressif, these ESP32 micro controllers are the successor of the widely used ESP8266 in low cost or DIY Wi-Fi based IoT applications (Van de Wynckel, 2016).

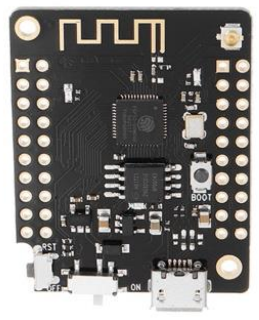


Figure 11 - Wemos TTGO Mini ESP32 Development board. (Banggood.com, 2018)

The *Wemos TTGO ESP32 Mini* was chosen as the development board. This is a very small PCB (printed circuit board) with a PCB antenna and connector for an external antenna. The external antenna can be used to improve the connectivity between the Wi-Fi access points and Bluetooth scanning.

At the time of writing, these were purchased for less than 6 euros per piece including an omnidirectional antenna. The included battery connector was used for a small batch of scanners to allow for rapid installation in a building. These batteries were originally meant for a specific brand of Quadcopters and were purchased for 3 euros per piece.

If this microcontroller were ever used in a real product, many of the components such as the PCB antenna, unused GPIO pins (Input Output pins at the two sides of the development board), battery connector, battery charger and unused LED’s could be removed to make the board considerably smaller and more power efficient.

Apart from the possibilities to use both Wi-Fi, Bluetooth classic and Bluetooth Low Energy; the ESP32 has the following specs:

Table 4 - ESP32 hardware specifications

ESP32			
CPU	Dual-core 32-bit	(240 MHz, 160 MHz)	
RAM	520 KB SRAM		
Flash	4MB		
BT Classic/BLE	12 dBm TX	-97 dBm RX	
Wi-Fi	20.5 dBm TX		
Security	1Kbit eFuse	Flash protection	Secure boot

(Espressif Systems Ltd, 2018a)

The dual-core processor allows for parallel Bluetooth discovery and result computation. With the 4MB flash it is possible to have ~2MB large programs that can support Over the Air updates (OTA updates). This allows the hardware to download a newer version of its firmware on the second part of the flash before installation. A small section of the flash must be dedicated to SPIFFS (Serial Peripheral Interface Flash File System) storage for settings such as the Wi-Fi credentials and server connection settings.

Other specifications such as the 1024-bit one-time programmable memory (electronic fuse) can be used to enable and configure security features to prevent unauthorized reading of the privately stored settings.

7.3.2. Software Implementation

The ESP32 hardware is natively programmed in C using the ESP-IDF framework (Espressif Systems Ltd, 2019a). This framework can be accessed in various programming languages and frameworks such as Arduino, JavaScript or Python. Because several low-level Bluetooth functionalities had to be changed in the implementation, the scanners were programmed in C++ with C bindings to the ESP-IDF framework.

Table 5 - ESP32 core utilization

Core 1 (240 MHz)	**Core 2 (160 MHz)**
Bluetooth scan initializing	Bluetooth listening
Tracking server communication	Wi-Fi listening

The two cores of the ESP32 were fully utilized. The fastest core (240 MHz) was used to initiate the Bluetooth scanning and MQTT tracking server communication. The 2nd core was completely dedicated for the RF-radio to listen (and buffer) incoming and outgoing Bluetooth and Wi-Fi communication. For more in-depth information on how this communication was shared on the same radio, see section 8.2.1.

7.4. Tracking Server Architecture

The tracking server is the central place connecting all scanners and patient's devices using the navigation application. It offers the maintainer of the system an easy to use Web UI for the configuration and handles the positioning of users through scan results from several hundred scanner nodes.

Other than connecting with scanners and navigation applications it needs to allow the tracking server configurator to create routes and position scanners. This will be done through a Web application that is served by the tracking server.

It needs to support three types of communication; the communication between scanners, communication between mobile applications that are navigating in the hospital and communication between internal applications. For this reason, the server needs to be scalable and possibly decentralized over different hardware for the case that the need arises for handling more requests.

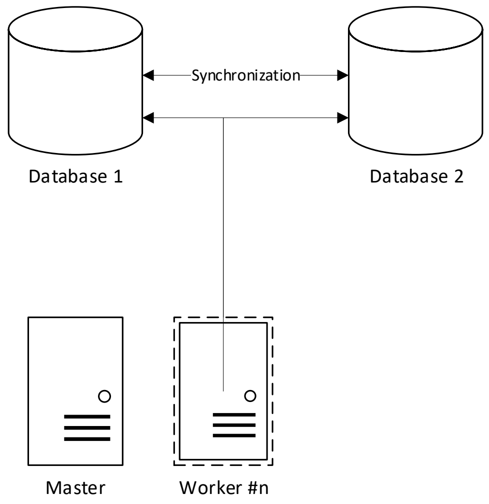


Figure 12 - Tracking server architecture

The architecture of the tracking server is shown in Figure 12. Two (MySQL) databases are set-up as a single load balanced database that is accessed by multiple workers. They contain all the persistent data of the system ranging from the latest readings of scanners to login sessions of the configuration.

These workers, spawned by the 'Master' server balance the server load that handles Web requests, MQTT connections and WebSocket connections.

The 'Master' server is responsible for synchronizing cached data between workers. Cached data mainly includes the shortest paths for all destinations, when this is updated on one worker – it needs to be synchronized between all other workers.

For practical reasons, Node.JS (Node.js Foundation, 2019) was used to create the tracking server. It provides various libraries that can be used to perform rapid prototypes of the implementation. It is a framework that is written in JavaScript, closing the gap between front-end and back-end development.

Node.JS is a single threaded language with the capabilities to share the same socket ports between processes. To conform to the aforementioned worker architecture, multiple processes need to be spawned that can all listen on the same server port.

7.5. Navigation App Architecture

The navigation application is only meant to start the BLE advertising and showing the instructions that are sent by the tracking server.

Apart from these two tasks, it is merely a way to provide instructions to the end-user without having to do any calculations or tracking.

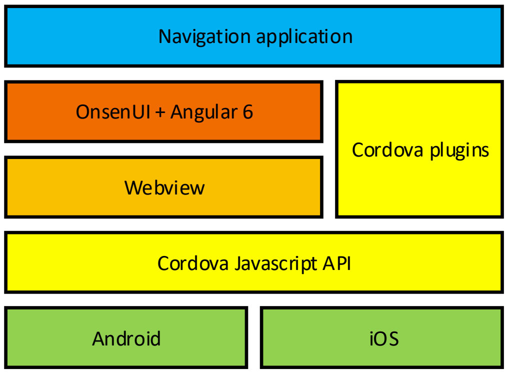


Figure 13 – Smarphone navigation application architecture

The choice was made to program the application as a hybrid application that runs on both iOS and Android. It's built upon the Apache Cordova JavaScript API that acts like a web browser (Apache Software Foundation, 2015). This allows for applications to be written in HTML, CSS and JS just like any Web app. It is a standardized format that can be run on most phones or even on computer operating systems.

To access the phone's OS specific features such as the BLE API, plugins can be used. They are programmed individually for all supported operating systems and provide a single JavaScript interface to control them. This results in an almost WORA (Write Once, Run Anywhere) environment where the application can be deployed to any platform without much effort.

OnsenUI and Angular 6 were used on top of Cordova as the UI framework. It provides the styles for both Android and iOS to follow the design guidelines (Monaca & Onsen UI Team, 2019).

Communication to the tracking server is done through REST API requests for getting basic information such as the available destinations and WebSocket's (see section 7.2.2).

7.6. System Security

This section of the system design explains the steps taken to assure the system can be deployed safely in a hospital environment.

The main security goal was to ensure that the system could be protected in various ways. The following choices made regarding the scanner hardware, communication protocols and architecture contributed to this goal.

- **Encrypted server communication**: All server communication (navigation app to server, scanner to server and web server) is encrypted using TLS. This is done for both the MQTT connections, WebSocket connections and tracking server Web application.
- **Firewall**: The internal communication of the tracking server configuration and MQTT server are separate from the publicly available REST API and WebSocket server that is used by the navigation applications.
- **MQTT authentication:** Scanners and 3rd party clients that connect to the MQTT server need to authenticate.
- **Scanner settings encryption**: With hundreds of scanners being scattered throughout the building, it is possible some of these scanners can get missing or stolen. ESP32 offers the possibility of secure flash storage. This prevents the extraction of stored settings such as the Wi-Fi credentials and MQTT authentication. Its secure booting functionality also prevents any malicious user from changing the program on a scanner to circumvent the protected storage.

8. Scanner Nodes

The chosen Bluetooth tracking positioning requires 'Bluetooth scanners' to be placed throughout the hospital. These scanners will continuously scan for Bluetooth devices and transmit the information obtained by these devices to the tracking server.

An ESP32 development board was chosen as the hardware for these nodes (section 7.3.1). These boards provide a relatively cheap and expandable development platform that can be programmed in various languages with various third-party libraries.

This chapter of the thesis contains a few important aspects related to the research of the scanner nodes. Section 8.1 starts with how these scanners would be installed in a building. Next, section 8.2 explains how these installed scanners can discover Bluetooth devices. Lastly, section 8.3 explains what information about these discovery results is transmitted to the tracking server.

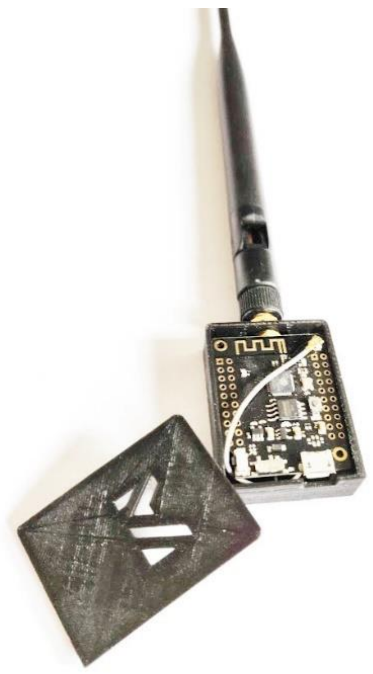


Figure 14 - Wemos ESP32 development board inside a 3D printed prototype casing

Figure 14 shows the development board inside a 3D printed prototype casing. These prototypes were used during the testing of the implementation.

8.1. Enclosure and Installation

One of the main advantages of Bluetooth beacons is its ease of installation inside a building. They are small and light, do not require a large antenna and can be battery powered for 1-2 years depending on the manufacturer. This allows them to be placed on ceilings or walls with some adhesive. However, the scanner nodes used in this system require a constant source of power because of the required Wi-Fi/Ethernet connection to the tracking server. This limits the freedom of placing the devices in locations that have an electrical source such as ceiling lamps, routers with USB ports, AC plugs or a false ceiling. On the other hand, because the scanners do not require yearly maintenance for battery replacement, their installation locations do not have to be easily accessible. The only requirement is that the RF interference does not cause problems for the range of the receivers.

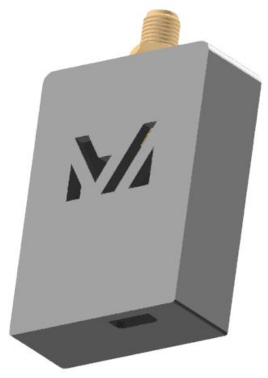


Figure 15 - 3D render of scanner node casing without battery (46mm long, 35mm wide)

The chosen hardware development board allows for an external antenna. This antenna increases the connectivity of both the Bluetooth scanning and Wi-Fi connection. While it is theoretically possible to connect any kind of antenna or even use the small antenna used on the PCB (Printed circuit board), the choice was made to use an omnidirectional 50 Ohm antenna (Connector shown in Figure 15 as the copper connector).

An omnidirectional antenna radiates a torus shaped RF pattern. Because of our interest in 2D positioning, these antennas should be placed vertically to obtain a 360-degree radiation on the same floor (Cisco Systems Inc., 2007). The hardware requires Wi-Fi connectivity throughout the hospital. This is to lower the logistical cost of connecting all these scanners with network cables.

For a select batch of prototyping scanners, the development boards battery connecter was used to connect a small LiPo battery. Connected to Wi-Fi and scanning for Bluetooth devices, these batteries provide enough power to allow them being used inside a building for testing purposes.

8.2. Bluetooth Discovery

The two Bluetooth implementations (classic and BLE) were considered and tested for navigation purposes. BT classic has the advantage that it can be used on very old devices. BLE on the other hand has a better way of transmitting its awareness since BT classic would first need to transmit an inquiry before the mobile device starts transmitting its response. This means that the scanner node would act as both a transmitter (for the inquiry) and a receiver (for the response).

8.2.1. Wi-Fi and Bluetooth Coexistence

To keep the hardware cost as low as possible, many IoT devices share the same RF radio for both Wi-Fi and Bluetooth (Wetzker, Splitt, Zimmerling, & Römer, 2016). The radio is shared between both BT Classic, BLE and Wi-Fi through a method called coexistence where radio usage times are distributed to all wireless communication protocols that require it. It is possible to have separated radios for both Bluetooth and Wi-Fi, but since they would be in very close (centimeters) proximity to each other and operate in the same radio band, it will inevitably cause interference.

Scanner nodes need to be able to publish their results as fast as possible but need to remain scanning for new devices. Prioritizing the scanning will result in more results, but if these results must be queued before they reach the server, they will no longer be relevant.

Several actions were taken to prioritize the Bluetooth scanning without letting them go to waste in a message queue:

- **MQTT QoS 0**: The scanner will not wait for an acknowledgement of the published results (at-most once). This limits the working time needed for the Wi-Fi radio (see section 7.2.1 for MQTT QoS levels).
- **Last result queue**: New results of devices that are already in the transmission queue will update. No transmission will be sent to the outgoing buffer if the Wi-Fi driver is still waiting. In section 8.2.5 it is explained that the queue update will use the updated result of a noise filter.
- **Result timeout:** If a result takes longer than one second to be transmitted, they will be dropped. They are no longer relevant and would negatively influence the position if they would be used by the tracking server.

- **SW coexistence balanced preference:** The radio will be shared equally for both Bluetooth (dual mode) and Wi-Fi. Both “BT priority” and “Wi-Fi priority” were tested but resulted in lesser relevant results.
- **Service type filtering:** The service UUID’s contains a prefixed part that is unique for all users. If the scanned device does not have this prefix, it will be dismissed (see section 8.2.3). This limits the amount of data that must be sent, allowing more room for device results that need to be tracked.

8.2.2. Discovery Scan Window

Because both implementations require a scan window where the device listens for responding or advertising devices, they cannot be run simultaneously. The coexistence would require them to be executed sequentially what will give a large gap of approximately 15-20 seconds between scans. This gap is equal to the recommended scan window for BLE devices (Bluetooth Special Interest Group inc., 2016).

Considered that hundreds of scanner nodes would send inquires every scan window, it would cause a lot of unneeded RF communication inside the hospital.

Market studies performed by the Bluetooth SIG also show that 97% of all devices shipped in 2022 will use BLE, while only 67% will contain BR/EDR (Bluetooth Special Interest Group Inc, 2018c). This led to the final decision to opt for BLE discovery instead of BT Classic.

BLE devices have fixed advertise intervals depending on the type of device. Low powered devices might want to limit the amount of advertisements to every 100ms to conserve battery, while other devices can advertise themselves every 10ms.

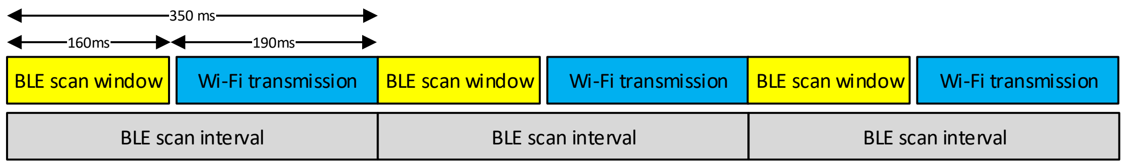


Figure 16 - BLE scan window and interval

Automated tests using the ESP32 concluded that the best ratio was to have a scan interval of 350ms with a switching scan window of 160ms that changes the listening channel (shown in Figure 16, the scan interval containing both the scan window (yellow) and Wi-Fi transmission period (blue)). This test was done by positioning 10 BLE advertising devices within a 5-meter radius of the ESP32. First, the number of advertise packets that could be captured was maximized by gradually decreasing the scan window and interval.

Because the first step of the test did not require Wi-Fi transmission, the scan window was equal to the scan interval. It started with 1 second and decreased in 5ms steps every 10 seconds. This resulted in a maximized scan interval of 346ms (it was rounded to 350ms).

With that scan interval/window in mind, the next step was to maximize how many of these results could be sent to the server before the queue was unable to keep up. For this, the scan window was lowered by 5ms every 10 seconds, giving the Wi-Fi controller 5ms more time.

The result of this final step was to give the Wi-Fi controller 190ms time to transmit the data. While this test will not always be perfect due to interference, more than ten devices or other factors – it does provide a good model for the scan and interval window that proofed itself during further testing.

8.2.3. Service Fingerprinting

Recent iOS versions restrict programmatic access to the Bluetooth MAC address. There are still methods to circumvent this restriction, but this would be against the App Store policy. Apps circumventing the restrictions set by Apple might not be accepted to the public app store. Google made similar changes in Android 6.0 because they wanted to provide the user with a more detailed explanation of why an application would want to know the device's MAC address. Therefore, Google made the decision to require applications to get the "device location permission" if they wanted to know the MAC address. They felt that the MAC address could be used by applications for tracking purposes (which is correct in this case).

Both BT Classic and BLE can create services (Bluetooth Special Interest Group inc., 2016). These services indicate the possible uses for the Bluetooth device, such as audio playback, messaging or in this case location tracking. For BT Classic these are broadcasted in the extended inquiry response and for BLE these are send within the advertise packets. Normally services are used in addition to a connection listener, so clients can connect to specific services. In this implementation the service identifiers are merely a quick way to identify devices that need to be tracked.

Service UUID composition

The 128-bit service identifier (UUID) is an identifier composed of 5 parts. See below for two examples. The last 4 parts are dedicated to uniquely identify the mobile device based on the hardware and are randomly generated. The first part (blue) is unique for all devices that use the application.

Example UUID for user 1:

94854821	**332B**	**4612**	**BD40**	**428963099B28**

Example UUID for user 2:

94854821	**35F6**	**11E9**	**B210**	**D663BD873D93**

Any user that uses the navigation app will broadcast an UUID whose first part begins with ‘94854821’ (in this example). The gray part in the above examples is unique per user and can be used to track the device throughout the building.

Instead of sending every scanned device to the tracking server, the scanner can filter on devices that use a service UUID that starts with the same first part. In the example above, the blue first part of the two UUID’s is the same. It identifies that the service is created by the navigation app.

8.2.4. RSSI to Distance

A mathematical positioning method determines the location based on the distance between reference points (scanners). To determine the location of the mobile device, the received signal strength needs to be converted to a similar scale such as the known distances between scanners.

Converting the RSSI to a distance is not accurate but using a *Path loss formula* it can be estimated. The path loss predicts how much of the initial transmitted power gets lost in the air. Assuming the signal travels through air and has several obstructions (people walking through the hallway, walls, doors) this loss can be expressed in meters.

The following formula taken from the paper by Dong & Dargie (2012) determines the received signal strength ($RSSI$) at a specific distance in meters (d). It can be determined using a reference signal strength that was measured at a 1-meter distance (A) and a signal propagation constant (n) that specifies how much of the signal will get lost in the air.

$$\mathrm{RSSI} = -(10 \times \mathrm{n}) * \log10(\mathrm{d}) - \mathrm{A}$$
$$\rightarrow d = 10^{\frac{\mathrm{A} - \mathrm{RSSI}}{-10\times\mathrm{n}}}$$

The reference signal strength over a 1-meter distance depends on the smartphone’s transmission power. This is an area were beacons would have an advantage because they have manufacturer calibrated reference measurements that are broadcasted in the advertisement packets. However, for either implementations it remains a calibrated and estimated measurement that is vulnerable to errors. The goal is to achieve an RSSI-to-distance formula that is as linear as possible for a given RSSI interval.

During the configuration and calibration of the scanner, the reference signal will be made at a 1-meter distance over 150 measurements (A). The signal propagation distance (n) will be set to 2.4 by default but can be changed on the tracking server.

This method of calibration was also used in the paper by Sung (2016). A more in-depth explanation on how the configuration is done on the tracking server can be found in section 9.2.2.

8.2.5. RSSI Noise Filtering

The strength of a received signal fluctuates depending on physical obstructions or RF interference of other devices operating on the ISM band.

Ultimately, these fluctuations result in exponentially greater errors when converting the RSSI to a distance with the formula given in section 8.2.4. This would make positioning unreliable and very inaccurate.

This section explains the steps that were taken to construct a noise filter that can help filter out this interference. Shown in Figure 17 are the readings of a stationary device at 1, 2, 4 and 8 meters away from a scanner node. These observations (indicated in red) show how noise influences the RSSI value based on the distance between the scanner and smartphone. The black line in the figure indicates the average RSSI value over all measurements.

The data of this experiment was used to construct a simulation to aid with the creation of this noise filter. This simulation allows testing the noise filter on simulated moving objects instead of stationary positions as used in the aforementioned experiment.

Lastly, using a modified Kalman filter, this simulation helped with the creation of a noise filter that can smooth out outliers without the need for additional sensor data.

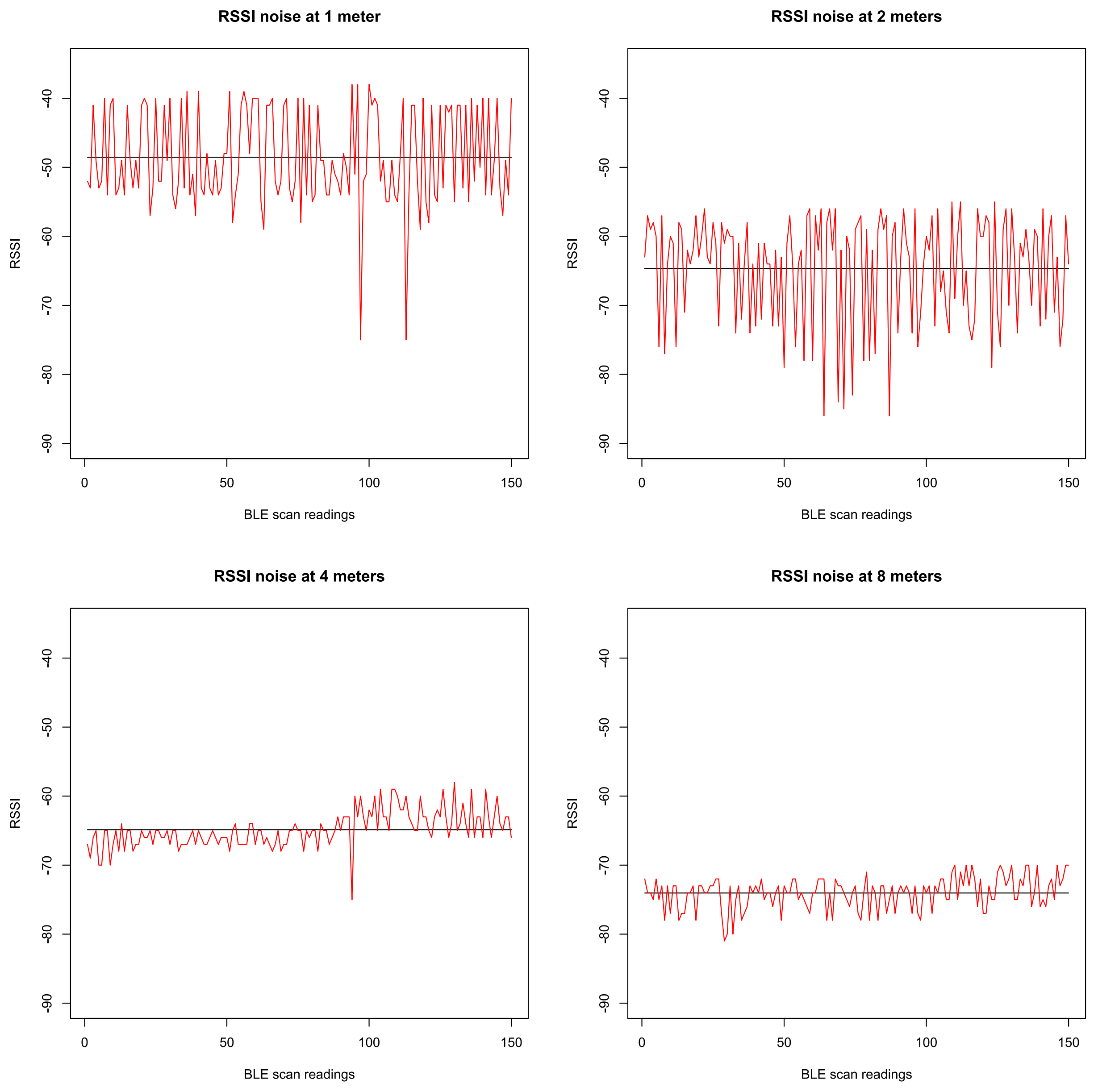


Figure 17 - RSSI noise measurements over 1, 2, 4- and 8-meter distance

The measurements were conducted in an empty hallway with the mobile device and scanner being on the same height (80cm). During these measurements, the scanner was positioned away from other Bluetooth or Wi-Fi radios apart from the mobile device that was broadcasting these signals.

Table 6 - RSSI Noise experiment data

RSSI Noise experiment	***RSSI Average***	***RSSI σ***	***Calculated distance σ***
1 meter	-48.55	7.06	3.60m
2 meters	-64.65	7.82	22.00m
4 meters	-64.86	2.56	3.14m
8 meters	-74.04	2.32	9.46m

The data collection in the RSSI noise experiment and information in the paper *Method for Improving Indoor Positioning Accuracy Using Extended Kalman Filter* by Lee, Lim and Lee (2016) were used to construct a simulation for performing a Kalman noise filter on a moving signal.

Table 6 shows that the standard deviation of the RSSI is not linear with the distance of the scanner and mobile device. To aid with the construction of a noise filter, a simulation was created that would be used to test and tweak the filter. This simulation can quickly be altered to test different scenarios such as different movement patterns or different noise levels.

With the knowledge that the average scans per minute is 150, the expected maximum radius of the scanner is 20 meters and the average walking speed is 1.4 m/s, the following simulation parameters were chosen.

Table 7 - Noise filtering simulation parameters

Simulated noise experiment		
Transmitter radius	20 meters (max)	
Avg. readings per minute	150	(BLE readings received per minute)
Avg. human indoor walking speed	1.4 m/s	
Simulation duration	30 seconds	(Rounded based of $(2 * radius)/1.4$)
Simulation readings	74	
Simulation movement	-20 to 20 meters	74 readings between -20 and 20 meters
Noise generation	$Variable$	Noise decreases over distance

The simulation was created in R by creating a linear distance from -20 m to 0.01 m and 0.01 to 20 meters over 74 readings. This distance was then converted to an RSSI value using the derived formula used to convert RSSI to distance (Dong & Dargie, 2012). For this conversion a reference signal (A) of -48 obtained from the 1-meter average in Figure 17 was used combined with a propagation value (n) of 1.6.

Noise was generated by creating a logarithmical function based on the distance formula that takes the distance and outputs the standard deviation for that distance. It was created by taking the convex function of two points (1 m and 8 m).

$\sigma_{1m} = a * \log_{10}(b * 1)$

$\sigma_{8m} = a * \log_{10}(b * 8)$

Both a and b are multipliers. b is used to specify the

$$a = \frac{\sigma_{1m} - \sigma_{8m}}{\log_{10}\left(\frac{1}{8}\right)} \ ; b = 10^{\left(\frac{\sigma_{8m} * log_{10}(1) - \sigma_{1m} * log_{10}(8)}{\sigma_{1m} - \sigma_{8m}}\right)}$$

This gives us the following formula for estimating the standard deviation σ_d of a certain distance between scanner and smartphone (d).

$$\sigma_d = a * \log_{10}(b * d) = \left(\frac{7.06 - 2.32}{\log_{10}\left(\frac{1}{8}\right)}\right) * \log_{10}\left(10^{\left(\frac{2.32 * log_{10}(1) - 7.06 * log_{10}(8)}{7.06 - 2.32}\right)} * d\right)$$

The standard deviation was then used in a randomize function to generate a noise between $[-\sigma_d, \sigma_d]$. In addition, outliers (or spikes) to the noise generation function were simulated by adding a multiplier of 2.3 to the standard deviation on 15% of the results. This was based on the outliers at certain distances.

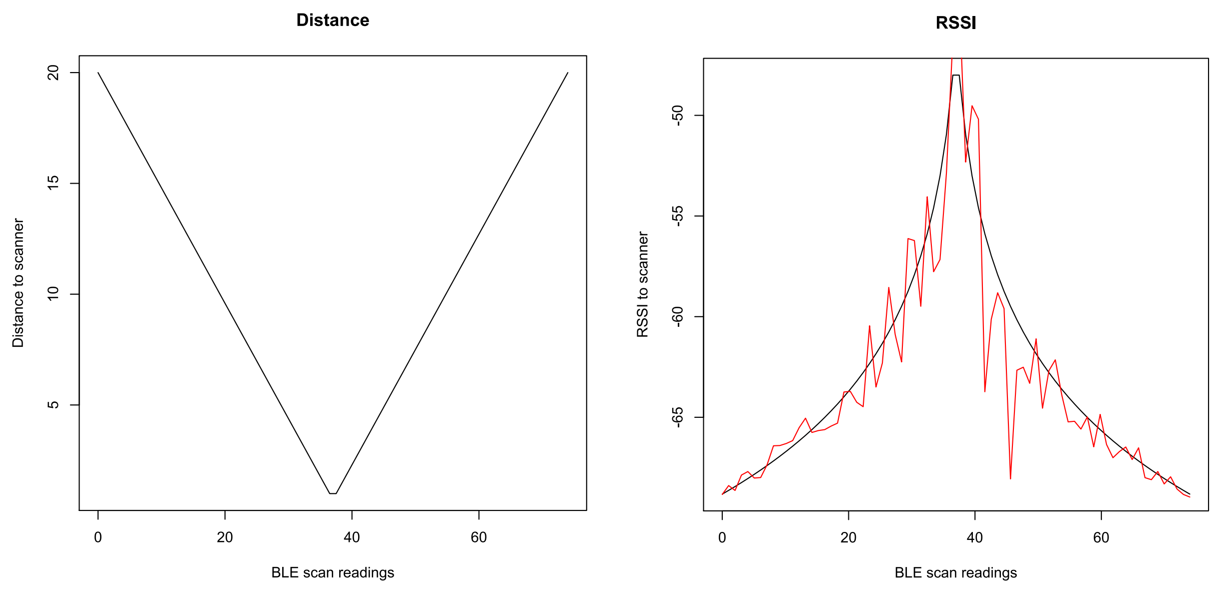


Figure 18 - Simulated Kalman filter experiment. Starting point distance and RSSI (with noise).

Figure 18 shows the simulation of a user walking inside a hallway. He/she starts from the farthest point towards the scanner until he passes the scanner and keeps moving away (left). At the right is the calculated RSSI with added noise using the constructed noise generation formula ($\sigma_d(d)$).

Basic Kalman Filter

The chosen noise filtering Kalman filter is based on the basic Kalman filter. As explained by Levy (2016), this filter assumes that scanners observe a reading at a specific time ($observed_k$) with the actual reading being known as $real_k$. However, because our readings are not 100% accurate, the observed reading is the actual reading with the addition of possible noise that might occur ($noise_k$).

$$observed_k = real_k + noise_k$$

<u>Model</u>

Kalman filters use this assumption to predict and correct the next result (x_k). This result evolves with each new reading based on the previous value (x_{k-1}) and multiplying it with the state-transition model (A). This state transition model defines how x progresses to the next value for each reading. Because this state transition is prone to errors, additional processing noise is added (w_k). It is assumed that this noise is normally distributed as $w_k \sim \mathcal{N}(0, R)$ with R being the covariance of the processing noise introduced by the state transition and optional control input.

$$x_k = Ax_{k-1} + \boldsymbol{Bu_k} + w_k$$

The formula can contain a control input model B and a control vector u_k. These can provide additional information on the evolution of the transition. For example, it could be used to specify an acceleration at time k. Because our scanner nodes do not have any other sensor data on the movement of the user, these control inputs $B\ with\ u_k$ are not used in this implementation.

The observed signal at time k is denoted as z_k. This signal is the real value (x_k) with the addition of sensor noise (v_k). Just like the processing noise in the model (w_k) this sensor noise is normally distributed as $v_k \sim \mathcal{N}(0, Q)$ with Q being the covariance of the sensor noise.

$$z_k = x_k + v_k$$

Using this model, we can predict and correct the next value based on the transition of previous readings.

Prediction and update

The predicted state estimate will predict how the previous value progresses to the current value by multiplying it with the state model. This $\hat{x}_k$ is denoted with a hat to indicate a predicted value (Levy, 2016).

$$\begin{aligned} &Predicted\ state\ estimate: && \hat{x}_k = Ax_{k-1} \\ &Predicted\ error\ covariance: && \hat{P}_k = AP_{k-1}A^T + R \end{aligned}$$

This predicted state $\hat{x}_k$ will be compared with the measured value z_k. This is done by updating the Kalman gain (the gain of the transition from $k - 1$ to k) (Bulten, Rossum, & Haselager, 2016).

$$K_k = \hat{P}_k * (P_k + Q)^{-1}$$

$$x_k = \hat{x}_k + K * (z_k - \hat{x}_k)$$

$$P_k = \hat{P}_k - (K_k * \hat{P}_k)$$

The basic Kalman filter assumes that $R, Q\ and\ F$ are predefined models that gives insight on the evolution of x. As can be seen from the noise experiment in the previous section, the error and RSSI change depending on the distance. Papers such as the one from Sung (2016) try to solve this issue by performing the Kalman filter on the already converted distance. This solves the issue of the predicted result not updating fast enough but does not account for the noise severity based on the distance. Eventually this causes the distance readings to fluctuate significantly.

Using the simulation formula (σ_d) create in the first part of this section, several different scenarios were simulated. These simulations either show a smooth distance that has a 5-meter accuracy or a highly fluctuating filtered distance when the measurement and processing noise was kept low. While tweaking these errors may give better results in the simulation, they will never be able to give an accurate prediction on a real-world scenario. In the next section this Kalman filter is modified to have a dynamic error and state-transition model. This additional change can help to predict the error based on the expected distance.

Modified Kalman Filter

Many variations on the basic Kalman filter exist. One of these variations is an Extended Kalman Filter (EKF) (Levy, 2016). It uses other sensor data to predict the results. For example, if the distance changes, but the accelerometer of the phone shows no change – it is most likely noise.

As mentioned previously in this chapter, scanners do not have any other sensor data of the mobile device. A possible solution for the noise filtering would be to completely move the filtering to the tracking server. The tracking server will have plenty of information about the mobile device such as the results from other scanners, the compass heading of the mobile device and maybe even the accelerometer. However, this would mean that the throughput of scan results to the tracking server will have to be increased because no filtering is applied. Instead, the noise filtering will be done in two stages: A simple noise filtering on the scanner nodes and a more advanced and optional filtering on the tracking server that uses other available data. The modified Kalman filter discussed in this section will be a small modification of the basic Kalman filter with additional assumptions on the model/movement:

- Assumption of traveling speed: It is assumed that a patient will not walk faster than 1.4m/s. While they can run, it would most likely not be while they are looking at their screen.
- Assumption that the converted distance is linear
  - This means that it is assumed that the RSSI to distance formula is flawless
  - This means that it is assumed that the user walks with the same speed
- Min and max RSSI. The RSSI has a known minimum and maximum. These boundaries will be considered when predicting the next result.
  - When nearing the scanner (reaching the RSSI for the minimum distance) the RSSI will never go beyond a specific minimum RSSI.
  - When the device is moving away from the scanner, it will not go past the boundary for the maximum distance, causing the converted distance to be impossibly far.

Combined, these assumptions mainly involve the error covariance of the measurement noise. If this covariance is changed to a function that changes on the 'expected' distance, it can smooth the RSSI value before it is converted to a distance.

$$Measurement\ noise: \qquad Q(x) = \sigma_d\left(dist(\hat{x}_k)\right)$$

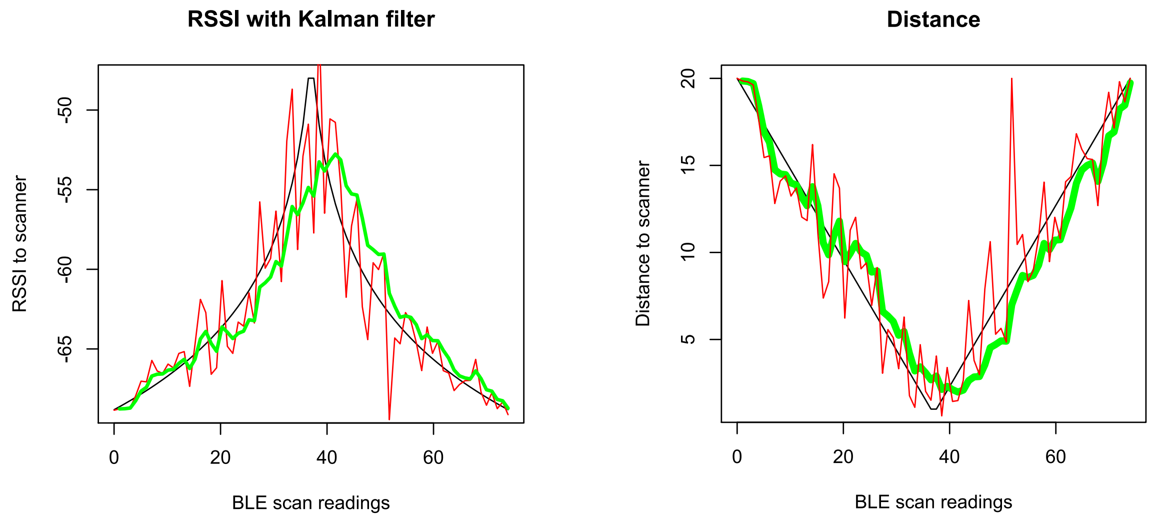


Figure 19 - Filtered RSSI (green) and converted distance using the modified Kalman filter

Figure 19 shows the modified Kalman filter with the dynamic measurement noise ($Q(x)$) applied to the noise example given with Figure 18. The green line shows the filtered RSSI and converted distance over BLE scan readings with simulated noise (red).

8.3. Tracking Server Communication

Once the scan is received and noise filtering is applied, a scan result packet is created and send to the tracking server.

The scan result packet contains important information shown in Figure 20. This information, encoded as a JSON string can help the server determine a more accurate position:

- **mac:** The MAC address of the scanned device
- **name:** The public Bluetooth device name
- **tx:** Transmission power if included in the advertisement packet (-1 when not included)
- **ble:** Flag that determines if the scan result was obtained using BLE
- **spm:** Scans per minute for the scanned device
- **scans:** Scan counter. Increases for each scan for the scanned device
- **service:** The Service UUID as mentioned in section 8.2.3
- **rssi:** The raw and Kalman filtered received signal strength
- **distance:** The calculated distance based on the calibration settings. The distance is calculated for both the raw and filtered RSSI

```
{
  "mac": "45:c3:e9:19:78:4e",
  "name": "",
  "tx": -1,
  "ble": true,
  "spm": 105.9,
  "scans": 141,
  "service": "94854821-fba4-3a93-8405-a448e0c5be61",
  "rssi": {
    "raw": -55,
    "filtered": -55
  },
  "distance": {
    "raw": 0.54,
    "filtered": 0.54
  }
}
```

Figure 20 – JSON scan result packet example

While most of this information is stored on the tracking server; only the service UUID, filtered distance, filtered RSSI and transmission power is used to determine the position. If the transmission power is available, it will be used to provide a more accurate distance.

9. Tracking server

The tracking server is the central place connecting all scanners and patients using the navigation application. It offers the maintainer of the system an easy to use Web UI for the configuration and handles the positioning of the end-users through scan results from several hundred of scanner nodes.

The server needs to support three different types of communication: the communication with the scanners, the communication with the mobile applications used for navigating in the hospital and communication with internal applications.

As mentioned in the system design at section 7.4, the tracking server also hosts a Web application that is used to configure the navigation system. See Figure 21 for the welcome screen of this application.

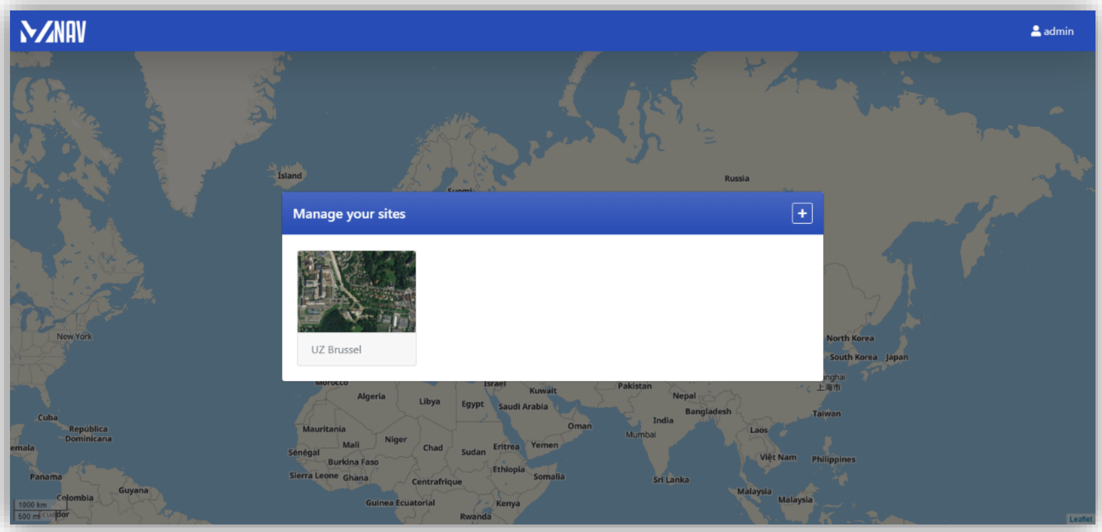
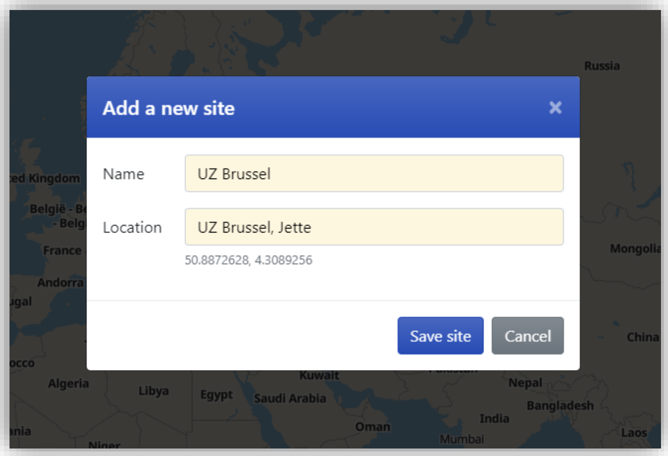


Figure 21 - Welcome screen of the tracking server configuration, screenshot taken from the implementation

9.1. Design Choices

Regardless whether the user is configuring routes, placing destinations or configuring scanner nodes, they will always work with (or on) a map or floor plan. After the first meeting with UZ Brussel and other possible users, it became clear that configuration interface should be like a "Google Maps" layout and UI.



Name: UZ Brussel

Location: UZ Brussel, Jette

50.8872628, 4.3089256

Figure 22 - Screenshot UI - creating a new site

Google Maps uses a map centric design. All user actions concerning the map blend in with the UI.

They do this by creating a layered based UI, with the map always being visible in the background.

Therefore, the indoor navigation configuration UI always has a map in the foreground or background. This not only provides consistency throughout the configuration, but it also assures that user has context on what (or where) he is editing something.

The configuration should be done based of the hierarchical decomposition of the concept *Site*. Each of these objects can be configured in any order, but the hierarchy needs to remain.

- **Site**: *A company can have multiple sites in different locations. Each site can contain multiple (connected) buildings, however the route creation is abstract about this and does not distinguish between buildings.*
    - **Floor**: *Each site can have multiple floors.*
        - **Floor plans**: *Each floor can have one or more floor plans (images). They can be georeferenced to existing outlines available on OpenStreetMap (see section 9.2.1)*
        - **Routes**: *Each floor has a network of routes constructed as a graph (see section 9.2.3).*
        - **Scanners**: *Each floor can have multiple scanners.*
        - **Destinations**: *Each floor can have destinations.*
    - **Scanner profiles**: *Scanner profiles are Wi-Fi settings. A site can have multiple profiles for different locations in the building.*
    - **Destination categories**: *Each site has its own hierarchical destination categories.*
        - **Available destinations:** *Available destinations that have not been assigned a floor will be visible when searching for a destination on each floor.*

Once a site is configured, the ground floor of the building is automatically created. Floor plans are an additional aid to help positioning the routes, destinations and scanner nodes.

This hierarchy becomes immediately visible when the user signs into the system. They are welcomed to a world map with the possibilities to “jump” to one of their company/building sites across the world. Once zoomed into a site, they can go up and down between floors where they can manage routes, scanners and their destinations.

9.2. Configuration of a Site

9.2.1. Configuration of the Floors

When a new site is created, the default floor of the site is the ground floor. The user can switch between floors or add new floors is done through the up/down buttons in the top right corner of the map. Each new floor can have its own routes, floor plans and destinations.

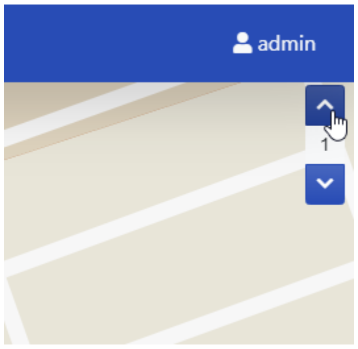


Figure 23 - Floor selector up/down

Indoor floor plans are not always included in Google Maps or OpenStreetMap. Google tries its best to include indoor floor plans of large public buildings, but there is no guarantee that they are available and accurate. However, the floor plans are required to configure routes and to position scanners and destinations. They must be reasonably accurate but above all should be usable by the administrating user to indicate the different destinations on the plan.

Floor plans of company buildings or public buildings such as UZ Brussel are available for various purposes. The IT department of the hospital has floor plans that show a detailed view of all access points (these are the floor plans that were used in Figure 24), AC plugs and network cabling. Floor plans are also available as evacuation plans and are mandatory in large public buildings such as hospitals. Therefore, evacuation plans were the focus when implementing georeferencing. They are always available because companies are required by law to have them. The goal was to be able to use a photographic picture or scan of one or more evacuation floor plans.

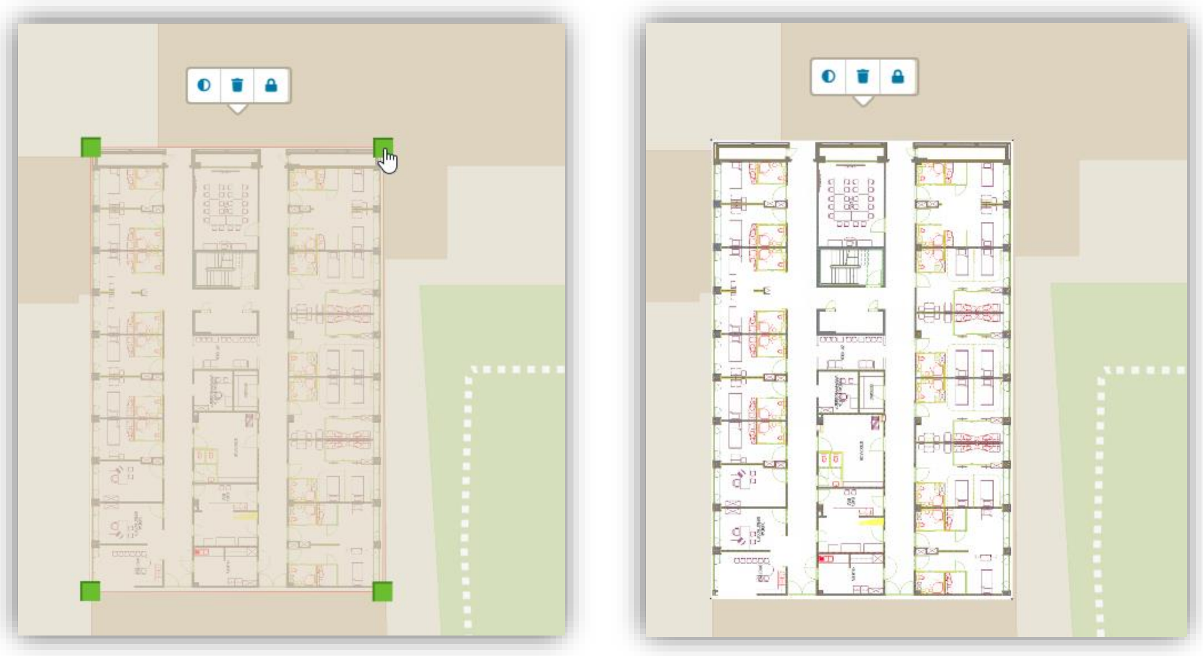


Figure 24 - Screenshots of georeferencing a network floor plan. Left while it is being positioned and right after locking it

After uploading a floor plan image, the user can drag, rotate and scale a transparent version of the image to fit on top of the existing OpenStreetMap map. The transparency helps lining up the walls with the contours that might already be available (See Figure 24 (left) for the transparency view).

By uploading multiple images per floor, the user can stitch multiple building sections together, forming one giant and detailed representation of the building floor.

9.2.2. Configuration of the Scanners

Scanners need to be configured and positioned in many locations throughout the hospital. This configuration needs to link the newly positioned scanners to the tracking server.

To make it easier for the user that is configuring these scanners, this configuration can be done through a modern smartphone or laptop running Google Chrome or Opera. It does not require an additional application and can be done directly on the tracking server Web application.

Because of the benefits of being able to configure the scanners without additional applications, the choice was made to use the Bluetooth Web API (Beaufort, 2018) to require the use of a Chrome (or Opera) for the installation. However, the API is still in its early stages of development and is not yet an official standard. Currently, Chrome for desktop, Android and Opera for desktop, and Android support this API (Google LLC, 2018b). Therefore, it is required to use Chrome (or Opera) for the installation. However, as only the tracking server configurator needs this scanner configuration, this is not a serious restriction. [1]

A configuration includes the Wi-Fi settings, server settings and RSSI calibration (see section 6.3.2). While the server settings will remain the same for all scanners in a building, the Wi-Fi and RSSI calibration can vary depending on the positioning of the scanner.

[1] Originally, the idea was to use this API also for the navigation on the smartphone. This would allow patients to use the existing patient portal to navigate to their destination. However, as this API is not yet supported by Safari on iOS devices. Furthermore, the API only supports the scanning and connecting of BLE devices, implying that the currently tracking method that lets the mobile phone broadcast advertisement packets would not be possible.

The configuration consists of the following steps:

- **Global Wi-Fi settings**: The "profiles" of individual Wi-Fi and server settings can be configured beforehand. One profile can be used by multiple (maybe even all) scanners throughout the hospital.
- **Placing a scanner**: Similar like configuring a route or destination the user can place a scanner by right clicking on the map.
- **Scanning and connecting:** The Bluetooth API required a user interaction to connect to a Bluetooth device. Therefore, user gets a pop-up of a filtered list of BLE devices. This list only shows unconfigured scanners that are turned on (see Figure 25).
- **Configuring a profile:** The next step is to select a preconfigured profile. These settings are transmitted to the connected scanner.
- **Waiting for server connection:** Once configured, the scanner will reboot and use the configuration to connect to the network and server.
- **Calibration**: Once connected through Wi-Fi, the user gets instructed to move away from the scanner to calibrate the RSS indicator. After this step the scanner is successfully added and placed on the map.

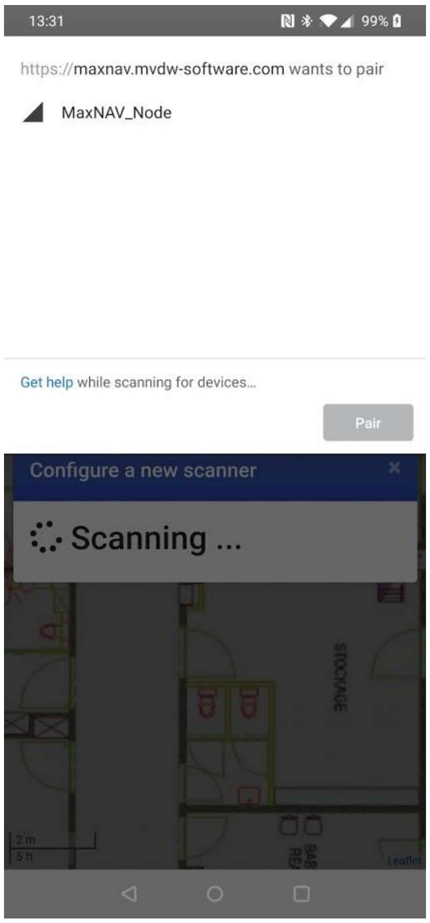

13:31
99%
https://maxnav.mvdw-software.com wants to pair
MaxNAV_Node
Get help while scanning for devices...
Pair
Configure a new scanner
Scanning ...
STOCKAGE
2 m
5 ft
Leaflet

Figure 25 - Bluetooth pairing of a scanner node on the tracking server

By separating the Wi-Fi connectivity settings, the user does not need to remember or fill in the network credentials for each scanner that needs to be added. In case of different Wi-Fi SSID's per floor or area, the user can simply give readable names to the profiles (e.g. FLOOR_1, ZONE_A, …).

9.2.3. Configuration of the Routes

Destinations inside a hospital are connected by hallways. They are configured on the tracking server as a weighted graph. Edges have certain properties, such as their availability or requirements (badges, wheelchair access, …). Vertices or nodes connect multiple edges, allowing the system to determine corners or cross sections. Just like edges, they can have properties that indicate if these nodes indicate stairs, elevators or exits.

Nodes are very important when creating navigation instructions. Therefore, edges can only connect two nodes in a straight line. This implies that nodes will be required in every turn, every cross section or every bend in a hallway. This, in combination with the edge and node properties, provides the tracking server with enough information to provide informational navigation instructions.

Visually, the graph is created by creating and connecting 'route nodes' (nodes shown as the red dots in Figure 26) that have a relative size based on the zoom level of the map. Multiple route nodes are required when creating junctions or intersections.

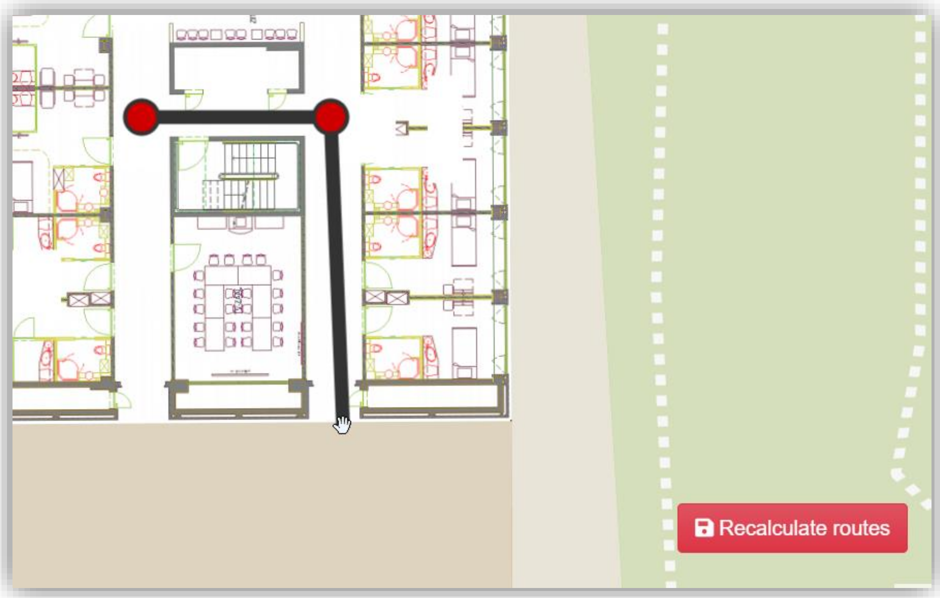


Figure 26 - Route configuration, holding Shift while creating a new edge. Recalculate button is shown

The user can create a route in a visual way. The user starts creating a route by right clicking on the map. A red dot will appear on the map with a 1.5-meter radius. Shift clicking the route node allows the user to drag an edge to another node to connect it. If the user releases the mouse while not hovering another node, a new route node is created at that location. Edges are visualized as dark gray lines with a width of 1-meter (see Figure 26).

After each change to the routes, be it a new route node, a moved route node or a new edge between nodes – the tracking server needs to recalculate these routes. Because this can be a time-consuming task on bigger sites and if performed while end-users are navigating these end-users may get incorrect instructions. A big red button will appear on the bottom right corner to manually commit these changes (see bottom right of Figure 26). A technical description on the recalculation can be found in section 9.4.1.

There are no restrictions on the amount of route nodes or on the proximity of each route node to another. Instead, a visual constraint is enforced by making the edges 1 meter in width and the route nodes 1.5 meters. This acts as an additional visual verification for the user to see if their georeferenced floor plan is correctly scaled.

Both route nodes and edges can have properties. The type of a route node or edge can be specified by means of a dialog window.

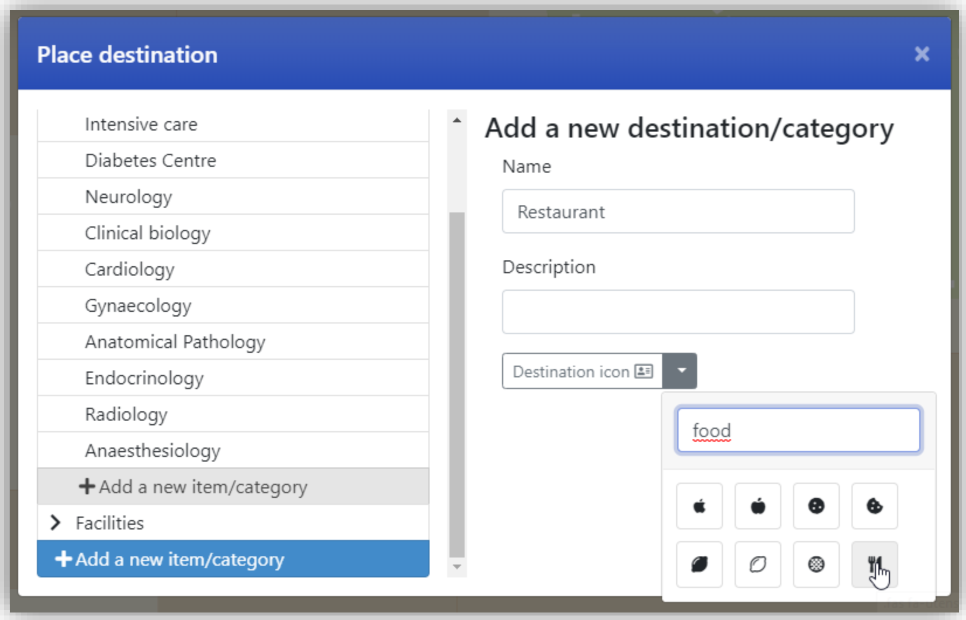
There are three types of route nodes:

- **Normal**: *A normal route node indicating a junction, corner or entrance. These nodes are created to create corners and to connect multiple routes together*
- **Elevator**: *Indicates a route node that brings you to another floor. The navigation uses this to show different instructions.*
- **Stairs:** *Like the elevator type but needs to be explicitly specified if it is a stairs or elevator so the navigation can create accurate instructions ("Go up the stairs to the xxx floor", "Take the elevator to the xxx floor").*

In the case of "Stairs" and "Elevator", the user must also specify the start and end floor because elevators or stairs do not always bring you to all available floors. After selecting the type and inserting the start and end floor, the route node will be connected to the floors above and below.

9.2.4. Configuration of the Destinations

Just as route configuration-user and scanner placement, placing a destination is done by right clicking on a location on the map. For each site, the destinations can have a hierarchical organization of possible destinations. This hierarchy is used by users of the navigation application to quickly find relevant destinations or facilities.


Place destination

Intensive care
Diabetes Centre
Neurology
Clinical biology
Cardiology
Gynaecology
Anatomical Pathology
Endocrinology
Radiology
Anaesthesiology
+ Add a new item/category
> Facilities
+ Add a new item/category

Add a new destination/category

Name: Restaurant

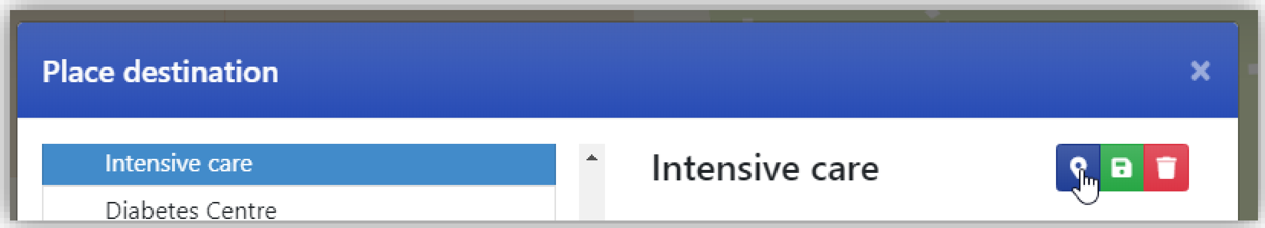
Description:

Destination icon

food

Figure 27 - Creating a new destination under the subcategory "Facilities"

The configurator has three actions that he/she can take when configuring or placing a new destination: Selecting and placing an already added destination (see selected button in Figure 28), creating and managing subcategories and creating new destinations ordered under a specific (sub)category (see Figure 27 showing the creation of the "Restaurant" destination).


Place destination

Intensive care
Diabetes Centre

Intensive care

Figure 28 - Positioning an already created destination

Each destination can have a simple icon that will be shown on the map (see green waypoint in Figure 29). This icon can be unique for each destination with the default icon being the icon of the parent category. For the sake of simplicity of this implementation, the icon that can be chosen must be available in the 'Font Awesome' icon library (Fonticons, Inc., 2019).

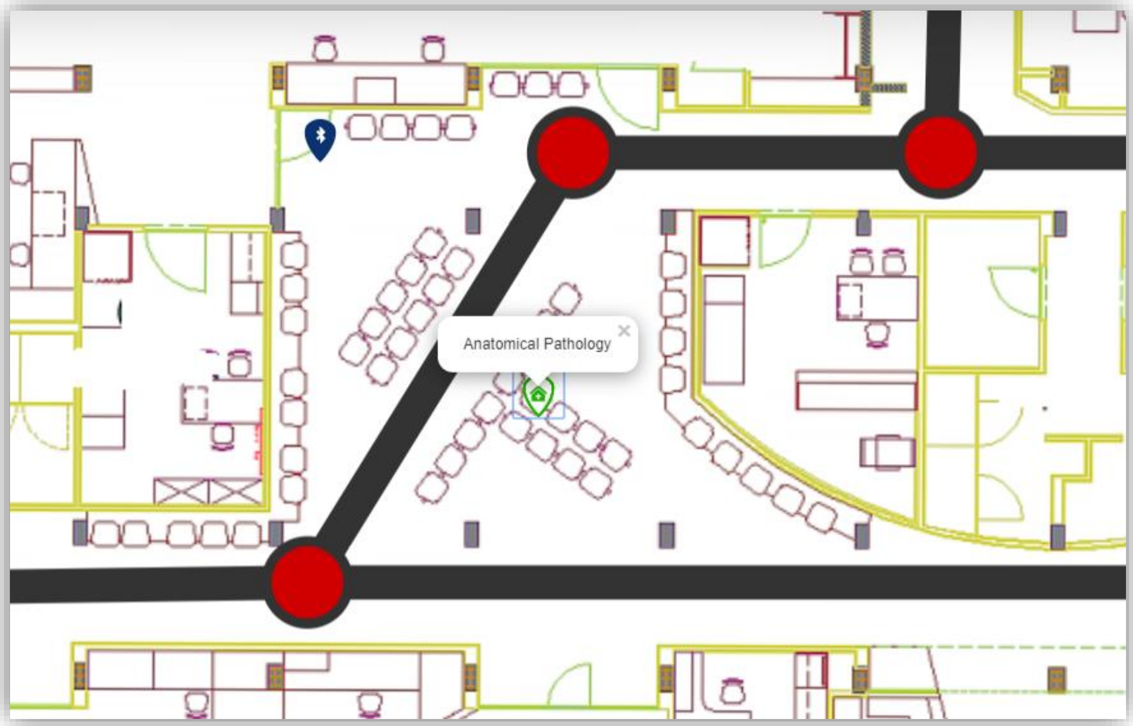


Figure 29 - Fictional example of a destination and scanner shown on the tracking server map

9.3. Calculation the End-User’s Position

From our scanner nodes we receive filtered RSSI readings that were converted to an approximated linear distance (see section 8.3 for the scan result packet). These readings combined with the known longitude and latitude of these nodes can be used to calculate the position on the building floor.

Because of the thick ceilings we can assume that the signal strength to nodes on the same floor are better than floors above or below. This allows us to determine the floor in our positioning by filtering out readings from other floors.

A position of a mobile device consists of the longitude, latitude, accuracy and floor level (elevation). Accuracy is the estimated distance between the actual location and the determined location.

Chapter 5 discussed two different approaches to indoor positioning, calibration based positioning and mathematical positioning. Calibration based positioning can offer an accurate location but requires a lot of work to set up and maintain. Mathematical positioning on the other hand does not require excessive calibration but works best in open spaces.

Because of the extensive amount of work needed to calibrate all scanner nodes in the hospital, the choice was made to try mathematical positioning.

Unlike applications such as Google Maps or other navigation applications, the goal is to allow the user to navigate inside a building. Distances between scanners or even destinations are much smaller than the destinations used in a Car GPS system for example. To keep the positioning mathematical correct, the formula’s used will follow the same principals as long-range GPS positioning. These formula’s take the curvature of the earth into account when calculating distances and angles between locations but have very little effect over the small distances inside a building.

There are four scenarios discussed in sections 9.3.1, 9.3.2, 9.3.3 and 9.3.4 that change the type of positioning depending on the number of scanners that are receiving advertisements from the mobile device (see section 4.1.2 on Bluetooth Advertising).

A scanner is considered in the range of the mobile device when the last advertisement is not older than 5 seconds. This range is required because scanners do not publish their results at a fixed and synchronous rate, meaning some results may be more up to date than other scanners.

9.3.1. Single Point Positioning

When only a single scanner node is in range of the device that is being tracked, the position can only be estimated by the estimated distance to this single scanner. This is called cell identification as explained in section 5.1.

The device's location will be set to the scanner's location and floor, with an accuracy that is the same as the estimated distance between the device and scanner.

9.3.2. Two-Point Positioning

As mentioned at the end of chapter 6, we can determine a location based on two scanner nodes. This is done with the knowledge that in most cases, the scanner nodes are positioned in hallways. The mobile device that needs to be positioned is located on a straight line between these two points.

$X = Unknown\ location$

$A = Scanner\ A$

$B = Scanner\ B$

$|AX| = Distance\ between\ scanner\ A\ and\ X\ in\ meters$

$|BX| = Distance\ between\ scanner\ B\ and\ X\ in\ meters$

Used as an example, Figure 30 shows two scanners that are in a hallway at UZ Brussel. The chosen hallway on the ground floor is for administrative purposes and therefor does not require a dense scanner grid.

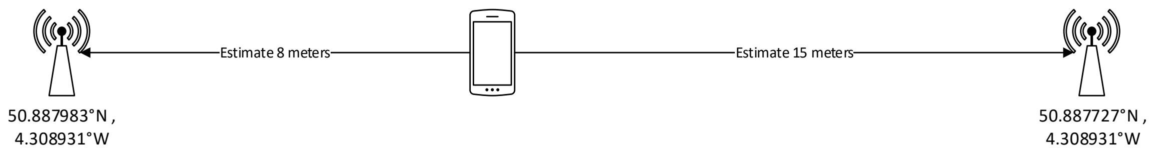


Figure 30 - Two scanners located in a large hallway at UZ Brussel

For demonstration purposes and coincidental orientation of the building, the longitude of the two scanner nodes is the same. However, the two-point positioning works using the bearings of the two points to make sure it works on different orientations as well.

Calculate the bearing and distance between the two points

The two coordinates of the scanners are used to determine a bearing from A to B and B to A. This is the direction in a straight line from one point to another using the *Forward Azimuth formula*.

Over large distances, the bearing of the starting point and destination point will be different. It is not expected that the bearing will have a huge variation in the marginal distances used for indoor positioning, but this small additional step has been made to keep the calculation mathematically correct.

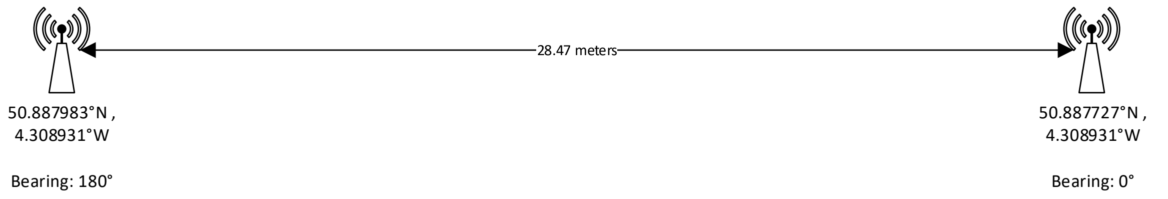


Figure 31 - Two-point calculated distance and bearings

Using the *Haversine formula* we can calculate the distance between the two scanners (Veness, 2011). This is the shortest distance on a sphere, in this case the curvature of the Earth. This distance will be used to estimate the accuracy of the position.

This is not the most accurate way to calculate the distance between two points because the Earth is not an exact sphere. Other algorithms such as *Vincenty's formula* use an accurate ellipsoidal model of the Earth to perform the calculations (Veness, 2011). Additional accuracy is negligible due the already inaccurate RSSI to distance conversion (see section 8.2.4). Even the Haversine formula will only achieve a difference of less than 1 cm over a 40-meter distance by taking the curvature into account. Since we are starting from the geographical coordinates of our scanners, it seems only correct to follow the geographical calculations instead of ignoring these.

Calculate the expected position between each scanner

Once the bearing and distance between the two scanners is known it is possible to calculate two reference points on a straight line between A and B. These reference points use the approximated distance between the scanners.

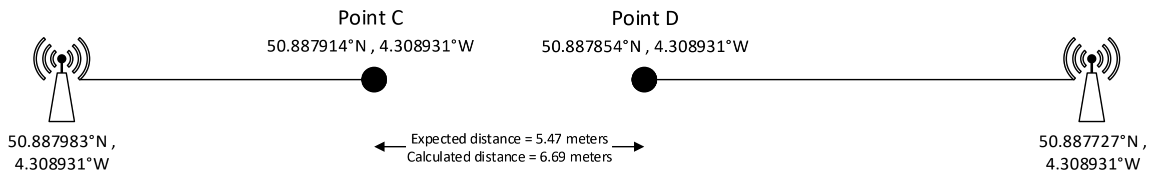


Figure 32 - Two-point calculated estimation points

Shown in Figure 32, the two points do not lay on top of each other. This is due to the inaccuracies in the distance approximation.

$C = Point\ originating\ from\ A\ with\ bearing\ A\ to\ B\ (brng_{AB})\ and\ distance\ |AX|$

$D = Point\ originating\ from\ B\ with\ bearing\ B\ to\ A\ (brng_{BA})\ and\ distance\ |BX|$

Calculate the midpoint of point C and D

We can use the two reference points calculated in the previous step to determine the midpoint.

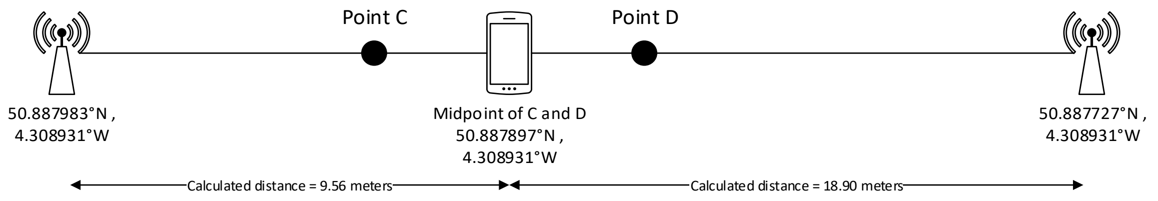


Figure 33 - Midpoint average of two reference points (C, D)

This provides us with the longitude and latitude of our mobile device situated on the direct line between the two scanners.

A more mathematical explanation on the two-point positioning is provided in appendix A.

9.3.3. Trilateration Positioning

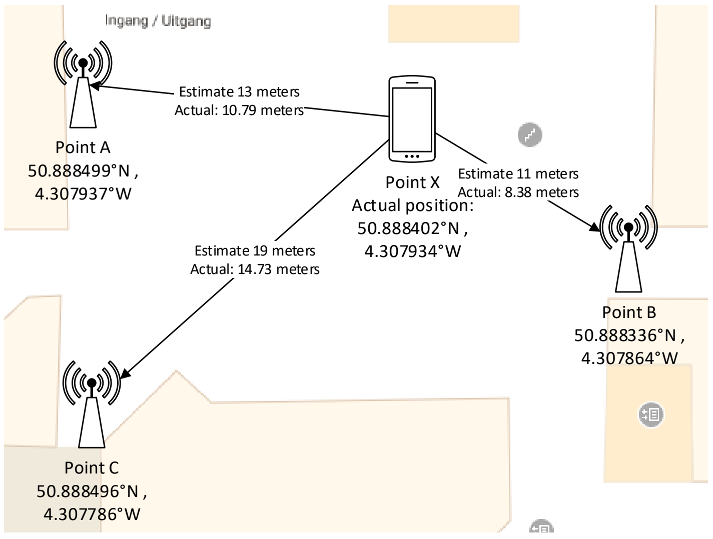


Figure 34 - Trilateration positioning in larger open spaces such as entrance halls (Pediatrics UZ Brussel)

Other than two-point positioning, trilateration does not assume the mobile device moves in a straight line between scanners.

In the example shown in Figure 34 the position X will be estimated by using three rough reference distances.

The example is situated in a large entrance hall at the pediatrics section of UZ Brussel.

Trilateration will mostly be applied/possible in open spaces such as this one where elevators, staircases, appointment registration desks and other possible destinations are located throughout this area.

The algorithm works by calculating the X and Y distance from one of the scanners to the unknown point X. This is done by using the known distances between the scanners and the estimated distances between the unknown point X and each of these scanners.

A more in-depth and mathematical explanation is given in appendix D.

9.3.4. Selective Trilateration

When more than three scanners are in range, the system will use a selective trilateration. A possible solution to three or more-point trilateration would be using nonlinear squares. This is a brute force method that determines the best fit position for all the available points.

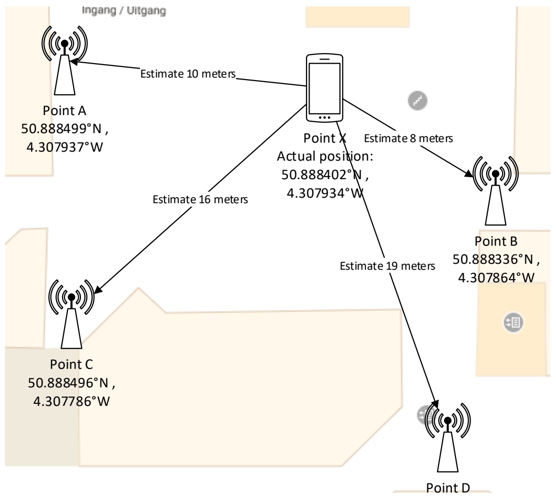


Figure 35 - Selective trilateration with 4 scanners in range

This works very well in open spaces, but indoors it would mean that multiple points also mean that these scanners are behind walls, doors or other obstacles. This would influence the actual location.

Instead, for indoors the choice was made to use selective trilateration. Instead of using every single scanner that is in range, it will order the scanners based on their calculated distance. Assuming the ordered scanners are named $P_1, P_2, P_3, \dots, P_n$, only the first four scanners will be used for positioning.

Two regular trilateration calculations are made with P_1, P_2, P_3 and P_1, P_2, P_4 (see section 9.3.3).

The two resulting points of these trilateration are then used in the same calculation discussed in the *Two-Point Positioning* (section 9.3.2).

This selective trilateration method limits the influence of obstructed scanners to the determined position.

9.3.5. Scanner Position Snapping

Scanner position snapping or priority snapping is an additional method that is tested before any other mathematical positioning method.

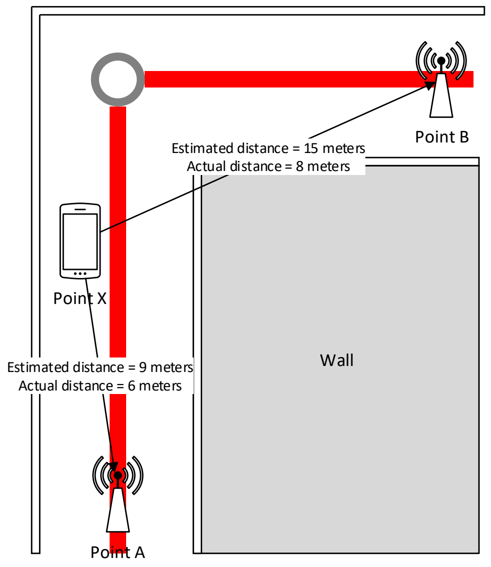
When a mobile device comes close enough to a scanner, the location of that mobile device will be set to the scanner’s position. This is the cell-ID based positioning as mentioned in section 5.1.

Scanners located within a 2-meter radius of a cross sections or corner will have a 3-meter snapping radius. Other scanners have a snapping radius of 1-meter.

9.3.6. Route Snapping

The goal of the tracking server is not to determine the accurate geographical location of the mobile device within the hospital, but to determine a location that is helpful during the navigation.

While the scanners and mathematical positioning offer a rough estimate on the position of the user, our positioning method does not take walls or obstacles into account when determining a position. As can be seen in Figure 36, the phone is nearing a corner with two scanners in range. Two-point positioning determined the location of the mobile device on a straight line between these two points (section 9.3.2). This would give a physically impossible position inside the wall.



Point B
Estimated distance = 15 meters
Actual distance = 8 meters
Point X
Estimated distance = 9 meters
Actual distance = 6 meters
Wall
Point A

Figure 36 - Cornered hallway with two-point positioning (Actual Point X location)

Therefore, route snapping will project the smartphone's calculated location ($Point\ X$) to the closest route location ($Point\ X'$) as shown in Figure 36. This assures that the location of the user is always on a physically possible location.

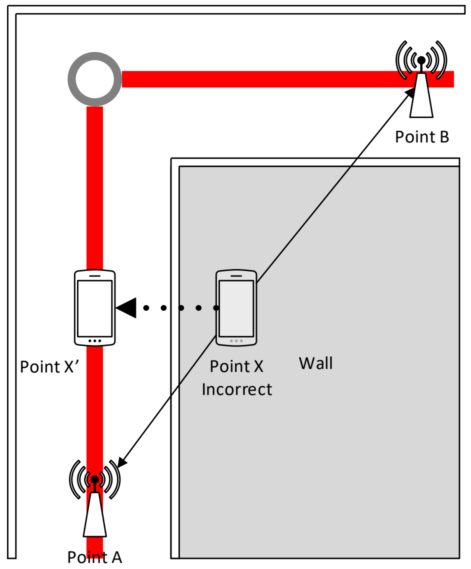


Figure 37 - Cornered hallway with two-point position snapping (Incorrect and projected X location)

The tracking server will do this in two parts:

It will first convert all the edges of the predicted floor as a pair of two vertices to vectors. Lines going through the raw $Point\ X$, orthogonal to these edge vectors are used to find the closest intersections on each of these edges. Lastly the closest intersection is used as the projected $Point\ X'$.

Next, the system creates a set ($\mathcal{V}$) of vertex points with ECR locations. These are all the vertices of the route graph from the predicted floor. Just as with trilateration, the ECR points are treated as vectors in the calculations.

$\mathcal{V} = \{V_1, V_2, \dots, V_n\}$

Each of these points has one or more edges that contribute to the total set of all edges ($\mathcal{E}$). This 'edges' set contains pairs of vertices that are connected. Using this set of edges and the ECR locations of the vertices, the vectors of each of these edges are calculated.

With $V_i \; and \; V_j$ being two points $\in \mathcal{V}$. The vector $\vec{e}_{\{i,j\}}$ is the direction between i and j.

$\vec{e}_{\{i,j\}} = V_j - V_i$

The vector V_i to the incorrect point P_X is $\vec{r} = P_X - V_i$, using this vector the projected orthogonal point P'_X is:

$$P'_X = V_i + \frac{\vec{r} * \vec{e}_{\{i,j\}}}{\vec{r}^2} * \vec{e}_{\{i,j\}}$$

This point (P'_X) is the projection of point P_X on the same direction of the edge i to j. It does not stop at the ends of this line. The calculation of P'_X is changed so the snapped location does not exceed the two outer points of the edge line:

$$P'_X = V_i + \min\left(1, max\left(0, \frac{\vec{r} * \vec{e}_{\{i,j\}}}{\vec{r}^2}\right)\right) * \vec{e}_{\{i,j\}}$$

This is done for all edges, if the closest possible difference between $P_X \; and \; P'_X$ is less than 10 meters away, it will be used as the route snapped location. If it is further away, it is assumed that the user went "off road". In this case, the navigation will guide the user towards the nearest route.

9.4. Route Navigation

After the route snapped location of the device is known, the navigation functionality of the tracking server should guide the user in the right directions.

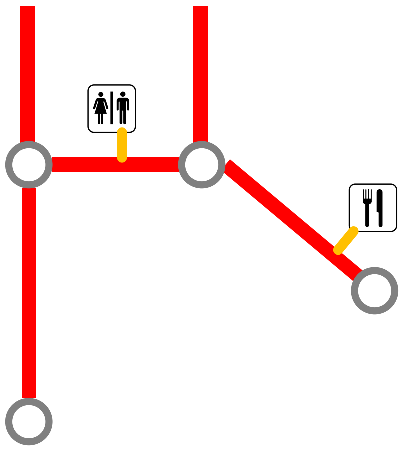


Figure 38 - Internal graph representation connecting routes and destinations

In section 9.2.2, we explained that the routes are configured as a weighted graph, with the weight representing the distance between vertices. This distance is then converted to "traveling time" by dividing it by the average walking speed. By using traveling time as the weight, it is possible to indicate the difference between taking the stairs and waiting for an elevator. It can also be used to indicate busy hallways where walking would be much slower during certain hours.

Note that the destinations do not correspond to individual vertices of the graph, because they can be anywhere alongside or near the edges of the routes. To solve this, an internal graph is created that adds additional vertices for the destinations. This newly created graph ($\mathcal{G}$) with its vertices (V) and edges (E) will be used to navigate the visitor. By adding the destinations as vertices, it provides the tracking server with additional information to instruct the visitor (e.g. Turn left near the vending machines).

In Figure 38 an example is given of how this internal graph is constructed. In red are the route edges, that are all connected with route nodes. The "Toilet" and "Restaurant" destinations are linked to this route graph by creating a new route (orange) towards the closest route. The mathematical construction of this graph is very similar to the route snapping discussed in section 9.3.6. Other than projecting a point on the closest edge, this point is used to draw a new edge between the original point and projected point.

While Figure 38 shows a 2D route of one floor, the internal graph is a combination of all floors, with the edges of the elevators and stairs connecting each floor together.

The next subsection, section 9.4.1, describes how the shortest path for the route is calculated. Section 9.4.2 explains how the instructions for the end-users are generated.

9.4.1. Floyd-Warshall's Algorithm

Destinations and vertices of the route do not move once configured. A path finding algorithm will be used to calculate the shortest paths between all vertices after the configuration is modified.

When the navigation app requests the tracking server to navigate to a destination (v_d), it will first determine the closest vertex from the device location (v_s). This pair ($\{v_s, v_d\}$) is then matched to the precalculated route in the database. If the user does not follow the route, a new query to the tracking server will be made to correct for this change.

For ease of explanation it is assumed that the algorithm needs to find the shortest path of all pairs of vertices. In the implementation, these pairs cannot consist of two route vertices – because the user can only navigate to a destination and not a corner or cross section.

The Floyd-Warshall shortest pathfinding algorithm was used for calculating these paths (Wikipedia, 2019a). It works for both directed and undirected graphs that have a positive or negative weight. It has a worst-case complexity of $\Theta(|V|^3)$ for a complete run that returns the paths and lengths of all pairs in the graph $\mathcal{G}$.

Negative weights will not really be used in this implementation, as the weight depends on the time or distance between two points. Possible uses of these negative weights would be for crowd control, to force (certain) users to take other paths regardless of their traveling time but were not further researched.

Although all hallways and doors in UZ Brussel can be accessed in all directions (except for fire emergency doors, but these are not considered as routes) the possibility of having directed edges was a deciding factor in the choice of this algorithm. Other buildings such as airports or malls have moving walkways or escalators that can only be accessed in a single direction, if the implementation would be unable to take this into account, it would not provide a correct route.

The implementation of the algorithm is divided into four steps. In the first two steps, a ($|V| \times |V|$) distance matrix and vertex matrix are created and initialized. The distance matrix shows the minimum distance between a pair of vertices ($i\ and\ j$), the vertex matrix shows the next vertex step when traveling from one vertex to another ($i\ to\ j\ has\ k\ as\ the\ next\ step$). During the initialization the distance and vertex matrices are filled with all edge pairs. (Wikipedia, 2019a).

The third step compares the weight of going from one vertex to another. If the weight is smaller, this is the new preferred path that will be inserted in both the distance and vertex matrices.

The last step uses the vertex matrix to construct a complete path consisting of all the traversed vertices from each possible vertex that is not a destination to each possible vertex that is a destination.

Step 1) Initialize the matrices

Two $(|V| \times |V|)$ matrices are initialized with V being the set of vertices in the graph $(\mathcal{G})$. D is the distance matrix that will contain the shortest distances from $u \in V$ to $v \in V$, $\forall u, v \in V$. N is the vertex matrix that will contain the shortest next move from u to v. Following these 'next' moves will result in a complete route.

$$D = \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,|V|} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ d_{|V|,1} & d_{|V|,2} & \cdots & d_{|V|,|V|} \end{bmatrix} = \begin{bmatrix} \infty & \infty & \cdots & \infty \\ \infty & \infty & \cdots & \infty \\ \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \cdots & \infty \end{bmatrix}$$

$$N = \begin{bmatrix} n_{1,1} & n_{1,2} & \cdots & n_{1,|V|} \\ n_{2,1} & n_{2,2} & \cdots & n_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots \\ n_{|V|,1} & n_{|V|,2} & \cdots & n_{|V|,|V|} \end{bmatrix} = \begin{bmatrix} NaN & NaN & \cdots & NaN \\ NaN & NaN & \cdots & NaN \\ \vdots & \vdots & \ddots & \vdots \\ NaN & NaN & \cdots & NaN \end{bmatrix}$$

Step 2) Insert the edge weights in the distance matrix

After initialization, the weight of the edges $(\mathcal{E})$ from the graph $(\mathcal{G})$ are inserted into the distance matrix.

$\forall e \in \mathcal{E}$ with e_{start} being the start vertex of the edge, and e_{end} being the end vertex of the edge:

$$d_{e_{start}, e_{end}} = weight(e)$$
$$n_{e_{start}, e_{end}} = e_{end}$$

If the edge is not directed, the reverse is also possible:

$$d_{e_{end}, e_{start}} = weight(e)$$
$$n_{e_{end}, e_{start}} = e_{start}$$

The distance from vertices to themselves will be set to zero:

$$\forall v \in V: d_{v,v} = 0$$

<u>Step 3) Construct the distance and vertex matrices</u>

With $u, v, k \in V$:

If the distance from u to v ($d_{u,v}$) is greater than the distance from u to $k + k$ to v ($d_{u,k} + d_{k,v}$), then it is faster to pass through k when going from u to v.

The distance matrix (D) is changed with $d_{u,v} \leftarrow d_{u,k} + d_{k,v}$
The vertex matrix (N) is changed with $n_{u,v} \leftarrow n_{u,k}$

<u>Step 4) Construct the paths based on the vertex matrix</u>

The vertex matrix (N) contains the next moves that should be made when traveling from one vertex to another. To construct a path, these next moves can be followed until the destination is reached.

Typescript code calculating the shortest paths can be found in the appendices at section F.

9.4.2. Generating Instructions

Navigation instructions are generated on the tracking server. They consist of both visual and textual instructions that update every time a new scan result of the tracked device is received. All instructions are based on the route graph and combined graph of routes and destinations as mentioned in the previous section.

Together, the visual and textual instructions need to guide the end-user throughout the building. Determining the level of detail for the instructions was a very time-consuming task. As mentioned in multiple sections throughout this thesis, the mathematical positioning of scanned smartphone transmission strength does not always allow a high accuracy. Sometimes it is better to leave out details if you are not able to guarantee the correctness of information.

It became clear that arrows would be the best way to guide a user through a building with hallways, as opposed to using a 2D map with a location marker. A small user experiment with a low fidelity prototype was done with simple directional arrows (forward, left, right, backwards). The participants (students) had to go to a specific destination (classroom) using these remotely shown arrows and simple instructions that were vocally provided by me.

This experiment has taught us a few important things that distinguish indoor navigation from outdoor navigation:

- **Distances are too detailed**: Providing the user with the distance to a certain turn or door was too detailed. This type of accuracy cannot be provided by the system and needs a very fast latency for the short distances found in buildings.
- **Speed/Step based actions require too much thinking**: Another idea that was tried out was to tell the user to turn left/right in a certain number of steps or after a certain number of seconds. While this worked for most participants, their focus shifted from trying to find their destination to counting the seconds or steps.
- **Arrows alone are not detailed enough:** At first, simple arrows seemed a perfect solution to guide the user. However, when a user looks around or starts walking in the wrong direction, these arrows become very hard to follow.

- **Reference points work the best:** The best results were obtained when providing instructions that included certain visual landmarks in the hallway, such as a vending machine, a reception (e.g. the Basic Fit reception of Building L at VUB Etterbeek Campus) or even corners.
- **Instructions should be at least one step ahead:** Participants were given instructions based on the virtual position of the scanners. They received instructions such as “Follow the hallway”, “Turn left”, …. However, the participants often wanted to know when or what to do after reaching the end of the hallway or after making a turn.

Based on these observations, the chosen solution was to use reference points and contextual navigation arrows in the instructions. Contextual navigation arrows do not only show the arrow in the direction you need to take but show other possible routes or paths that lay nearby. Car GPS systems already use these arrows when showing directions on a road map. The other routes or paths are the roads, and the road you must follow is indicated in a different (more visible) color. The tracking server will use all information that is configured as reference points. This includes the configured routes and destinations.

Visual Instructions

The graphical arrows showing the direction the user should take are rendered on the mobile device and will be discussed in section 10.2. The tracking server will generate a formatted message that can be used by the mobile app to generate these arrows. This gives the developer of the navigation app the freedom to render this information according to a preferred style, e.g. using the style guidelines of the company where this application was created for. It also reduces the data that is transmitted to the smartphone.

The instructions are based on a mini map consisting of a background (rendered as a light gray route in the provided examples) and a foreground that shows the directional arrow on top of the routes. Note that, as explained, the navigation app can visualize this mini map in a different way.

The scale of the mini map needs to be big enough so it can provide timely information to the user, but small enough that it does not overwhelm the user with a "route plan" that contains unneeded information. Therefore, the route is projected on a 7x7 grid with each pixel in the grid covering $2m^2$. This means that the width and height of the mini map is 14x14 meters with the center containing the junction or corner that the user is nearing and facing [2].

This size and scale were chosen for various reasons: By having the width and height as an uneven number, it was possible to center the "action" in the middle of the mini map. The scanner position snapping discussed in section 9.3.5 has a 2-meter radius, meaning that it would be appropriate to keep the same scale for the mini map. Lastly, the width and height were chosen based on the reaction time of the scanners and users. With the "action" being in the center of the screen, 6-8 meters before the user must perform the action gives him/her a long enough reaction time. Its orientation is towards the center, meaning the route starts at the bottom middle.

[2] The facing direction can be obtained by the compass heading that is transmitted to the tracking server (see section 7.2.2).

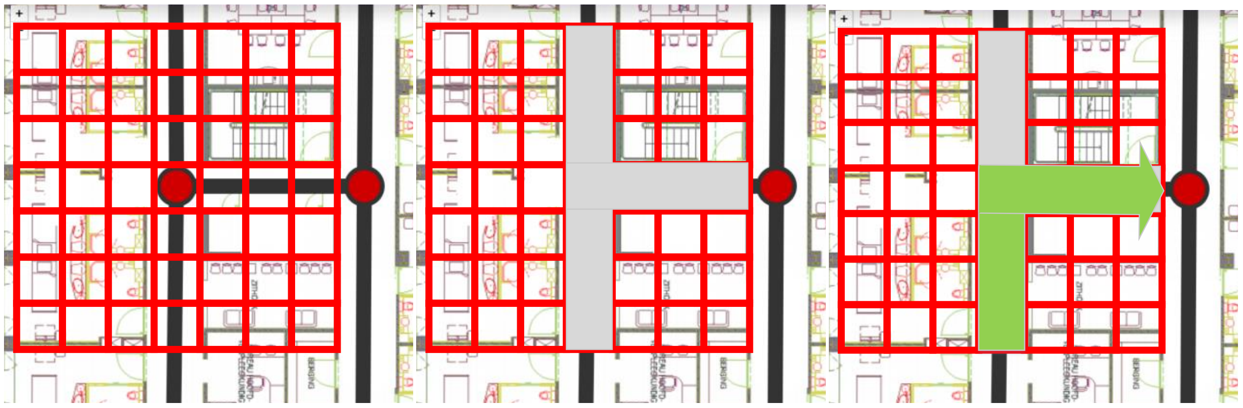


Figure 39 - Mini map creation based on routes. (1) 7x7 pixel grid with a size of 14mx14m, (2) Grid with background context, (3) Grid with directional arrows on top of the background context

When the user nears a critical point (see chapter 6), a 7x7 grid based mini map is constructed with this critical point being in the center (Figure 39, 1). All routes that lay on the grid (and which are relevant to the navigation [3]) are then drawn as the background. The route that the user must follow will be displayed by the navigation application in a different, more noticeable color (see section 10.2).

The rendering resolution of the 7x7 grid is not pixel based. In the example given in Figure 39, the routes lay perfectly on the grid, but the final mini map simply provides instructions to draw a line from $(x_1, y_1)\ to\ (x_2, y_2)$.

[3] It is not relevant for the user to see routes that he/she will never visually pass or see. Only the route branches that lay on the path are rendered in the background

```
{
   "floor": "2",
   "background":[
      {
         "x1":0,
         "y1":-3.5,
         "x2":0,
         "y2":0
      },
       …
      {
         "x1":0,
         "y1":2,
         "x2":0,
         "y2":3.5
      }
   ],
   "foreground":[
      {
         "x1":0,
         "y1":-3.5,
         "x2":0,
         "y2":0
      },
       …
      {
         "x1":0,
         "y1":2,
         "x2":-2,
         "y2":2
      }
   ],
   "destinations":[
      {
         "x":-2,
         "y":2,
         "icon":"fa-bed",
         "name":"Room 1"
      },
      {
         "x":2,
         "y":2,
         "icon":"fa-bed",
         "name":"Room 2"
      }
   ]
}
```

Figure 40 - JSON mini map instruction

The mini map is transmitted to the navigation app as a JSON object. Shown on the left is a mini map that can be used to render the directions to the fictional destination “Room 1”.

It consists of the floor number that will be empty when nearby an elevator or staircase, the background lines that need to be rendered, the foreground lines that display the path that should be taken and lastly a list of destinations that fall within the $14m^2$ grid of the mini map.

Each ‘line’ instruction consists of a start and end point. The foreground lines are ordered based on the path that must be followed. This ordering can be used by the navigation app to determine the location of the arrowhead.

If a line ends at an elevator or stairs, it will contain the data for the destination floor.

The final section of the JSON description contains the destinations. Each of these destinations includes a location, configured icon (Font Awesome icon) and name.

For a visual representation of this packet to the navigation app, see section 10.2 with Figure 46.

Textual Instructions

Textual instructions are based on distances, route nodes and destinations that could be used as reference points. During the georeferencing (section 9.2.1), the user provided the floor plan with a scale that can be used to determine the accurate distance between locations.

Figure 41 shows the 7x7 grid used for the visual mini map. The black center pixel represents the critical point where the user needs to pay attention.

The textual instruction always applies to the next action the user must take. This is the action the user needs to perform at the black center pixel.

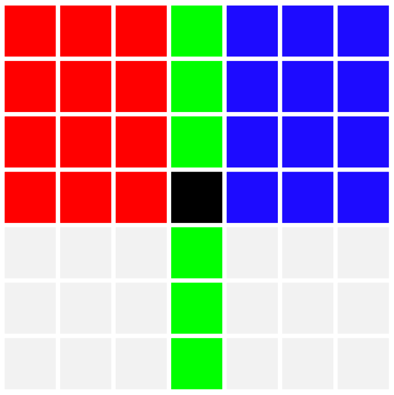


Figure 41 - Textual direction instruction based on the next route

If the next edge the user needs to take lies on the green pixels, the direction is forward. If the next edge lies on the red or blue pixels, the direction is respectively left or right.

Lastly, some hallways may have sharp turns, these are shown as the light gray pixels. It would not always suffice to simply use "left" and "right" in these situations because the user needs to turn around to see the hallway that he/she needs to follow.

Once the direction is known, visual landmarks must be found to provide the user with reference points. For this, the destinations are used in combination with the 7x7 grid. If the direction is either left or right, it checks if there are any destinations near the inner side of the corner (priority 1 shown in Figure 42). If none are available, it will check if there are any destinations above the turn (priority 2 shown in Figure 42). Again, if no destinations are on the complete left side, it will prioritize destinations found in sections 3 and 4 of Figure 42.

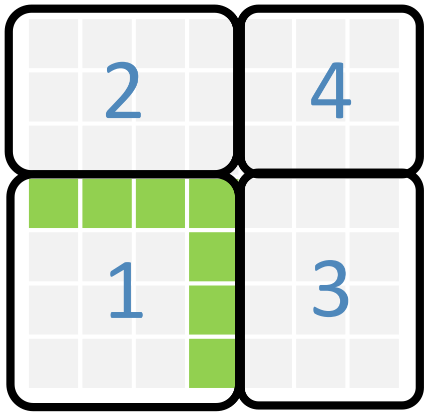


Figure 42 - Textual reference points based on the priority of a left turn

Respectively depending on a destination in 1 to 4, the format of the instruction will be:

Turn left <u>at</u> ***destination***

Turn left <u>before</u> ***destination***

Turn left <u>after passing</u> ***destination***

Turn left <u>in front of</u> ***destination***

The process is similar for a right or sharp turn. In the case of a forward direction or incase configured destinations are nearby; the system will fall back on simpler instructions using the routes as reference points (e.g. *"Continue following the hallway"*).

10. Smartphone Navigation App

The aim for the smartphone navigation app was to have a very short development time. We could realise this because most of the work is done on the tracking server and scanners. Existing indoor navigation systems often focus on the application to do all the work. This is necessary because they use beacons, and the application needs to determine their distance to calculate a location.

The installation of an app is an unfortunate limitation of our current indoor navigation. Modern browsers are slowly starting to support the unofficial Bluetooth Web API that currently allows scanning and connecting to other BLE devices (Beaufort, 2018). Because we want to do the reverse and turn on the BLE advertising – it was not possible to use this API for our case. Therefore, an app was needed.

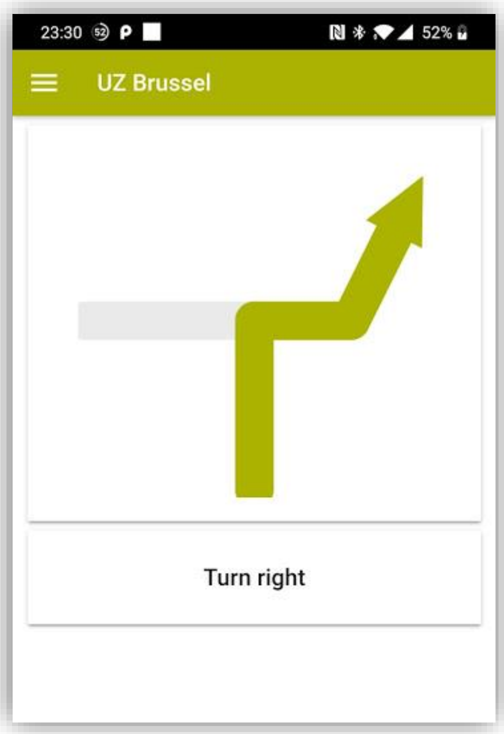


Figure 43 – Smartphone App screenshot during navigation

The navigation performs the following five basic tasks:

1. Load the available destinations from the tracking server
2. Ask the user for the destination (or load it from another system, e.g. the appointment system)
3. Start Bluetooth BLE advertising
4. Connect and identify the device to the tracking server
5. Display the instructions sent by the tracking server until navigation ends

With "Start BLE advertising" being the only native interaction with the smartphone, future implementations that rely on the Bluetooth Web API are able to completely remove the need of the navigation app. Developers could easily port the current tasks of the app to a Web application that performs AJAX (Asynchronous JavaScript and XML) requests to update the instructions.

The next sections provide more details on the performances of the different tasks.

10.1. Device Identification

Devices need to identify themselves with the tracking server in order to receive navigation instructions. This identification process with the communication between the application and tracking server is visualized in Figure 44.

When the application first starts, it will generate a random service UUID with the first part being pre-defined in the code of the application. This 'first part' allows the scanners to identify Bluetooth devices that use the navigation application (section 8.2.3).

A connection will be established with the tracking server to identify the device with this generated service UUID. This allows the server to link scan results from the scanner nodes to the connected device.

The application will then ask the tracking server for all available destinations. These will be displayed to the end-user so he/she can initiate a navigation towards one of these destinations.

After the user selects the destination to navigate to, the application will start the Bluetooth advertising with this service UUID and transmit this destination to the tracking server, so navigation instructions can be generated.

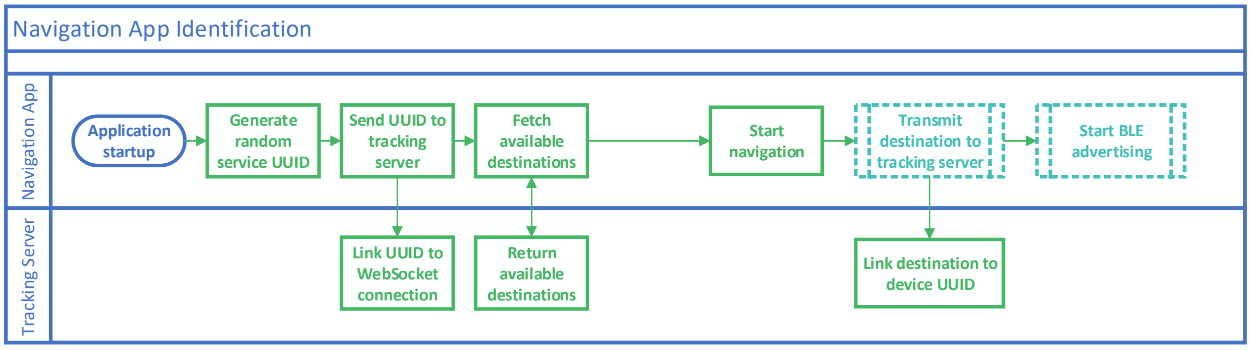
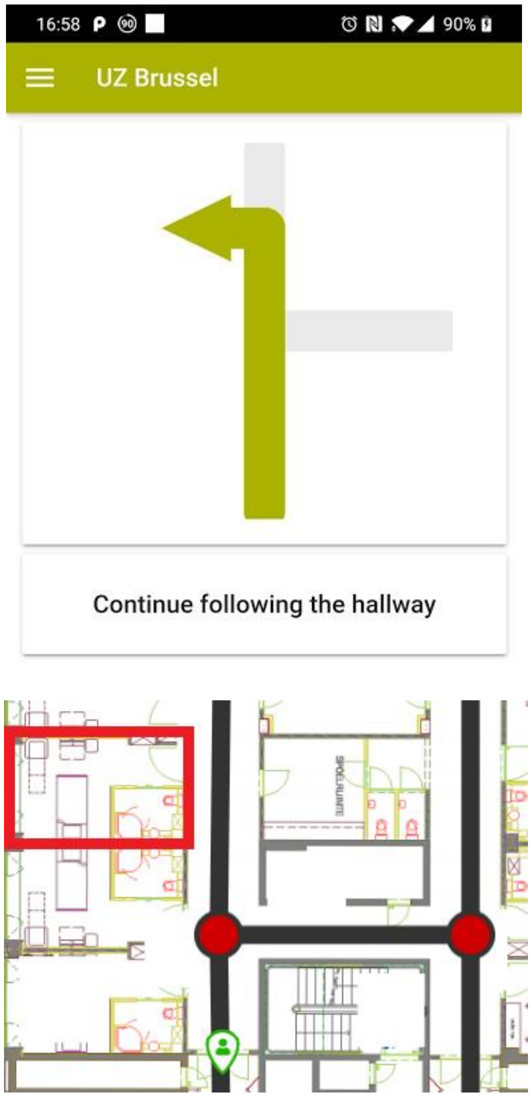

Navigation App Identification
Navigation App
Tracking Server
Application startup
Generate random service UUID
Send UUID to tracking server
Fetch available destinations
Start navigation
Transmit destination to tracking server
Start BLE advertising
Link UUID to WebSocket connection
Return available destinations
Link destination to device UUID

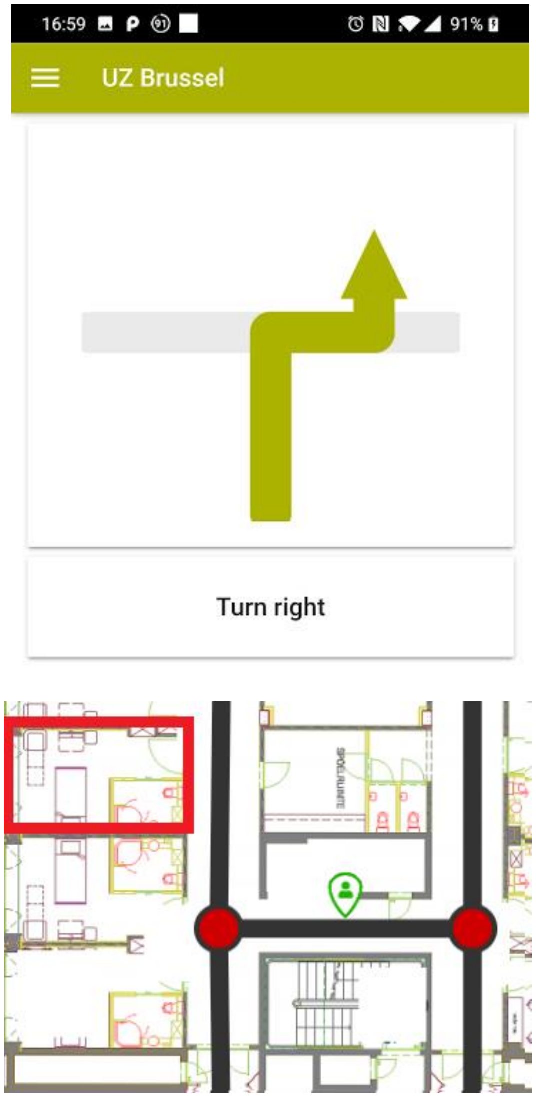
Figure 44 - Navigation app identification process model and communication

10.2. Displaying Navigation Instructions

As explained in section 9.4.2, navigation instructions are generated on the server, but the task of the navigation app is to display these instructions to the user in a useful way. The tracking server created instructions on how to draw a mini map. In our app, we have used the style guide for UZ Brussel for drawing this mini map.



Continue following the hallway



Turn right

Figure 46 - Generated mini map - scenario 1

Figure 45 - Generated mini map - scenario 2

These instructions provided by the server are very low level, meaning that the application developer does not need to spend a lot of time to figure out how to represent them, while still leaving him with the choice of how to visualize them.

Examples are given in Figure 46 and Figure 45. The information configured on the tracking server is shown at the bottom with the position of the user marked as the green waypoint marker (with the user icon). The top of the figures shows the mini map rendered on the navigation app.

A basic implementation of this mini map drawing (implemented in TypeScript) can be found in appendix G.

11. Conclusions, Limitations and Future Work

The main goal of this thesis was to create a system that helps patients and visitors to navigate inside a hospital using the patient's location with the requirements that the acquisition and maintenance cost should not be higher than for existing solutions. Furthermore, patients or visitors should not be forced to use any other devices or hardware other than their own smartphone.

The solution to the problem can be divided into two major issues: determining the user's position and generating helpful navigation instructions based on the user's position.

Different positioning methods exist with their own advantages and disadvantages. Consequently, the choice of a suitable position method is a very challenging problem. Some positioning methods are more accurate than others but require a more difficult or time-consuming installation and development.

Goal of this thesis was to make the indoor positioning problem as scalable as possible. We achieved this by moving the positioning tasks from the smartphone application to a server application, i.e. the tracking server, which has the advantage that such a system can be used in any type of building and for any type of navigation or positioning purpose.

While this goal required various compromises in tracking accuracy, we could compensate them by some of the system requirements. These requirements stated that the goal was to navigate an end-user through hallways inside a hospital. The architectural layout of a hospital (that consists of various routes that connect destinations) allowed for various modifications to the mathematical positioning (such as shown in sections 9.3.4, 9.3.5, 9.3.6) to increase accuracy.

Furthermore, we developed dedicated scanner nodes to track a user's (smartphone) position. These scanner nodes, discussed in chapter 8, are one of the main components of the system. The thesis shows how these inexpensive devices can discover smartphone devices and rapidly transmit these results to the centralized server.

The centralized tracking server (placed on-site in the hospital) almost completely replaced the tasks that other positioning systems perform inside the smartphone application. It uses the information from the scanner nodes to mathematically determine a location and uses this location to generate navigation instructions.

To have the smartphone application widely available, i.e. on different types of devices and operating systems, a hybrid implementation was used. Chapter 10 showed how this application became merely a (Web-based) user interface that visually displays the instructions provided by the server. This gives stakeholders such as UZ Brussel the possibility to implement the navigation into an existing or simple application without much effort.

Even though the system is usable for other buildings, it is still aimed towards a hospital environment. In its current state, it will be very difficult to use the system for buildings with very open and large hallways, such as malls or airports, because it does not show a 'map' of the floor you are on. In that case, it would be useful for the user that he/she could follow a dot on an indoor map with high accuracy and live updates. However, when navigating from one point to another through a series of hallways as required in a hospital, this is not necessary.

Every system involves multiple actors. This system creates a user-friendly experience for all actors, including the people who have to install and configure the system, rather than just focusing on the end user.

Although the different components of the system were tested extensively and the early prototypes of the navigation app were evaluated with end-users, further work should be a final end-user evaluation. In this evaluation, the system would be implemented in a large building with multiple floors. The end-users should be guided by only using the navigation app and without knowing the route or destination. Due to time constraints, this final evaluation could not be performed.

Other technical improvements include the use of machine learning for the positioning. Continuing on the work of Ugave (2004) mentioned in section 5.4.2, where calibration is done by capturing routes that are then used to train a neural network. This type of 'route' calibration can help speed up the fingerprinting. This RSS fingerprinting would provide a better indoor accuracy but was not used in this thesis due to the increased installation time and cost.

12. References

Aalto, L., Göthlin, N., Korhonen, J., & Ojala, T. (2004). Bluetooth and WAP push based location-aware mobile advertising system. *The 2nd international conference on Mobile systems, applications, and services* (pp. 49-58). Boston: ACM. doi:10.1145/990064.990073

Accuware Inc. (2018). *Bluetooth Beacon Tracker*. Retrieved October 1, 2018, from Accuware: https://www.accuware.com/support/bluetooth-beacon-tracker-quick-start-guide/

Apache Software Foundation. (2015). *Architectural overview of the Cordova platform - Apache Cordova*. Retrieved January 5, 2019, from Apache Cordova: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

Apple Inc. (2017, January 20). *About the security content of iOS 8*. Retrieved November 1, 2018, from Official Apple Support: https://support.apple.com/en-us/HT201395

Apple Inc. (2018, 10). *iBeacon - Apple Developer*. Retrieved from https://developer.apple.com/ibeacon/

Banggood.com. (2018). *Wemos® TTGO MINI 32 V2.0 ESP32 WiFi Bluetooth Module Development Board.* Retrieved September 20, 2018, from Banggood.com: https://www.banggood.com/en/Wemos-TTGO-MINI-32-V2_0-ESP32-WiFi-Bluetooth-Module-Development-Board-p-1286046.html

Beaufort, F. (2018, October 10). *Interacting with Bluetooth devices on the Web*. Retrieved January 22, 2019, from Google Developer: https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web

Bigelow, S. J., Carr, J. J., & Winder, S. (2001). *Understanding telephone electronics (fourth edition).* Boston: Newnes.

Bluetooth Special Interest Group Inc. (2018b, September 29). *Qualification Listing Fees*. Retrieved from Bluetooth Technology Website: https://www.bluetooth.com/develop-with-bluetooth/qualification-listing/qualification-listing-fees

Bluetooth Special Interest Group Inc. (2018c). *Bluetooth Market Update 2018.* Retrieved November 10, 2018, from Bluetooth Technology Website: https://www.bluetooth.com/markets/

Bluetooth Special Interest Group inc. (2016, December 6). *Specification of the Bluetooth System, v5.0.* Retrieved October 2018, from Bluetooth Technology Website: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043

Bose, A., & Foh, C. H. (2007). A practical path loss model for indoor WiFi positioning enhancement. *Information, Communications & Signal Processing, 2007 6th International Conference* (pp. 1-5). Singapore: IEEE. doi:10.1109/ICICS.2007.4449717

Bulten, W., Rossum, A., & Haselager, W. (2016). Human SLAM, Indoor Localisation of Devices and Users. *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI),* (pp. 211-222). doi:10.1109/IoTDI.2015.19

Bureau International des Poids et Mesures. (1983). *17e Conférence Générale.* Sèvres: BIPM.

Cisco Systems Inc. (2007, February 27). *Omni Antenna vs. Directional Antenna*. Retrieved February 6, 2019, from Cisco.com Worldwide: https://www.cisco.com/c/en/us/support/docs/wireless-mobility/wireless-lan-wlan/82068-omni-vs-direct.html

Dissanayake, M. G.-W. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 229-241.

Dong, Q., & Dargie, W. (2012). Evaluation of the Reliability of RSSI for Indoor Localization. In None (Ed.), *International Conference on Wireless Communications in Underground and Confined Areas.* Clermont Ferrand: IEEE. doi:10.1109/ICWCUCA.2012.6402492

Einstein, A. (1905). Zur elektrodynamik bewegter körper. *Annalen der physik.* doi:10.1002/andp.19053221004

Espressif Systems Ltd. (2018a). *ESP32 Series Datasheet.* Retrieved November 1, 2018, from Espressif Systems - Wi-Fi and Bluetooth chipsets and solutions: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Espressif Systems Ltd. (2018b). *Get Started &mdash; ESP-IDF Programming Guide documentation*, v3.2-dev-1418-g76fcee5. Retrieved October 26, 2018, from https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html

Espressif Systems Ltd. (2019a). *ESP-IDF Programming Guide*, ad3b820e. Retrieved September 25, 2018, from Documents | Espressif Systems: https://docs.espressif.com/projects/esp-idf/en/latest/

Espressif Systems Ltd. (2019b). *Arduino core for ESP32 WiFi chip*. Retrieved March 10, 2019, from GitHub: https://github.com/espressif/arduino-esp32

European Union Agency for Network and Information Security. (2009, July 22). *Health Insurance Portability and Accountability Act - ENISA*. Retrieved from ENISA: https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/laws-regulation/data-protection-privacy/health-insurance-portability-and-accountability-act

European Union Agency for Network and Information Security. (2018, May 28). *Data Protection - ENISA*. Retrieved from ENISA: https://www.enisa.europa.eu/topics/data-protection

Faragher, R., & Harle, R. (2015, November). Location Fingerprinting With Bluetooth Low Energy Beacons. *IEEE Journal on Selected Areas in Communications, 33*(11), 2418-2428. doi:10.1109/JSAC.2015.2430281

Fette, I., & Melnikov, A. (2011). *The WebSocket Protocol.* (No. RFC 6455). doi:10.17487/RFC6455

Fonticons, Inc. (2019). *Font Awesome*. Retrieved April 9, 2019, from https://fontawesome.com/

Gao, V. (2015, September 21). *Proximity And RSSI*. Retrieved October 22, 2018, from Bluetooth Technology Website: https://blog.bluetooth.com/proximity-and-rssi

Gatwick Airport Press Office. (2017, May 25). *Gatwick Media Centre – Gatwick installs 2000 indoor navigation beacons enabling augmented reality wayfinding*. Retrieved from Gatwick Media Centre: http://www.mediacentre.gatwickairport.com/press-releases/2017/17_05_25_beacons.aspx

Gauer, C. (2015). Delivering Location Based Services with Cisco Enterprise Mobility Services Platform. *Cisco Connect.* Dubai. Retrieved October 21, 2018, from https://www.cisco.com/web/offer/emear/38586/images/Presentations/P7.pdf

Google LLC. (2018a, 10). *Beacons | Google Developer*. Retrieved from https://developers.google.com/beacons/

Google LLC. (2018b, August 23). *Web Bluetooth API - Chrome Platform Status*. Retrieved January 16, 2019, from Chrome Platform Status: https://www.chromestatus.com/feature/5264933985976320

Hansen, R., Wind, R., Jensen, C., & Thomsen, B. (2003, May 18-20). Seamless Indoor/Outdoor Positioning Handover for. *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, 267-272. doi:10.1109/MDM.2009.39

HiveMQ. (2015, February 16). *MQTT Essentials Part 6: Quality of Service 0, 1 & 2*. Retrieved November 15, 2018, from HiveMQ - Enterprise ready MQTT to move your IoT data: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

infsoft GmbH. (2018). *Indoor Navigation using Beacons*. Retrieved May 28, 2019, from Indoor Navigation, Indoor Positioning and Indoor Tracking by infsoft: https://www.infsoft.com/technology/sensors/bluetooth-low-energy-beacons

Jami, I., Ali, M., & Ormondroyd, R. M. (1999). Comparison of Methods of Locating and Tracking Cellular Mobiles. *IEE Colloquium Novel Methods of Location and Tracking of Cellular Mobiles and their System Applications.* IET. doi:10.1049/ic:19990238

Kha Hoang, M., Schmitz, S., Drueke, C., Hai Tran Vu, D., Schmalenstroeer, J., & Haeb-Umbach, R. (2013, March 20-21). Server based Indoor Navigation Using RSSI and Inertial Sensor Information. *2013 10th Workshop on Positioning, Navigation and Communication (WPNC)*, 1-6. doi:10.1109/WPNC.2013.6533263

Lee, S.-H., Lim, I.-K., & Lee, J. (2016, March). Method for Improving Indoor Positioning Accuracy Using Extended Kalman Filter. *Mobile Information Systems, 2016*, 15. doi:10.1155/2016/2369103}

Lehtimaki, S. (2018). Understanding Advanced Bluetooth Angle Estimation Techniques for Real-Time Locationing. *Embedded World.* Nürnberg.

Levy, S. D. (2016). *The Extended Kalman Filter: An Interactive Tutorial for Non-Experts.* Retrieved October 10, 2018, from Simon's home page: https://home.wlu.edu/~levys/kalman_tutorial/

Lindemann, A., Schnor, B., Sohre, J., & Vogel, P. (2016). Indoor Positioning: A Comparison of WiFi and Bluetooth Low Energy for Region Monitoring. *HEALTHINF*, 314-324.

Monaca & Onsen UI Team. (2019). *Angular 2+ - Onsen UI*. Retrieved 2018, from Onsen UI 2: Beautiful HTML5 Hybrid Mobile App Framework and Tools - Onsen UI: https://onsen.io/v2/guide/angular2/

Node.js Foundation. (2019). *Node.js*. Retrieved October 20, 2018, from https://nodejs.org/en/

Norman, G. V. (1999). Design guidelines for landmarks to support navigation in virtual environments. *SIGCHI conference on Human Factors in Computing Systems* (pp. 278-285). New York: ACM. doi:10.1145/302979.303062

OASIS. (2014, October 29). *MQTT Specification.* Retrieved October 30, 2018, from MQTT: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

Oosterlinck, D., Benoit, D. F., Baecke, P., & Van de Weghe, N. (2017). Bluetooth Tracking of Humans in an Indoor Environment: An Application to Shopping Mall Visits. *Applied geography, 78,* 55-65. doi:10.1016/j.apgeog.2016.11.005

Oxford University Press. (2019). *Navigation*. Retrieved from Oxford Online Dictionary: https://en.oxforddictionaries.com/definition/navigation

Partezis. (2010). Het Klinisch WerkStation (KWS): de GPS van de bedrijfsmatige geneeskunde in actie. *Healthcare Executive*(53).

Pathak, O., Palaskar, P., Palkar, R., & Tawari, M. (2014, April). Wi-Fi Indoor Positioning System Based on RSSI Measurements from Wi-Fi Access Points–A Tri-lateration Approach. *International Journal of Scientific & Engineering Research, 5*(4).

Pur3 Ltd. (2017). *Frequently Asked Questions - Espruino*. Retrieved October 26, 2018, from Espruino: https://www.espruino.com/FAQ#what-is-espruino-

Roberts, J. (2017). *Privacy and robot vacuum cleaner.* Retrieved November 10, 2018, from Vacuums Guide: https://www.vacuumsguide.com/privacy-robot-vacuum-cleaners/

Ruhr-Universitaet-Bochum. (2015). Personal navigation: How we know where we are. *Science Daily*.

Schaefer, J. (2018, November 28). *Trilateration Versus Triangulation for Indoor Positioning*. Retrieved November 29, 2018, from Leverege: https://www.leverege.com/blogpost/trilateration-vs-triangulation

Sung, Y. (2016, November). RSSI-Based Distance Estimation Framework Using a Kalman Filter for Sustainable Indoor Computing Environments. *Sustainability, 8*(11), 1136. doi:10.3390/su8111136

Taylor, E. G. (1971). *The Story of Navigation - The Haven-Finding Art, A History of Navigation from Odysseus to Captain Cook.* London: Hollis and Carter Ltd. doi:10.1017/S0373463300040364

Test-Aankoop. (2018, April 5). *Kies het beste kanaal voor je wifisignaal*. Retrieved May 24, 2019, from Test Aankoop, een halve eeuw consumentenbescherming: https://www.test-aankoop.be/hightech/telecom/dossier/kies-het-beste-kanaal-voor-uw-wifi-signaal

The Institute of Electrical and Electronic Engineers. (2010). *Information technology Real-time locating systems (RTLS) - Part 5: Chirp spread spectrum (CSS) at 2,4 GHz air interface.*

The Office of the Secretary of Defense. (2008). Section A.3 Coverage. In *GPS Standard Positioning Service (SPS) Performance Standard (4th edition)* (pp. A4-A6). Washington.

Torres-Sospedra, J., Avariento, J., Rambla, D., Montoliu, R., Casteleyn, S., Benedito-Bordonau, M., . . . Huerta, J. (2015). Enhancing integrated indoor/outdoor mobility in a smart campus. *International Journal of Geographical Information Science, 29*(11), 1955-1968. doi:10.1080/13658816.2015.1049541

Trevisani, E., & Vitaletti, A. (2004). Cell-ID location technique, limits and benefits: an experimental study. *Sixth IEEE Workshop on Mobile Computing Systems and Applications* (pp. 51-60). Cumbria: IEEE. doi:10.1109/MCSA.2004.9

Ugave, V. A. (2014). *Smart indoor localization using machine learning techniques.* Colorado: Doctoral dissertation, Colorado State University. Retrieved from http://hdl.handle.net/10217/84567

UZ Brussel. (2018). *Universitair Ziekenhuis Brussel*. Retrieved November 11, 2018, from http://www.uzbrussel.be/u/view/en/17362-Home.html

Van de Wynckel, M. (2016, October 26). *How easy is it to hack my WiFi light bulb?* Retrieved October 26, 2016, from Medium – a place to read and write big ideas and important stories: https://medium.com/@Maximvdw/how-easy-is-it-to-hack-my-wifi-light-bulb-b6ba9192176d

Veness, C. (2011). Calculate distance and bearing between two Latitude/Longitude points using Haversine formula in JavaScript. *Movable Type Scripts*. Retrieved November 26, 2018, from https://www.movable-type.co.uk/scripts/latlong.html

Wang, Y., Yang, X., Zhao, Y., Liu, Y., & Cuthbert, L. (2013). Bluetooth positioning using RSSI and triangulation methods. *Consumer Communications and Networking Conference (CCNC)* (pp. 837-842). IEEE. doi:10.1109/CCNC.2013.6488558

Wetzker, U., Splitt, I., Zimmerling, M., & Römer, K. (2016). Troubleshooting Wireless Coexistence Problems in the Industrial Internet of Things. *14th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC), At Paris, France.* doi:10.1109/CSE-EUC-DCABES.2016.167

Wikipedia. (2019a, February 7). *Floyd–Warshall algorithm - Wikipedia*. Retrieved February 1, 2019, from https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

Wikipedia. (2019b, April 2). *Electromagnetic hypersensitivity - Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/Electromagnetic_hypersensitivity

Wikipedia. (2019c, May 20). *Representational state transfer - Wikipedia*. Retrieved May 26, 2019, from https://en.wikipedia.org/wiki/Representational_state_transfer

Zandbergen, P. A., & Barbeau, S. J. (2011). Positional accuracy of assisted GPS data from high-sensitivity GPS-enabled mobile phones. *The Journal of Navigation, 64*(3), 381-399. doi:10.1017/S0373463311000051

Appendices

The appendices mainly include source code and irrelevant mathematical calculations that can provide more in-depth information on the topics discussed in this thesis.

A. Scanner Node 3D Design

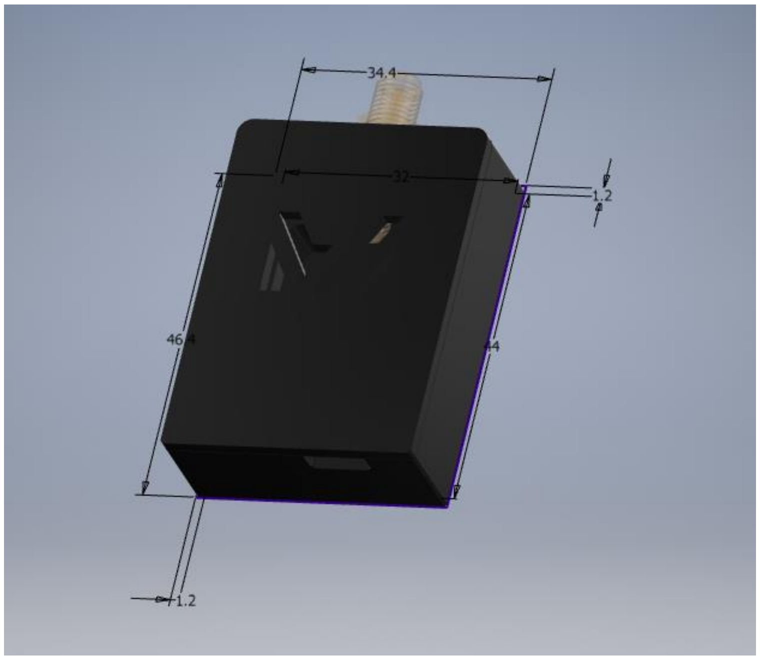


Figure 47 - Scanner node enclosure assembly

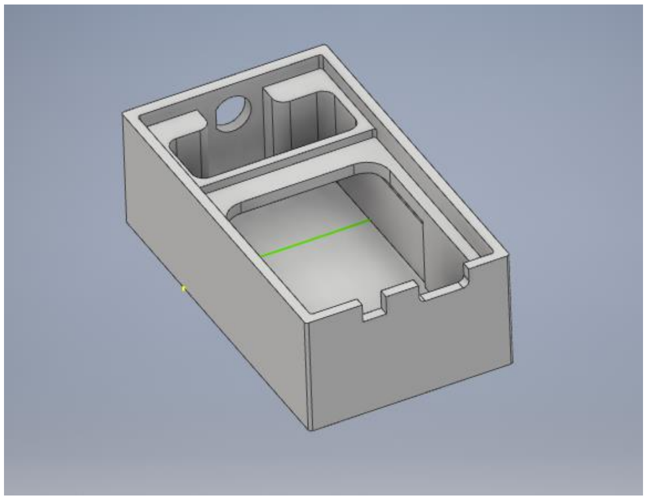


Figure 48 - Scanner node enclosure with battery (Used for experimentation)

B. Tracking Server Configuration Screenshots

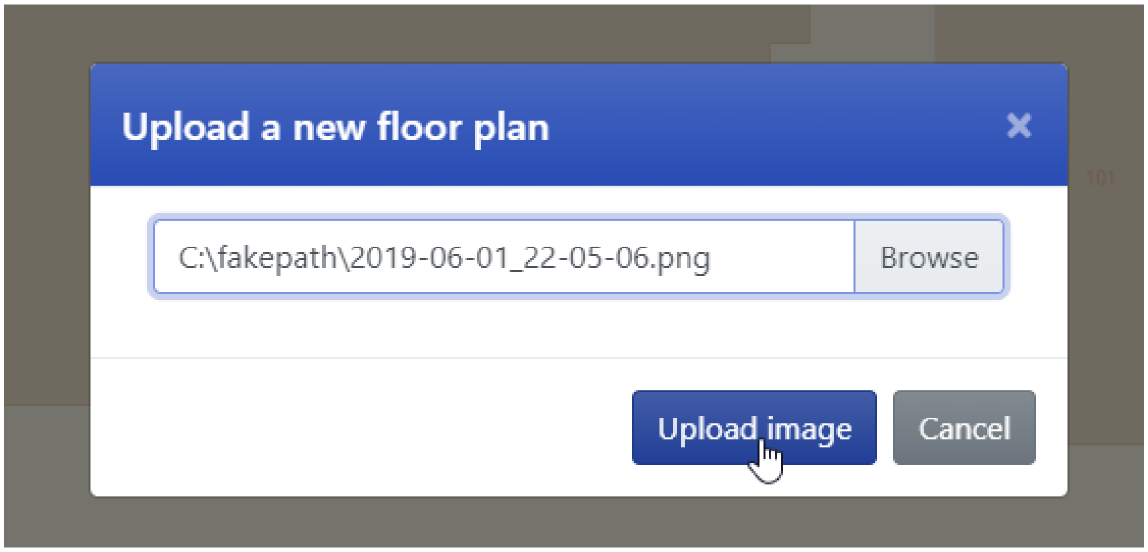


Figure 49 - Uploading a floor plan to the tracking server

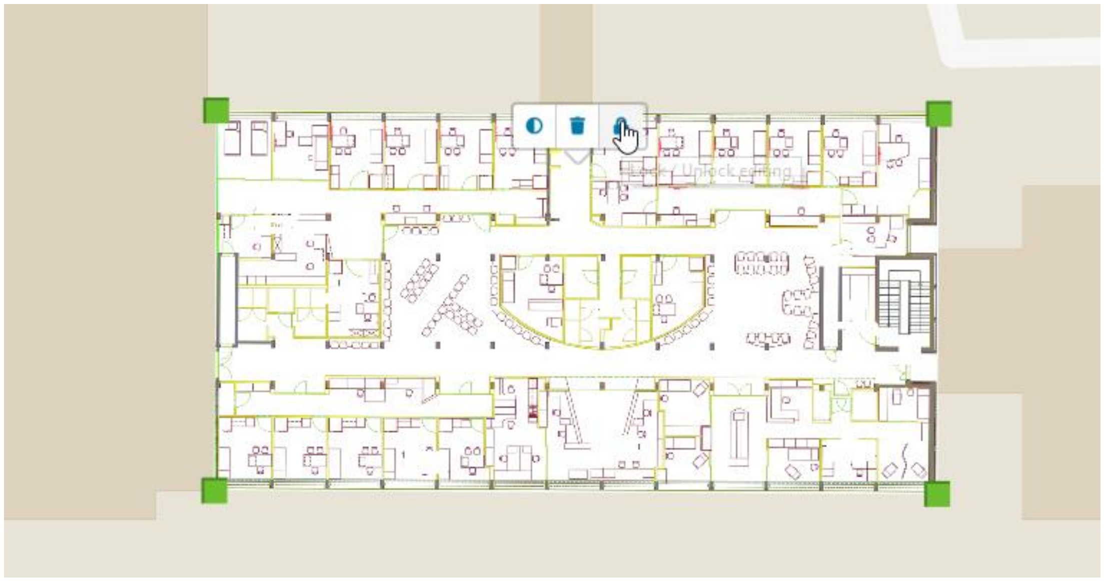


Figure 50 - Aligning a floor plan to the contours on OpenStreetMaps

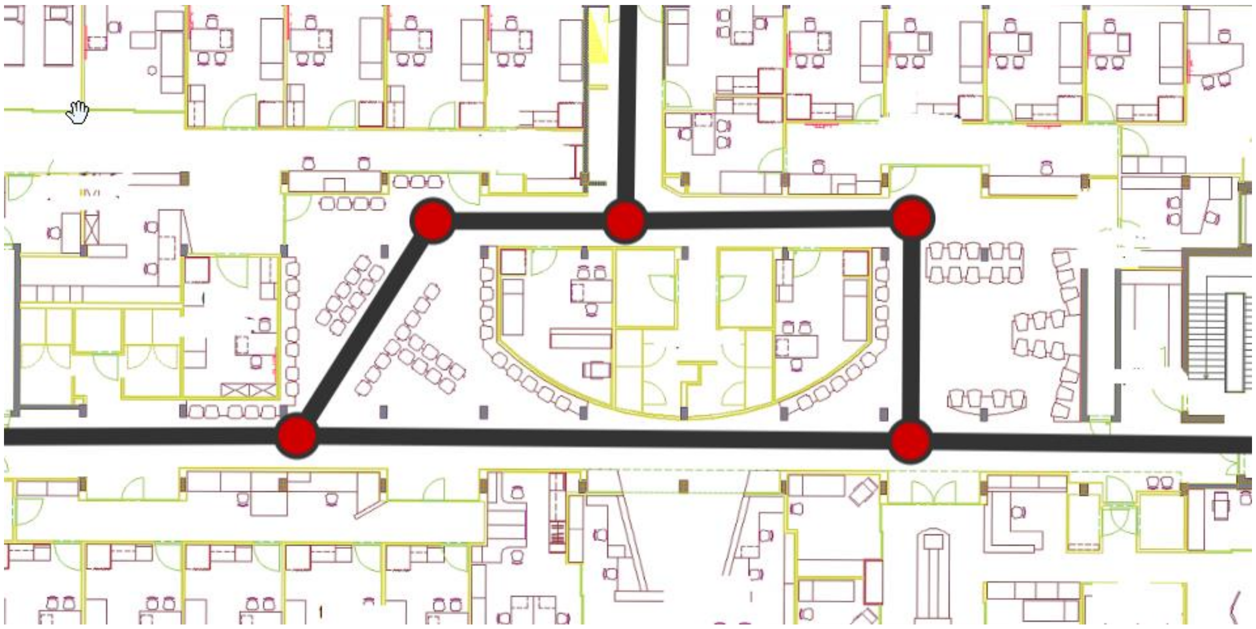


Figure 51 - Route graph in UZ Brussel (Zone C building section 11, floor -1)

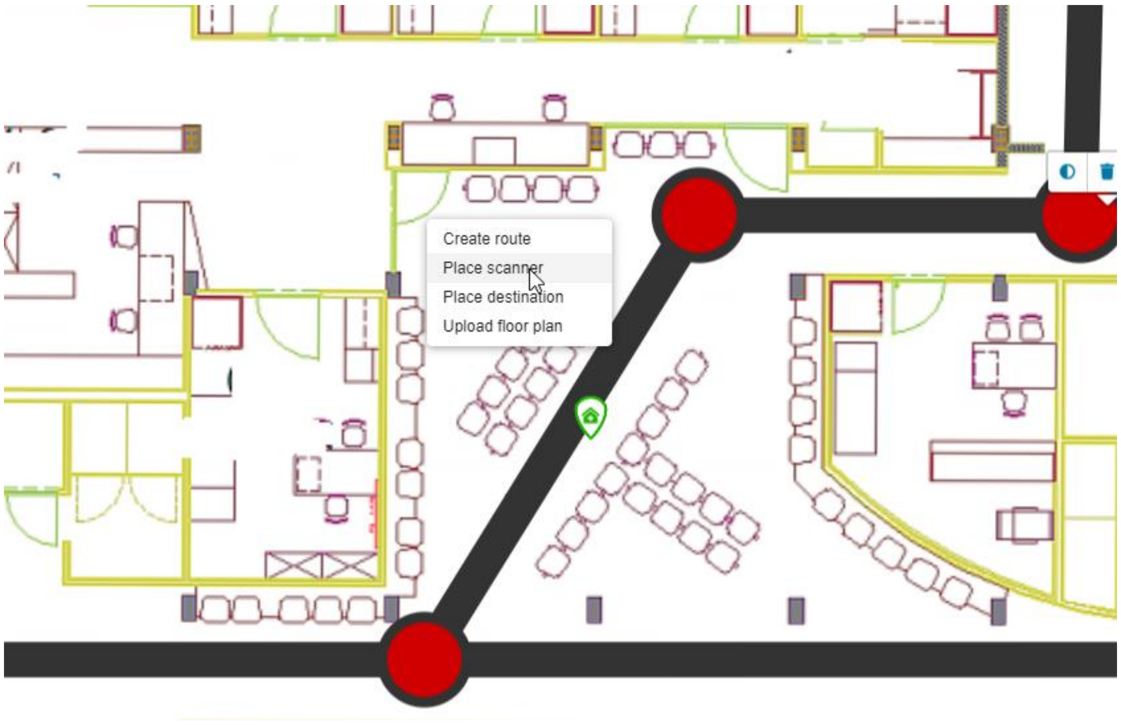
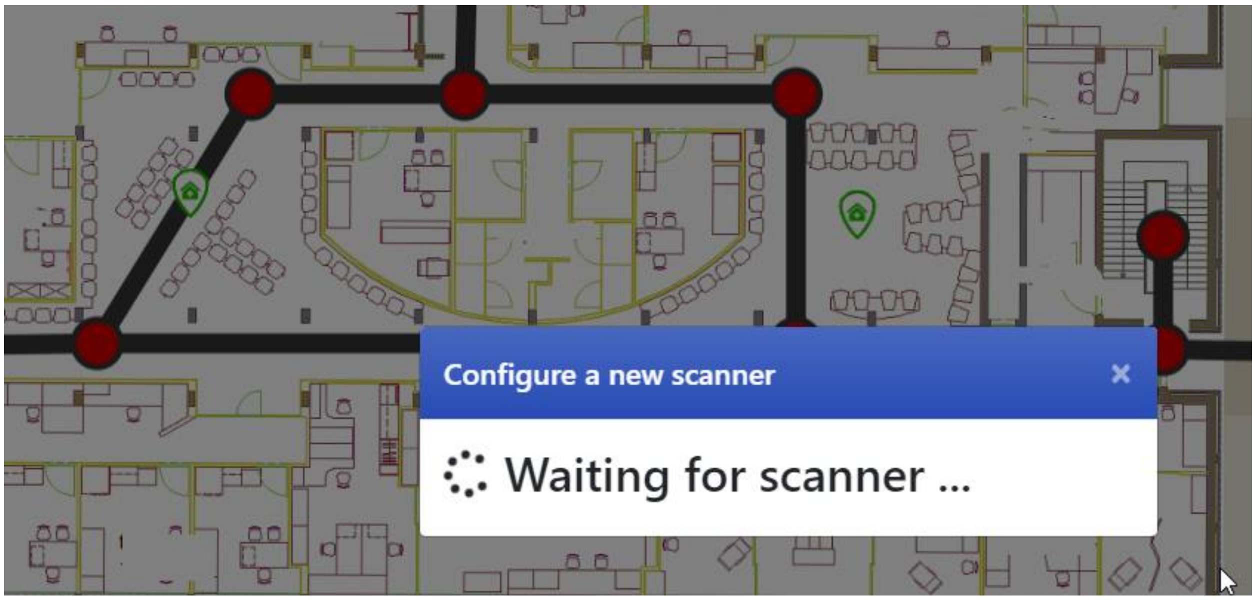


Figure 52 - Tracking server right click context menu



Waiting for scanner ...

Figure 53 - Tracking server waiting for the scanner to connect through MQTT

C. Two-Point Positioning

As mentioned at the end of chapter 6, we can determine a location based on two scanner nodes. This is done with the knowledge that in most cases, the scanner nodes are positioned in hallways. The mobile device that needs to be positioned is located on a straight line between these two points.

$X = Unknown\ location$

$A = Scanner\ A$

$B = Scanner\ B$

$|AX| = Distance\ between\ scanner\ A\ and\ X\ in\ meters$

$|BX| = Distance\ between\ scanner\ B\ and\ X\ in\ meters$

Used as an example, Figure 30 shows two scanners that are in a hallway at UZ Brussel. The chosen hallway on the ground floor is for administrative purposes and therefor does not require a dense scanner grid.

For demonstration purposes and coincidental orientation of the building, the longitude of the two scanner nodes is the same. However, the two-point positioning works using bearings to make sure it works on different orientations as well.

Calculate the bearing and distance between the two points

The two coordinates of the scanners are used to determine a bearing from A to B and B to A. This is the direction in a straight line from one point to another using the *Forward Azimuth formula*.

Over large distances, the bearing of the starting point and destination point will be different. It is not expected that the bearing will have a huge variation in the marginal distances used for indoor positioning, but this small additional step has been made to keep the calculation mathematically correct.

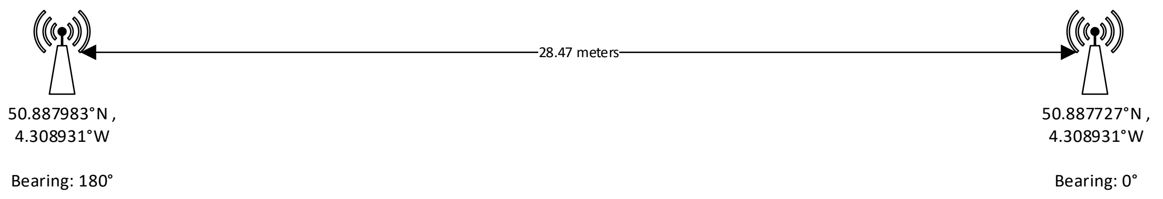


Figure 54 - Two-point calculated distance and bearings

Longitude and Latitude in radians

$$\Phi_A = A_{lat} * \frac{\pi}{180}\ , \Phi_B = B_{lat} * \frac{\pi}{180}\ ; \lambda_A = A_{lon} * \frac{\pi}{180}\ , \lambda_B = B_{lon} * \frac{\pi}{180}$$

Bearing from A to B (in radians)

$$y_{AB} = \sin(\lambda_B - \lambda_A) * \cos(\Phi_B)$$
$$x_{AB} = \cos(\Phi_A) * \sin(\Phi_B) - \sin(\Phi_A) * \cos(\Phi_B) * \cos(\lambda_B - \lambda_A)$$
$$brng_{AB} = atan2(y_{AB}, x_{AB})$$

Bearing from B to A (in radians)

$$y_{BA} = \sin(\lambda_A - \lambda_B) * \cos(\Phi_A)$$
$$x_{BA} = \cos(\Phi_B) * \sin(\Phi_A) - \sin(\Phi_B) * \cos(\Phi_A) * \cos(\lambda_A - \lambda_B)$$
$$brng_{BA} = atan2(y_{BA}, x_{BA})$$

Using the *Haversine formula* we can calculate the distance between the two scanners (Veness, 2011). This is the shortest distance on a sphere, in this case the curvature of the Earth. This distance will be used to estimate the accuracy of the position.

This is not the most accurate way to calculate the distance between two points because the Earth is not an exact sphere. Other algorithms such as *Vincenty's formula* use an accurate ellipsoidal model of the Earth to perform the calculations (Veness, 2011). Additional accuracy is negligible due the already inaccurate RSSI to distance conversion (see section 8.2.4). Even the Haversine formula will only achieve a difference of less than 1 cm over a 40-meter distance by taking the curvature into account. Since we are starting from the geographical coordinates of our scanners, it seems only correct to follow the geographical calculations instead of ignoring these.

$$R = Mean\,radius\ of\ the\ Earth = 6371\ km = 6371000\ meters$$

$\Delta\Phi = (B_{lat} - A_{lat}) * \frac{\pi}{180}$ *Difference of latitude between the two points*

$\Delta\lambda = (B_{lon} - A_{lon}) * \frac{\pi}{180}$ *Difference of longitude between the two points*

$d_{AB} = Distance\ between\ A\ and\ B$

$$d_{AB} = 2R * atan2\left(\left(\sqrt{\sin\left(\frac{\Delta\Phi}{2}\right) * \sin\left(\frac{\Delta\phi}{2}\right) + \cos(\Phi_A) * \cos(\Phi_B) * \sin\left(\frac{\Delta_\lambda}{2}\right) * \sin\left(\frac{\Delta\lambda}{2}\right)}\right), \left(\sqrt{1 - \left(\sin\left(\frac{\Delta\Phi}{2}\right) * \sin\left(\frac{\Delta\phi}{2}\right) + \cos(\Phi_A) * \cos(\Phi_B) * \sin\left(\frac{\Delta_\lambda}{2}\right) * \sin\left(\frac{\Delta\lambda}{2}\right)\right)}\right)\right)$$

<u>Calculate the expected position between each scanner</u>

Once the bearing and distance between the two scanners is known it is possible to calculate two reference points on a straight line between A and B. These reference points use the approximated distance between the scanners.

Shown in Figure 32, the two points do not lay on top of each other. This is due to the inaccuracies in the distance approximation. Any floating-point rounding error can be determined by getting the absolute subtraction of the expected distance ($d_{AB} - |AX| - |BX|$) with the actual distance between point C and D (see Haversine formula used to calculate d_{AB}). In this case, with a floating-point accuracy of 6 digits, the error is $1.22\ meters$.

$C = Point\ originating\ from\ A\ with\ bearing\ A\ to\ B\ (brng_{AB})\ and\ distance\ |AX|$

$D = Point\ originating\ from\ B\ with\ bearing\ B\ to\ A\ (brng_{BA})\ and\ distance\ |BX|$

Calculate point C

$$\Phi_C = \arcsin\left(sin(\Phi_A) * cos\left(\frac{|AX|}{R}\right) + cos(\Phi_A) * sin\left(\frac{|AX|}{R}\right) * cos(brng_{AB})\right)$$

$$\lambda_C = \lambda_A + atan2\left(\sin(brng_{AB}) * \sin\left(\frac{|AX|}{R}\right) * \cos(\Phi_A), \cos\left(\frac{|AX|}{R}\right) - \sin(\Phi_A) * \sin(\Phi_C)\right)$$

$$C_{lat} = \Phi_C * \frac{180}{\pi}\ ; C_{lon} = \lambda_C * \frac{180}{\pi}$$

Calculate point D

$$\Phi_D = \arcsin\left(sin(\Phi_B) * cos\left(\frac{|BX|}{R}\right) + cos(\Phi_B) * sin\left(\frac{|BX|}{R}\right) * cos(brng_{BA})\right)$$

$$\lambda_D = \lambda_B + atan2\left(\sin(brng_{BA}) * \sin\left(\frac{|BX|}{R}\right) * \cos(\Phi_B), \cos\left(\frac{|BX|}{R}\right) - \sin(\Phi_B) * \sin(\Phi_D)\right)$$

$$D_{lat} = \Phi_D * \frac{180}{\pi}\ ; D_{lon} = \lambda_D * \frac{180}{\pi}$$

Calculate the midpoint of point C and D

We can use the two reference points calculated in the previous step to determine the midpoint

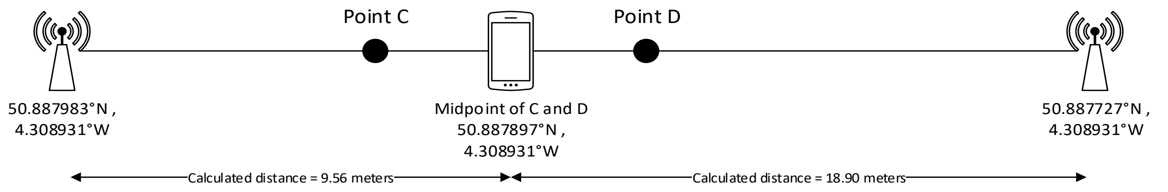

Point C
Point D
50.887983°N , 4.308931°W
Midpoint of C and D 50.887897°N , 4.308931°W
50.887727°N , 4.308931°W
Calculated distance = 9.56 meters
Calculated distance = 18.90 meters

Figure 55 - Midpoint average of two reference points (C, D)

$$Bx = \cos(\Phi_D) * \cos(\lambda_D - \lambda_C)$$
$$By = \cos(\Phi_D) * \sin(\lambda_D - \lambda_C)$$
$$\Phi_X = atan2\left(\sin(\Phi_C) + \sin(\Phi_D), \sqrt{(\cos(\Phi_C) + Bx) * (\cos(\Phi_C) + Bx) + By^2}\right)$$
$$\lambda_X = \lambda_C + atan2(By, \cos(\Phi_C) + Bx)$$

$$X_{lon} = \Phi_X * \frac{180}{\pi}$$
$$X_{lat} = \lambda_X * \frac{180}{\pi}$$

This provides us with the longitude and latitude of our mobile device situated on the direct line between the two scanners.

D. Trilateration Positioning

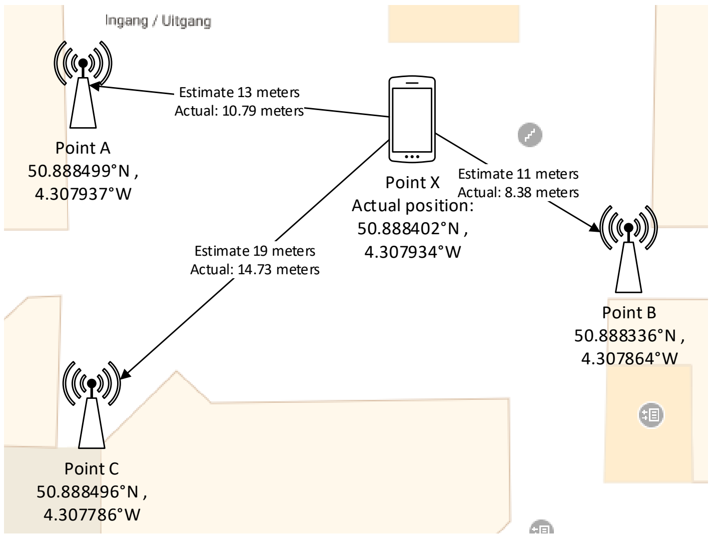


Figure 56 - Trilateration positioning in larger open spaces

Other than two-point positioning, trilateration does not assume the mobile device moves in a straight line between scanners.

In the example shown in Figure 34 the position X will be estimated by using three rough reference distances.

The example is situated in a large entrance hall at the pediatrics section of UZ Brussel.

Trilateration will mostly be applied/possible in open spaces such as this one where elevators, staircases, appointment registration desks and other possible destinations are located throughout this area.

$X = Unknown\ location$

$A = Scanner\ A$

$B = Scanner\ B$

$C = Scanner\ C$

$|AX| = Distance\ between\ A\ and\ X\ in\ meters$

$|BX| = Distance\ between\ B\ and\ X\ in\ meters$

$|CX| = Distance\ between\ C\ and\ X\ in\ meters$

<u>Convert the location of the three scanner nodes to ECR positions</u>

Convert the longitude and latitude of the three scanners to ECR/ECEF locations (earth-centered rotational/earth-centered earth-fixed). This is a Cartesian coordinate system with the center (0,0,0) being the center of mass of the Earth. Just like the Haversine formula in section 9.3.2 this conversion assumes the Earth is a perfect sphere.

$R = Mean\ radius\ of\ the\ Earth = 6371\ km = 6371000\ meters$

$\Phi_A = A_{lat} * \frac{\pi}{180}$; $\lambda_\mathrm{A} = A_{lon} * \frac{\pi}{180}$

Geographical coordinates to radians $(\Phi = latitude, \lambda = longitude)$

$\Phi_B = B_{lat} * \frac{\pi}{180}$; $\lambda_B = B_{lon} * \frac{\pi}{180}$

$\Phi_C = C_{lat} * \frac{\pi}{180}$; $\lambda_C = C_{lon} * \frac{\pi}{180}$

Convert point A to ECR position

$x_A = R * \cos(\Phi_A) * \cos(\lambda_A)$

$y_A = R * \cos(\Phi_A) * \sin(\lambda_A)$

$z_A = R * \sin(\Phi_A)$

$P_1 = (x_A, y_A, z_A)$

Convert point B to ECR position

$x_B = R * \cos(\Phi_B) * \cos(\lambda_B)$

$y_B = R * \cos(\Phi_B) * \sin(\lambda_B)$

$z_B = R * \sin(\Phi_B)$

$P_2 = (x_B, y_B, z_B)$

Convert point C to ECR position

$x_C = R * \cos(\Phi_C) * \cos(\lambda_C)$

$y_C = R * \cos(\Phi_C) * \sin(\lambda_C)$

$z_C = R * \sin(\Phi_C)$

$\mathrm{P}_3 = (\mathrm{x}_\mathrm{C}, \mathrm{y}_\mathrm{C}, z_B)$

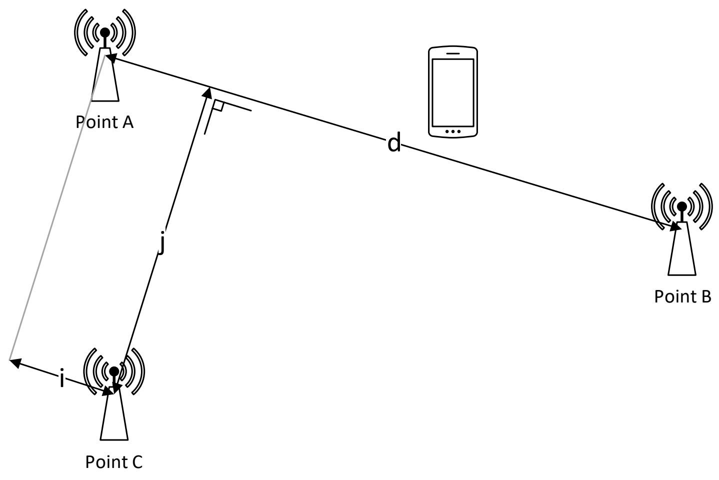


Figure 57 – Trilateration signed magnitude of the x and y components (i, j)

Use these coordinates to create three spheres centered around these locations with a radius that is equal to the approximated distance between each scanner and the mobile device. These spheres have their center position on the $z = 0$ plane. In the following calculations, the three ECR points (P_1, P_2, P_3) will be used as vectors.

$\hat{e}_x = \frac{P_2 - P_1}{\|P_2 - P_1\|}$ *Unit vector in the positive x direction from P_1 to P_2*

$i = \hat{e}_x * (P_3 - P_1)$ *Distance between P_1 and the perpendicular intersection of P_3*

$\hat{e}_y = \frac{P_3 - P_1 - i*\hat{e}_x}{\|P_3 - P_1 - i*\hat{e}_x\|}$ *Unit vector in the positive y direction perpendicular to the vector $i * \hat{e}_X$*

$j = \hat{e}_y * (P_3 - P_1)$ *Distance between P_3 and the perpendicular intersection of*

$\hat{e}_z = \hat{e}_x * \hat{e}_y$ *Unit vector of the third base*

$d = \|P_2 - P_1\|$ *Distance between the centers of P_1 and P_2 (scanner A and scanner B)*

<u>Calculate the coordinates of the mobile device</u>

Distance 'x' of the unknown location originating at P_1

$$x = \frac{|AX|^2 - |BX|^2 + d^2}{2 * d}$$

Distance 'y' of the unknown location originating at P_1

$$y = \left(\frac{|AX|^2 - |CX|^2 + i^2 + j^2}{2 * j}\right) - \left(\left(\frac{i}{j}\right) * x\right)$$

$$z = \sqrt{|AX|^2 - x^2 - y^2}$$

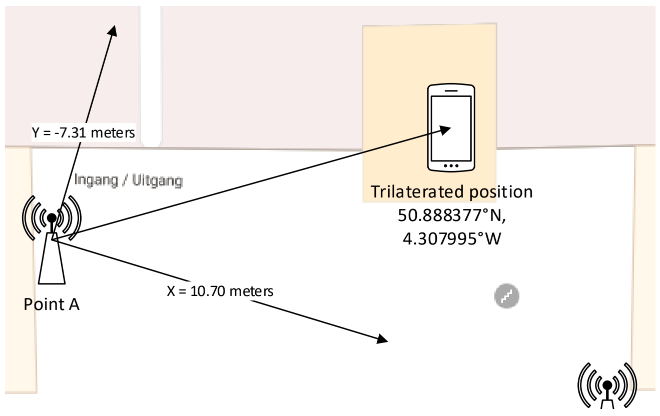


Figure 58 - x, y distance originating from scanner A in the direction of the (x,y) unit vectors

In the provided example, the three reference distances ($|AX|, |BX|, |CX|$) have a common intersection. This is not always the case if the reference distance is lower than the actual distance.

When this happens, $|AX|^2 - x^2 - y^2$ will be less than 0 causing an impossible solution for the common intersection. To solve this issue, the radius of all spheres will be equally increased by 10 centimeters until an intersection is found.

ECR point of point X using the origin (P_1) and the distance 'x' and 'y' in the direction of the unit vectors $\hat{e}_x$ and $\hat{e}_y$

$$(x_X, y_X, z_X) = P_1 + \hat{e}_x * x + \hat{e}_y * y + \hat{e}_z * z$$

$$X_{lat} = \operatorname{asin}\left(\frac{z_X}{R}\right) * \frac{180}{\pi}$$

$$X_{lon} = atan2(y_X, x_X) * \frac{180}{\pi}$$

This is the longitude and latitude of the mobile device based of three reference points.

E. Tracking Server Internal Route Graph

```
let graph: Graph = new Graph();
// Add routes to the graph
App.instance.routeController.findGraph(siteId).then(routeGraph => {
    // For each route node, add it to the graph
    routeGraph.nodes.forEach(node => {
        let vertex = new Vertex();
        vertex.legacyId = node.id;
        vertex.lon = node.lon;
        vertex.lat = node.lat;
        vertex.type = node.type;
        vertex.floor = node.floorId;
        graph.vertices.push(vertex);
    });
    // Add all route edges to the graph
    routeGraph.edges.forEach(routeEdge => {
        let edge = new Edge();
        edge.legacyId = routeEdge.id;
        graph.vertices.forEach(vertex => {
            if (vertex.legacyId == routeEdge.entryNodeId){
                edge.start = vertex;
                vertex.edges.push(edge);
            }else if (vertex.legacyId == routeEdge.exitNodeId){
                edge.end = vertex;
                vertex.edges.push(edge);
            }
        });
        edge.weight = routeEdge.weight; // Weight of the route is already converted to time
        graph.edges.push(edge);
    });

    // Find all destinations
    App.instance.mapItemController.findSiteItems(siteId).then(items => {
        // Adjust graph. For each item there will be newly generated edges
        items.forEach(item => {
            // Step 1: Add the vertex to the graph
            // This vertex is the destination
            let itemVertex = new Vertex();
            itemVertex.legacyId = item.id;
            itemVertex.destination = true;
            itemVertex.name = item.name;
            itemVertex.icon = item.icon;
            itemVertex.lon = item.lon;
            itemVertex.lat = item.lat;
            itemVertex.floor = item.floorId;
            graph.vertices.push(itemVertex);

            let possibleLocation = null;
            let shortestDistance = Number.MAX_VALUE;
            let closestEdge = null;
            // Find the closest edge in the graph to the destination
            graph.edges.forEach(edge => {
                // Line
                let A = Location.toLocation(edge.start).toECR();
                let B = Location.toLocation(edge.end).toECR();
                // Point
                let mapItemLocation = Location.toLocation(item);
                let X = mapItemLocation.toECR();

                let AB = math.subtract(B,A);        // Vector A to B
                let AP = math.subtract(X,A);        // Vector A to X

                let dot = math.multiply(AP,AB);
                let len = math.multiply(AB,AB);
                let t = Math.min( 1, Math.max( 0, math.divide(dot,len) ) );
                let P_X_snapped = math.add(A,math.multiply(t,AB));
                let P_X_location = Location.fromECR(P_X_snapped);
                let distance = P_X_location.distance(mapItemLocation);
                if (shortestDistance > distance){
                    possibleLocation = P_X_location;
                    shortestDistance = distance;
                    closestEdge = edge;
                }
            });

            // Step 2: Split the "closestEdge" at location possibleLocation
            // Create two new edges that connect this newly "itermediate" vertex
            let addedVertex = new Vertex();
            addedVertex.legacyId = "CREATED_FOR_" + itemVertex.legacyId;
            addedVertex.lon = possibleLocation.lon;
            addedVertex.lat = possibleLocation.lat;
            addedVertex.floor = possibleLocation.floorId;
            graph.vertices.push(addedVertex);
            let newEdge1 = new Edge();
            newEdge1.start = closestEdge.start;
            newEdge1.legacyId = "CREATED_FOR_" + itemVertex.legacyId;
            newEdge1.end = addedVertex;
            newEdge1.directed = closestEdge.directed;
            newEdge1.weight =
Location.toLocation(newEdge1.start).distance(Location.toLocation(newEdge1.end)) * 1.4;
            closestEdge.start.edges.push(newEdge1);
            addedVertex.edges.push(newEdge1);
            let newEdge2 = new Edge();
            newEdge2.start = addedVertex;
            newEdge2.legacyId = "CREATED_FOR_" + itemVertex.legacyId;
            newEdge2.end = closestEdge.end;
            newEdge2.directed = closestEdge.directed;
            newEdge2.weight =
Location.toLocation(newEdge2.start).distance(Location.toLocation(newEdge2.end)) * 1.4;
            closestEdge.end.edges.push(newEdge2);
            addedVertex.edges.push(newEdge2);

            graph.edges.push(newEdge1);
            graph.edges.push(newEdge2);
            // Remove closest edge
            graph.edges.splice(graph.edges.indexOf(closestEdge),1);
            closestEdge.start.edges.splice(closestEdge.start.edges.indexOf(closestEdge),1)
            closestEdge.end.edges.splice(closestEdge.end.edges.indexOf(closestEdge),1)

            // Step 3: Create a new edge between "itemVertex" and
            // the intermediate vertex that was added in step 2
            let newEdge3 = new Edge();
            newEdge3.legacyId = "CREATED_FOR_" + itemVertex.legacyId;
            newEdge3.start = itemVertex
            newEdge3.end = addedVertex;
            newEdge3.route = false;
            newEdge3.directed = false;
            newEdge3.weight =
Location.toLocation(newEdge3.start).distance(Location.toLocation(newEdge3.end)) * 1.4;
            itemVertex.edges.push(newEdge3);
            addedVertex.edges.push(newEdge3);
            graph.edges.push(newEdge3);
        });

        // Calculate shortest paths
        let graphWithRoutes = App.instance.navigationController.calculateRoutes(siteId, graph);
        this.siteGraphs.set(siteId,graphWithRoutes);
    });
});
```

F. Tracking Server Shortest Path

```
// Create a minimum distance matrix of size |V| x |V|
let dist: number[][] = [];
// Create a vertex matrix of size |V| x |V|
let next: Vertex[][] = [];

// Initialize the distance and vertex matrices
for (let i = 0; i < graph.vertices.length; i++) {
 dist[i] = [];
 next[i] = [];
 for (let j = 0; j < graph.vertices.length; j++) {
  dist[i][j] = Number.MAX_VALUE;
  next[i][j] = null;
 }
}

// Insert the weight of all edges to the distance matrix
// and construct the 'next' moves
graph.edges.forEach(edge => {
 dist[graph.vertices.indexOf(edge.start)][graph.vertices.indexOf(edge.end)] = edge.weight;
 next[graph.vertices.indexOf(edge.start)][graph.vertices.indexOf(edge.end)] = edge.end;
 // If the edge is not directed, also at the other direction (end to start)
 if (!edge.directed) {
  dist[graph.vertices.indexOf(edge.end)][graph.vertices.indexOf(edge.start)] = edge.weight;
  next[graph.vertices.indexOf(edge.end)][graph.vertices.indexOf(edge.start)] = edge.start;
 }
});

// Set the weight of all vertices to themselves to 0
graph.vertices.forEach(vertex => {
 dist[graph.vertices.indexOf(vertex)][graph.vertices.indexOf(vertex)] = 0;
 // The next move when source=target is itself
 next[graph.vertices.indexOf(vertex)][graph.vertices.indexOf(vertex)] = vertex;
});

// Floyd-Warshall implementation
for (let k = 0; k < graph.vertices.length; k++) {
 for (let i = 0; i < graph.vertices.length; i++) {
  for (let j = 0; j < graph.vertices.length; j++) {
   // If the distance between vertex i and j is greater than the
   // distance between i and k, and k to j
   if (dist[i][j] > dist[i][k] + dist[k][j]) {
    // Then the faster route from i to j is by passing through k
    dist[i][j] = dist[i][k] + dist[k][j];
    next[i][j] = next[i][k];
   }
  }
 }
}

// Use the next matrix to determine the route/path to all destinations
graph.vertices.forEach(source => {
 // Only calculate the routes FROM non destinations
 if (!source.destination) {
  graph.vertices.forEach(end => {
   // Only calculate the routes TO destinations
   if (end.destination) {
    let nextMove = next[graph.vertices.indexOf(source)][graph.vertices.indexOf(end)];
    if (nextMove != null) {
     // Store path in graph linking a pair (source|target) to the next vertex
     graph.addRoute(source, end, nextMove);
    }
   }
  });
 }
});
```

G. Mini Map Drawing

```
/**
 * Generate a mini map image
 *
 * @param data JSON mini map
 * @param backgroundColor Background line color
 * @param foregroundColor Foreground line color
 * @returns promise for a base64 encoded image
 */
public generateImage(data, backgroundColor,forgroundColor) : Promise<string> {
    return new Promise((resolve,reject) => {
        // 'data' contains our JSON data
        let floorNumber = data.floor;

        // Create a new canvas with any size
        let canvas = createCanvas(512,512);

        let ctx = canvas.getContext('2d');
        ctx.translate(256, 256);  // Origin in center
        ctx.scale(70,-70);        // Cartesian coordinates with correct scaling for 7x7

        // Render gray background lines
        data.background.forEach(line => {
            ctx.beginPath();
            ctx.lineWidth = 0.8;
            ctx.strokeStyle = backgroundColor;
            ctx.lineCap = "round";
            ctx.moveTo(line.x1, line.y1);
            ctx.lineTo(line.x2 , line.y2);
            ctx.stroke();
        });
        // Draw the green foreground lines on top
        data.foreground.forEach(line => {
            let lastLine = data.foreground[data.foreground.length - 1] == line;
            ctx.beginPath();
            ctx.lineWidth = 0.8;
            ctx.strokeStyle = forgroundColor;
            ctx.lineCap = "round";
            let angle = Math.atan2(line.y2 - line.y1, line.x2 - line.x1);
            ctx.moveTo(line.x1, line.y1);
            ctx.lineTo(line.x2 , line.y2);
            ctx.stroke();

            // Check if an arrow head needs to be drawn
            if (lastLine){
                // Clear the place where the head will be drawn
                ctx.beginPath();
                ctx.lineWidth = 0.9;
                ctx.strokeStyle = "#FFFFFF";
                ctx.lineCap = "round";
                ctx.moveTo(line.x2, line.y2);
                ctx.lineTo(line.x2 - 0.9 * Math.cos(angle) , line.y2 - 0.9 * Math.sin(angle));
                ctx.stroke();
// Draw the arrow head
                // Partly based on source: https://stackoverflow.com/questions/808826/draw-arrow-
on-canvas-tag
                ctx.beginPath();
                ctx.moveTo( line.x2,  line.y2);
                ctx.lineTo( line.x2 - 1.5 * Math.cos(angle - Math.PI/7),line.y2 - 1.5 *
Math.sin(angle - Math.PI/7));
                ctx.lineTo( line.x2 - 1.5 * Math.cos(angle + Math.PI/7),line.y2 - 1.5 *
Math.sin(angle + Math.PI/7));
                ctx.lineTo( line.x2, line.y2);
                ctx.lineTo( line.x2 - 1.5 * Math.cos(angle - Math.PI/7),line.y2 - 1.5 *
Math.sin(angle - Math.PI/7));
                ctx.fillStyle = forgroundColor;
                ctx.fill();
            }
        });

        // Return the base64 image
        resolve(canvas.toDataURL());
    });
}
```