An Indoor Localization System for Nursing Homes Based on RSSI

BSc Thesis

Marijn Boringa, Job van Erp, Corné Ploumen

TU Delft

TU Delft

An Indoor Localization System for Nursing Homes Based on RSSI

Bachelor Graduation Thesis

By
Marijn Boringa, Job van Erp, and Corné Ploumen

Commissioned by
Momo Medical

December 27, 2022

Student number:	M.W. Boringa	5028574
	J.J.M. van Erp	4226798
	C.E.J.M. Ploumen	5074029

Project duration: October 17, 2022 - December 23, 2022

Thesis committee:	Dr. P.J. French	TU Delft, supervisor
	M. Gravemaker	Momo Medical, project proposer
	D. Eldering	Momo Medical, supervisor
	J. van Rijn	Momo Medical, supervisor
	Dr. ir. I.E. Lager	TU Delft
	Ing. J. Bastemeijer	TU Delft

Delft University of Technology

Faculty of Electrical Engineering, Mathematics
and Computer Science

Electrical Engineering Programme

Abstract

Over the years, care givers in nursing homes have seen their workload gradually increase. With no end of this trend in sight, the need for smart support systems increases. Especially systems which decrease the time spent on menial tasks are valued highly, because this frees more time for high quality and personal care.

To achieve this, Momo Medical is expanding on its nurse support system. This is a network of integrated smart solutions aiming at supporting care givers in nursing homes to provide better, faster and more personal care. The backbone of this nurse support system consists of the BedSenses, sensors which are placed under mattresses of each resident and can measure a variety of things.

This thesis describes the process of designing and creating a localization algorithm for this nurse support system. This algorithm can find residents by tracking the panic buttons they wear, so that in case of an emergency or whenever a care giver wants to know where a resident is located, they do not need to undertake a time consuming search in order to find them. These panic buttons send out an alive signal once every minute as well as a signal whenever the button is pressed. These signals are received by any nearby BedSenses. The algorithm looks at the signal strength with which each BedSense receives these signals and uses this to perform localization.

Preface

This bachelor graduation project has been an interesting and challenging experience. As a team, we wanted to work on a project that would yield tangible results. Not only did we find such a project, but we also became part of a Bachelor Graduation Project team that was really welcoming and open.

We have learned quite some important things in this project. On the one hand we had to struggle with the many theoretical findings and we encountered problems that we had to overcome by thinking creatively and pooling our knowledge, such as RSSI measurements that tend to fluctuate significantly even if complete measurement setup seems to be completely static. We also gained valuable insight into the start up environment we were working in, the importance of team dynamics and the influence of not always following the conventional hierarchies in a company. Sprint meetings according to the SCRUM method, retrospective meetings to reflect on the way of working, and the legendary VrijMiBo, to name but a very few!

We hope that this thesis will help future students to have a feeling of how the bachelor graduation project can possibly go at the TU Delft. Readers that are specifically interested in some possible ways of setting up a localization system are advised to focus on Chapter 4.

We would like to first hand show our gratitude for the support and coaching we got from the Momo Medical team, especially Danny Eldering and Joey van Rijn for being always available for questions and taking part in our meetings. We would like to thank Dr. Paddy French for supervising us on behalf of the TU Delft. We would like to thank Dr. Ioan Lager and Menno Gravemaker for giving us this opportunity. At last (but not least), we want to thank colleague subgroup (Arthur Bennebroek and Shayan Ramezani) for their passion and the frustrating, but above all the joyful moments we have shared together.

Delft, The Netherlands
December 27, 2022
Marijn Boringa, Job van Erp and Corné Ploumen

Contents

List of Abbreviations

ACS	Access Control System
AoA	Angle of Arrival
AGC	Automatic Gain Control
CDF	Cumulative Density Function
EM	Electromagnetic
GPS	Global Positioning System
ID	Identity
LoS	Line-of-Sight
NBSA	Nearest BedSense Algorithm
PDF	Probability Density Function
PL	Path Loss
RF	Radio Frequency
RSS	Received Signal Strength
RSSI	Received Signal Strength Index
RTLS	Real-Time Locating System
RTL	Real-Time Localizing
TDoA	Time Difference of Arrival
ToA	Time of Arrival
ToF	Time of Flight

1 Introduction

Momo Medical[1], a startup company, is developing new technologies so that care givers can work more efficiently in the nursing homes and therefore are able to give more specific and better care to their residents. At the moment, their main product is a sensorplate, the BedSense, which is placed underneath the mattress of a bed and together with the corresponding app gives a better insight into the behavior and the needs of the resident while he/she is sleeping. It notifies the care giver when a resident is about to get out of bed and gives insight in the state of a residents wellbeing. Also they designed a smart incontinence pad which lets the care giver know if the incontinence pad is full and needs to be changes. They are always looking for more ways to help the care givers work more efficiently.

1.1. Problem Definition

Currently most nursing homes are leaving the concept of closed departments behind. In this old system, residents were locked up in their own department and were not allowed to move freely through the nursing home. In this case, locked up literally means locking the door between the different departments so that the residents were not able to leave their own department. As this is ethically questionable, more and more nursing homes are switching to a system with open departments in which the residents are allowed to move more freely between different departments. However, this introduces new challenges, such as residents being able to wander to places they are not allowed to go and that whenever a resident needs help, they are a much more difficult to find.

One problem regarding the change to open departments, is that some but not all residents are allowed to go to other departments. Psychogeriatric residents (with severe dementia) are not allowed to leave their department while the other residents are allowed to leave. To solve this, a system needs to be designed that is able to recognize which resident is trying to pass through a door to another department. Depending on whether or not they are allowed to go there, the system locks or unlocks the door. This system can also notify care givers when someone, who is not allowed, is trying to pass a door.

In most of these nursing homes, residents wear a panic button, which they can press if they need help. Pressing this button will send a signal to the care giver that the resident needs attention. However, as the resident is no longer confined to a single department, it can be very challenging to find the resident in need. This search can take up a lot of the care givers' valuable time. To prevent this loss of time and also decrease the workload of the care givers, the panic button needs to be localized as this often is the only trackable device the residents always have with them. By localizing the panic button, care givers will now be able to find the resident faster when they are in need of help.

1.2. Project Overview

For this project, two systems will be designed for Momo Medical. The first system is an access control system which will allow or not allow a resident access to other parts of a nursing home. In addition, the system setup can be used for other goals, e.g. notifying a care giver when a resident is trying to exit his/her room at night. This system is completely described in its corresponding thesis [2].

The second system is a real-time localization system which will determine the location of a resident inside a nursing home. The care givers can either request the location of the resident, or, whenever the panic button is pressed, find the location within the message indicating that the resident needs help. In addition, this system could be set up in a way that it can also localize utilities such as lifting aids and keys. Furthermore, the system can also be used by care givers whenever they are in need of help themselves.

A schematic overview of the complete project is shown in Figure 1.1. In this figure, a clear division can be seen between the two different part of the project, which will work together on the localization of a resident. These two parts, which are different systems, will not communicate directly, but only via the Momo Medical database. For both systems, data is obtained from the database. For the localization system, this data will also contain the output of the access control system. The messages for the care givers will come from the server and will be visible in the Momo Medical app.

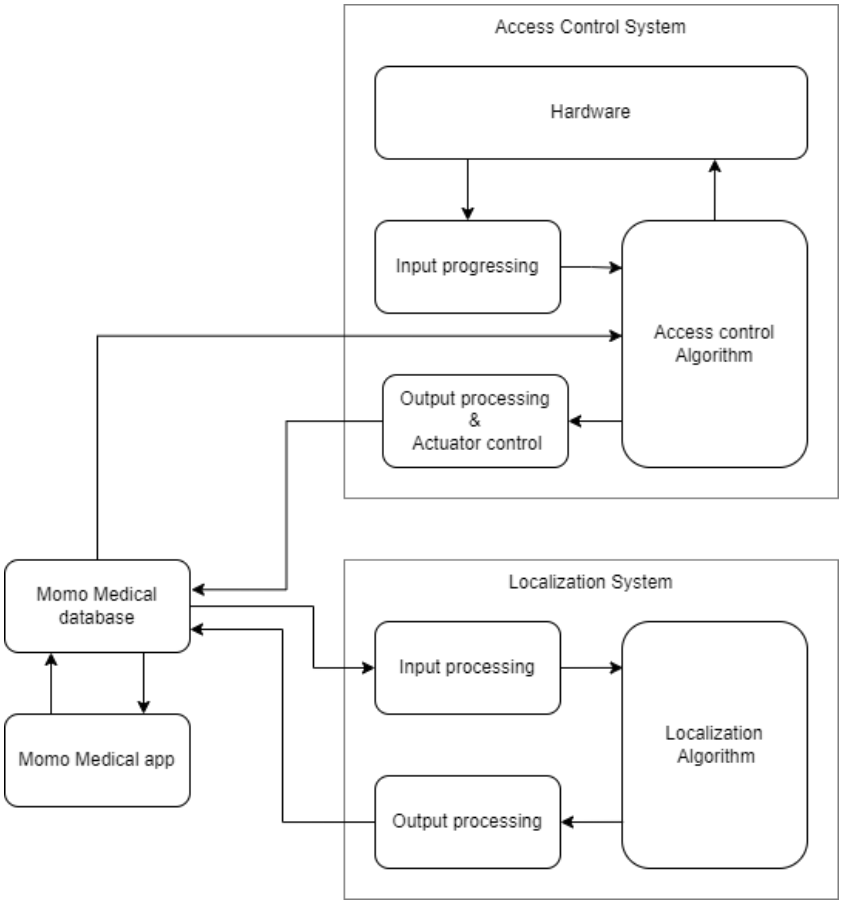


Figure 1.1: Schematic project overview.

1.3. Thesis Synopsis

This thesis concerns the localization subgroup that tackles the localization of the panic button within the walls of a nursing home. Chapter 2 states the program of requirements, gives some optional goals and defines the scope limitations. Chapter 3 explains the choices made during the design of the prototype using the theory and gives a detailed explanation of the measurements done. Chapter 4 gives an explanation of the prototype, its functions and the choices made during programming. Furthermore, Chapter 5 discusses the results and Chapter 6 concludes the thesis. Finally, Chapter 7 gives recommendations and input for future works.

1.4. State-of-the-art Analysis

Within the field of localization, a lot of research is already been done and a lot of different methods can be used for indoor localization specifically as well. Other methods like GPS (Global Positioning System) cannot be used due to severe degradation together with indoor environments as the satellite or cellular signals are interrupted. Partial localization techniques like Time of Flight (ToF), Time Difference of Arrival (TDoA), Angle of Arrival (AoA) and Received Signal Strength Index (RSSI) are methods which are much better suited for indoor localization. [3]

Three different algorithms can be used based on these different techniques. Trilateration is based on the distance measurement between three receivers and a transmitter. This is an algorithm mostly used for ToF, TDoA and RSSI. Multilateration is similar to trilateration only are more receivers used. The last algorithm is triangulation which is mostly used in combination with AoA. Here trigonometry laws of sine and cosine are used to calculate the distance between the transmitters and the receiver.[4]

1.4.1. Localization Techniques

ToF

Both the methods of ToF and TDoA are based on the time it takes for a signal to travel through the medium from transmitter to receiver. ToF exploits this signal propagation time to calculate the distance between receiver and transmitter. Then using the same method for at least two other receivers, the location of the transmitter can be determined via trilateration, or multilateration if more than three receivers are used. However ToF requires a strict synchronization between both transmitter and receiver because the exact time of transmission and receiving must be known. [5]

TDoA

TDoA can be derived by the ToF difference between multiple signals transmitted synchronously [6]. To do this clock synchronization is needed between the different receivers [5]. By comparing the hyperbolas, the result of the difference between the ToF signals, a location can be determined via tri- or multilateration [7].

A drawback for both the methods ToF and TDoA is that the signals which are send, are traveling through air at a speed close to the speed of light. If the receiver has a clock rate in the order of hundreds of MHz or less, it would not be near fast enough to determine the time of arrival with a high accuracy as the flight time is only a fraction of the clock period.

AoA

The method of AoA makes use of antenna arrays to determine the angle at which the signal is received [5]. An advantage of AoA is that it is a range-free method of localization. It does however, uses two or more fast-scanning multi-beam antennas. The intersection of the beams marks an area in which the object should be located [8, 9]. A drawback of Angle of Arrival is that it can not be used without the correct antenna hardware, as will be explained in Section 2.2.5.

RSSI

Lastly, RSSI is a widely used method for localization. This method uses the estimate of the signal energy of the received signal to estimate the distance that the signal has traveled. To estimate this, a path loss exponent, n, needs to be determined. The relation between the received power and the distance is given in Equation 1.1. In this equation $\hat{P}_{RX}$ (dBm) is the received signal power, P_{TX} (dBm) is the transmitted power and $\hat{d}$ (m) is the estimate of the distance. PL_0 (dBm) is the path loss of the reference distance d_0 (m). [10, 11, 12, 13]

$$\hat{P}_{RX} = P_{TX} - 10nlog_{10}\Big(\frac{\hat{d}}{d_0}\Big) - PL_0 \tag{1.1}$$

The method of RSSI is widely used for localization because, in theory, the accuracy is independent of the signal bandwidth. This allows for the use of popular, already existing networks like a WiFi network. However this method also has a lot of drawbacks. One of those is that the path loss exponent n, is unknown and can only be roughly estimated [10]. Also this exponent differs a lot depending on the environment and what is in it. For example walls and floors have a big influence on the received signal strength but modeling it in the past loss exponent has proven to be difficult and inaccurate [14, 15].

1.4.2. Methods to Improve Localization Techniques

Chapter 4 introduces several algorithms that process RSSI data from BedSenses and data from the ACS. These algorithms can be improved by adding already existing calculations and methods. Several, widely used methods are proposed in this section.

Fingerprinting

The method of fingerprinting is based on comparing the received signal strength (RSS) of all receivers in the space (fingerprint) covered by the real-time localization system (RTLS), with a radio map that was built beforehand. This radio map is a collection of RSS values of all receivers for each location within this space. Creating such a radio map involves manually visiting all locations within the space covered by the RTLS and annotating the measured RSS values at all receivers, in this case BedSenses. This process is described as collecting fingerprints. For localizing a transmitter, in this case a panic button, the measured fingerprint by a received signal from the transmitter (online phase) is compared to the radio map (offline phase) for similarities. Then, after several computational steps on the comparisons, the result represents the most probable location.

However, this method has some problems. Amongst those is the time and human resources needed to build such a radio map as each location needs to be visited separately. This increases if there is a lack of accurate indoor maps that either need to be made, or, in their absence, decrease the RTLS accuracy. Furthermore, effects by reflections and shadowing and dynamic layout changes of the environment (e.g. people moving around) highly influence the RSS by receivers. These later two problems causes mismatches in fingerprints over time. [16]

Kalman Filter

The Kalman filter is used to estimate states based on linear dynamic systems. It gives an estimate of the variable $\mathbf{x}_k$, with a state-space system, at a certain time instance k. Here, this estimated variable is the location of a panic button. The Kalman filter takes an initial estimation, $\mathbf{x}_0$ and a series of fresh measurements $\mathbf{z}_i$ for time instance k, and combines it with the description of the system in the form of matrices in the state-space model. The system includes Gaussian distributed white noise as parameters for possible error. Process model 1.2 shows the evolution of the state from time $k-1$ to k.

$$\mathbf{k}_k = F\mathbf{x}_{k-1} + B\mathbf{u}_{k-1} + \mathbf{w}_{k-1} \tag{1.2}$$

Here, F is the state transition matrix with $\mathbf{x}_{k-1}$ being the previous state vector, B is the control-input matrix with $\mathbf{u}_{k-1}$ being the control vector, and $\mathbf{w}_{k-1}$ is the process noise vector assumed to be zero-mean Gaussian distributed. [17]

The relationship between the state and the measurement at time instance k is described by the measurement model, Equation 1.3.

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k \tag{1.3}$$

Here, $\mathbf{z}_k$ is the measurement vector, H is the measurement matrix, and $\mathbf{v}$ is measurement noise vector assumed to be zero-mean Gaussian distributed. [17]

The Kalman filter method consists out of two stages: prediction and update. In the prediction stage the previous, updated state is renewed according to the way the filter estimates the state evolution. This estimation accounts for error by adding error covariance to the prediction of $\mathbf{x}_{k,pred}$ during the renewing. Next, in the update stage, the measurement residual $\mathbf{y}_k$ is calculated. This is the difference between the true measurement $\mathbf{z}_k$ and the estimated measurement - being the product of the prediction and the measurement matrix, $H\mathbf{x}_{k,pred}$. The sum of the predicted value and the updated value are multiplied by the Kalman gain K_k, to form the final estimation of the variable $\mathbf{x}_{k,upd}$ by the Kalman filter. [17, 18]

2

Program of Requirements

In order to properly assess the success of the final product, it is imperative to determine a clear set of requirements regarding this final product. These objectives are based on the problem definition in Section 1.1. First, the requirements for the full system (both ACS and localization) are given in Section 2.1. Second, the requirements of the localization system are given in Section 2.2. The functional and non-functional requirements listed in Sections 2.2.1 and 2.2.2 respectively. These requirements are then explained in Section 2.2.3. In Section 2.2.4 some optional goals/uses of the project are described. Finally, as is important in every project, is a clear description on what is not going to be done. This is therefore explained in Section 2.2.5.

2.1. Full System Requirements

As the localization system will work together with the access control system, the combination of both needs to meet several requirements.

- The possible radio frequency (RF) interference between the two systems must not affect the workings of each system.
- The system must output the location of the residents at least once a minute.
- The system must not need extensive calibration to work, meaning it has to be plug and play.
- The system must not need adjustments whilst in use.
- The system has to work in indoor environments that can differ in layout and materials used.

2.2. Localization System Requirements

The localization system itself must also meet several requirements of its own.

2.2.1. Functional Requirements

Functional requirements describe the things the final product has to be able to do and how the final product has to work.

- The location of a resident, represented by a room number, must be present in the message saying that the resident is in need of help.
- Information of the location of a resident must be presented in textual format.
- The location of a resident must be available when requested in the Momo Medical app.
- The algorithm must be able to communicate in real-time with the Momo Medical database.
- The localization system must work together with the access control system via the Momo Medical database.

- The location of the resident must be three dimensional, that is, it must be able to operate across multiple floors in a building.

2.2.2. Non-functional Requirements

Non-functional requirements state how well the final product must work and which attributes it must have.

- The accuracy of the system is defined such that the maximum error on localizing a resident is allowed to vary by one room around the actual room.
- The accuracy mentioned at the point above, must be met at least 90 % of the time.
- The location of the resident will be stored under a room number.

2.2.3. Requirement Explanation

The main use of the localization algorithm is to be able to notify care givers about the location of a resident whenever his/her alarm button is pressed. In other words, when a resident needs help, they can press the button and the caregiver will receive an alert notification which does not only contain a message about which resident needs help (current situation), but also where in the building that resident is at that moment. This information needs to be presented in a textual manner, that is, the care giver will receive a message like: 'Resident of room X needs help and is located in the west wing of the building on the second floor in room A3'. To a care giver, this format is much more useful than giving a more precise location in a more abstract format, such as relative coordinates within a building. This means that the precise location is not as relevant as long as a care giver can find a resident in need. To this end, an accuracy of 5 meters will suffice. On the other hand, it is imperative that at least the room the resident is in is correctly signified 90 percent of the time with a maximum error of one room next to the actual location of the resident.

Not only when a resident is in need of help, but also when a care giver needs to know where the resident is, the location of the resident must be available. This could for example be in case of a fire and all residents need to be located quickly. In a case like this, the care giver must be able to request the location of a resident.

In order to accomplish this, the final algorithm needs to be able to communicate with the Momo Medical database, where for each ping the RSSI values from the BedSenses are stored. Also, location data from the access control system is stored in this database and needs to be accessed in order to make the localization more accurate. It needs to access the database in real-time and after that it needs to forward its generated location-text message back to the database. From there the message is linked to the Momo Medical app so the location can be accessed by care givers. The database is accessed via the Momo Medical server. Communication to this server happens via internet provided by network signals in a building, to which the BedSenses are connected either directly or via their integrated mesh-network functionality.

As nursing homes often have multiple floors, the location of the resident must be three dimensional in order to have a precise location.

Due to privacy reasons, the location of the residents can only be stored under a room number and not under the name of the resident.

2.2.4. Optional Goals

It is not hard to imagine other uses for the developed localization algorithm. One of those is the localization of various medical instruments, such as lifting aids. If such instruments are 'tagged' with a beacon similar to those in the existing buttons, it is not difficult to see that the localization algorithm can localize those as well. This would allow care givers to more efficiently access these instruments by consulting the Momo Medical app that such an instrument is needed, after which the app returns the location of the nearest requested instrument.

It could also be used by care givers themselves in order to call for help when they cannot provide the care themselves. If a resident is in need of help and it is to much work for just one care giver, another care giver could be called by simply pressing the panic button and help will come. This system would also be of use when a resident becomes aggressive and the care giver is not able to handle the resident alone.

Even though the system must not need extensive calibration to work from the start, it can be equipped with a self-improvement function where for example the BedSense measures a signal received from a button located

in the same room as it at night in order to calibrate itself to improve reliability during the day.

2.2.5. Scope Limitations

It is not within the scope of this project to alter the existing hardware in order to improve localization or other capabilities. This will result in easy integration within the existing system and facilitates a smooth implementation on location where the BedSenses are already in use.

It might however turn out that other hardware would result in better localization, which can then be recommended to Momo Medical for future iterations of the existing hardware.

3

Design Process

During the design of the prototype, some important design decisions were made. These decisions are described in the following chapter. In Section 3.1, the decisions based on the provided devices and systems are explained. Furthermore, in Section 3.2 the theoretical behaviour of RSSI is explained as well as the decisions made based on the actual behaviour. Moreover, in Section 3.3, a common layout of a nursing home is described and Section 3.4 describes a measurement done in a nursing home. Finally, Section 3.5 the two RTLS improvement techniques (mentioned in 1.4.2) are evaluated.

3.1. Provided Devices and Systems

For this project Momo Medical provides the transmitter device, an off the shelf panic button [19, 20] specially designed for nursing homes, and a set of BedSenses [1] (being the main product of Momo Medical). The 868 MHz antenna module is the most important feature within this project. This antenna is already set up to communicate in a certain manner with the panic button used. Furthermore, the majority of the nursing homes where Momo Medical is active have these BedSenses placed under each residents' bed, capable of communicating with the panic button. However, there are some important characteristics to these two devices, the panic button and the BedSense, most of which pose limits.

The BedSenses are not highly time synchronised - not to the order of several nanoseconds. Time of Arrival (ToA) and Time Difference of Arrival (TDoA) are therefore unsuitable as proper synchronisation is needed to obtain accurate results using these methods. A solution to this is Time of Flight (ToF) in which three-step communication takes place: initially, the panic button sends a signal A to a BedSense starting the process. Upon receiving this signal, the BedSense sends a return signal B back to the panic button, which in turn responds by sending a third signal C back to the BedSense. The BedSense measures the time between the moments it sends signal B receives signal C. This measured duration consists of the ToF of signal B, the processing time of the button (which is assumed to be known), and the ToF of signal C. From this, the one-way ToF can be deduced which can be used to determine the distance between the panic button and the BedSense using the signal's expected average speed within the building.

Unfortunately, the 868 MHz transceiver in the BedSense has relatively low clock frequency. The implemented S2-LP transceiver [21] has an internal clock frequency of 50 MHz which converts to a clock period of 20 nanoseconds. RF waves, traveling at approximately speed-of-light, cover a distance of about six meters during this 20 ns clock period. This places a serious challenge on the accuracy of the system in case a ToF or TDoA based algorithm is implemented. The theoretical maximum achievable accuracy of these systems could work well for this project. However, the clock frequency of the chip should ideally be higher if fluctuations by the signal due to environmental effects are accounted for. And even if the clock frequency was high enough, the required communication protocols are not available on the existing hardware, and creation of those exceeds the project's scope.

For Angle of Arrival (AoA) based localization, specially designed antennas with high angular dependence are used to measure the incoming angle of the signal. Theoretically, with just two such antennas localization could already be performed. However, the existing BedSenses are not equipped with such antennas. There-

fore, this method is suggested as a possible future implementation, but will not be considered a solution within the scope of this project.

Currently, the comprehensive software which runs on the BedSense system provides and stores a measurement of the RSSI for each occurring communication between the button and BedSense. As mentioned in Chapter 1, RSSI can be used for localization and will be further explored in the upcoming sections. Initially, stored RSSI data can be fetched via the development interface used by Momo Medical called Grafana. In later stages, a simple additional algorithm can provide real-time reading of these measurements. Before RSSI is adopted as a starting point for the localization of the panic button, the theoretical and actual RSSI behaviour between the button and BedSense is numerically and graphically characterized.

3.2. RSSI, a Parameter for Localization

3.2.1. Theoretical Behaviour

Generally, through a lossless medium the power received by an antenna with effective area A_r (m^2) as a fraction of the transmitted power by an antenna with effective area A_t (m^2), is described by the Friis Equation [22].

$$\frac{P_r}{P_t} = \frac{A_r A_t}{\lambda^2 R^2} \tag{3.1}$$

Here λ (m) and R (m) are, respectively, the electromagnetic (EM) wavelength and the far-field line-of-sight (LoS) distance between the antennas respectively. From Equation 3.1 the path loss (PL) model for ideal circumstances is derived, Equation 3.2,

$$P(R) = P_0 - 10n\log_{10}\left(\frac{R}{R_0}\right)[dBm] \tag{3.2}$$

with P_0 (dBm) being the power received at the reference distance R_0 (m), and n being the path loss exponent. The later one is to be estimated for the type of environment [23]. For this application, inside a nursing home, the path loss exponent is initially taken to be 2.4. This value was measured inside an office for a system operating at 900 mHz by [24].

Meanwhile, the RSS is quite dynamic in itself. Many factors have an effect on RSS over time, such as noise, interference, and shadowing, but also the changes in device types, orientations, and environmental factors (e.g., temperature, humidity, open/closed door, and indoor population) [25, 26]. Even if the experimental setup is not changed, the RSS may vary significantly. The variation of RSSI can be through a Gaussian distributed random variable [27]. However, there are two practical phenomena that affect the ideal PL model: the multi-path phenomenon, also referred to as reflections, and the shadowing or large-scale fading phenomenon [18].

Reflections

Whenever a signal is transmitted in a non perfect environment, reflections of the signal can occur due to discontinuities in the transmission medium. These reflections in wireless transmission in an indoor environment can be caused by walls, floors, people and other objects. As the signal is transmitted, it travels in all directions and at some point it may hit an object which causes a reflection. The signal power hitting the object is called the incident power. With every reflection, some of the power travels through the object (transmitted power), while the rest is reflected in another direction (reflected power). The transmitted power is important for the influence of walls and floors on the received signal strength which will be discussed in 3.2.1. For now, the reflected power is of interest.

The reflected power ($\mathbf{S}^r_{av}$ (W/m^2)), see Equation 3.3, depends on the reflection coefficient of the surface (Γ) and the incident power ($\mathbf{S}^i_{av}$ (W/m^2))[28]. As the reflection coefficient is a value between -1 and 1, the reflected power will always be lower than or equal to the incident power.

$$\mathbf{S}^r_{av} = -|\Gamma|^2 \mathbf{S}^i_{av} \tag{3.3}$$

The reflection coefficient depends heavily on the surface of the object hit by the signal. For a rough surface, the reflections are more scattered, which will result in more loss. The roughness of a surface is described by the Rayleigh criterion, see Equation 3.4. In this equation, h_c (m) is the critical height which is the maximum

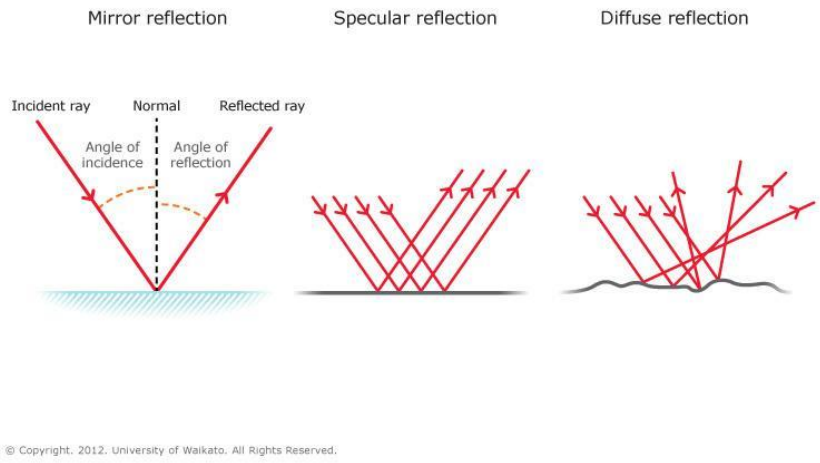


Figure 3.1: Reflection from smooth and rough surfaces.[32]

distance between the superficial heights of a given material[29] and λ (m) is the wavelength of the incident wave. Last, θ_i is the angle of incidence. It states that if the height of surface (minimum to maximum protuberance) is larger than the critical height, the surface is considered rough [30, 31]. Due to the roughness of the surface, the reflected power is not just reflected in the direction of the reflection angle, but in other directions as well (diffuse reflection), see Figure 3.1. Subsequently, the scattering loss of the surface is given by Equation 3.5, which accounts for the scattering of the energy in different directions [30].

$$h_c = \frac{\lambda}{8\cos\theta_i} \tag{3.4}$$

$$\rho_s = \exp\Big[-8\Big(\frac{\pi\sigma_h\cos\theta_i}{\lambda}\Big)^2\Big] \tag{3.5}$$

The received power P_x (W) along the LoS path from the transmitter is given by Lambert's Equation (Equation 3.6). Here, n_{TX} is the index providing information about the directionality of the emitter. A low index results in a wide emission lobe whereas a high index results in a small emission lobe. Further, d_{TX} (m) is the LoS distance from the emitter to the receiver and A_x (m^2) is the effective area of power. Moreover, ω is the angle at which the radiated intensity from the emitter is evaluated with respect to the axial angle of the emitter and γ is the same principle only then opposite, so the angle of the receiver with respect to axial angle of the receiver. Finally, P_{TX} (W) is the emitted power. [33]

$$P_x = \frac{n_{TX}+1}{2\pi}\frac{1}{d_{TX}^2}\cos^{n_{TX}}(\omega)A_x\cos(\gamma)P_{TX} \tag{3.6}$$

Practically, the problem of reflections is solved by measuring and averaging the RSSI for some interval T [18].

$$\overline{P_r} = \frac{1}{T}\int_0^T |P_r(t)|^2 dt \tag{3.7}$$

The RSSI measurement by the S2-LP transceiver on the BedSense has a similar built-in functionality. The total message that is sent by the panic button each time it transmits a signal starts with a SYNC message. The SYNC message enables the S2-LP to synchronise itself to the rate at which the data, contained in the total message, is received. By default, the RSSI is registered at the end of the SYNC, after it has been measured for the duration of the SYNC message.[21]

Shadowing
Successive to a signal's reflection on objects there is the transmission through this object that causes the effect of shadowing. It means that objects (partially) block the transmission medium for the signal. The signal shows a significantly deviating attenuation when passing these object in comparison to the larger part of its transmission medium. This effect is accounted for by adding noise to the ideal path loss model, as in Equation 3.8,

$$P(R) = P_0 - 10n\log_{10}\left(\frac{R}{R_0}\right) + \mathscr{X}_g[dBm] \tag{3.8}$$

According to [34] the measured noise $\mathscr{X}_g$ (dBm) is left-skewed distributed. However, in this project it is assumed that $\mathscr{X}_g$ is rather normally or Gaussian distributed (log-normally distributed in Watt) [13, 18, 27]:

$$P_{dBm}(R) \sim \mathscr{N}(\overline{P_{dBm}(R)}, \sigma^2) \tag{3.9}$$

Where $\overline{P_{dBm}(R)}$ is the average received power and σ^2 is the variance of $\mathscr{X}_g$, defined as:

$$\mu = \frac{1}{N}\sum_{n=1}^{N} P_n \tag{3.10}$$

$$\sigma = \frac{1}{N-1}\sum_{n=1}^{N} (P_n - \mu)^2 \tag{3.11}$$

Where P_n (dBm) is the n^{th} sample from a set of N static measurements at a static location. [27]

3.2.2. Actual Behaviour

The BedSense is equipped with the ANTX150P116B08683 PCB type antenna B. In the BedSense the antenna is mounted lying flat on its back. In Figure 3.2 the top of the antenna is visible, this side is thus facing toward the sky when build into a BedSense. According to the manufacturer, the radiation pattern has a peak offset of about -7.5 dBi. This peak offset affects signals coming in from above a BedSense.

Figure 3.2: ANTX150P116B08683 PCB type antenna, top point of view. Note. Adapted from [35]

The actual RSSI behaviour of the complete BedSense is investigated by means of taking measurements and creating polarization mismatch graphs. Figure 3.3 shows measured data of the RSSI at various distances. These measurements were conducted outside in an empty field, to gain an understanding of this relationship with minimal interference from obstacles or reflections on walls and the ground. The red line is a trendline based on the theoretical approach mentioned above. With an average deviation of 3.23 dBm, the results correspond rather well, which could make this a viable metric in the final localization algorithm. In practice, however, this metric might be challenging to use effectively due to the presence of obstacles and walls distorting this behaviour heavily and irregularly.

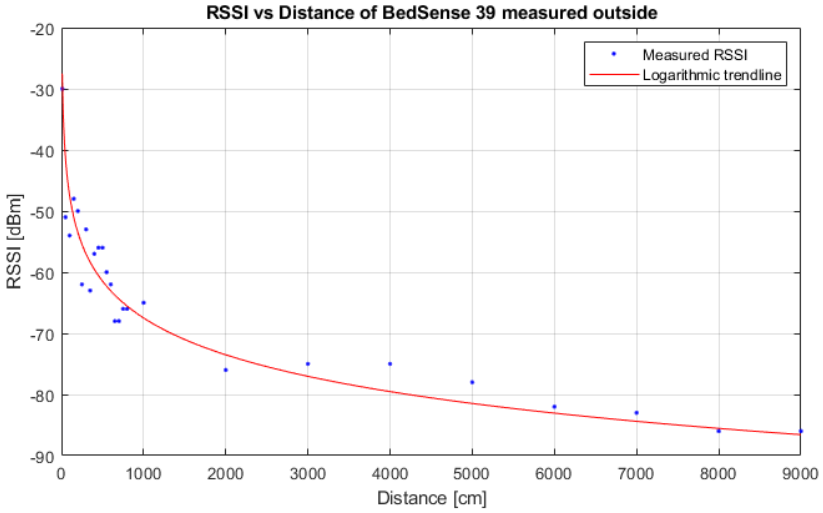
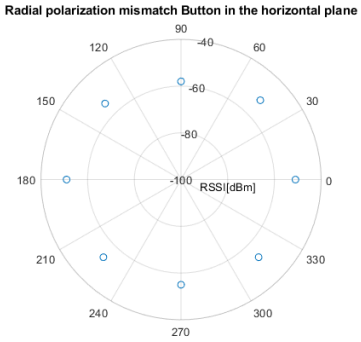
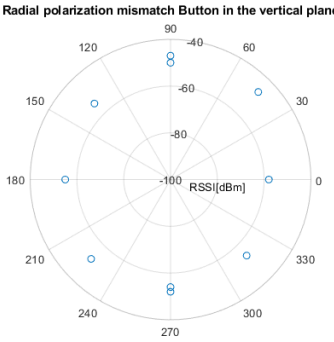


Figure 3.3: The measured RSSI at various distances.

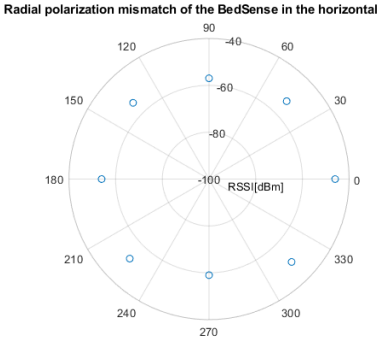
Another important metric to study is the polarization mismatch between the panic button and the BedSense. In order to do this, they were both mounted at a distance of 150 cm away from each other. Then both were rotated step-wise whilst the other remained still. The results are displayed in Figure 3.4. Both the button and the BedSense itself show very little radial variance in their respective RSSI values. Therefore, from here on, the effect of radial polarization mismatch on the received RSSI is considered to be negligible. To further strengthen this assumption, the radial sensitivity of the antenna is shown in the datasheet as being roughly invariant B.



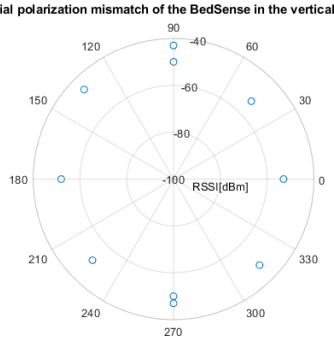
(a) The radial polarization mismatch of the button in the horizontal plane.



(b) The radial polarization mismatch of the button in the vertical plane.



(c) The radial polarization mismatch of the BedSense in the horizontal plane.



(d) The radial polarization mismatch of the BedSense in the vertical plane.

Figure 3.4: Radial polarization mismatch mapping of both the button and the BedSense

3.3. Layout of a Nursing Home

In general, within nursing homes there are two types of patients: psychogeriatric and somatic. Psychogeriatric residents (severe dementia) live mostly in closed departments. Somatic residents (physical problems or slight dementia) live in open departments. These two groups are therefore mostly separated from one another. Also, within a nursing home, the psychogeriatric departments are often smaller than the somatic departments. Some nursing homes are wide and long, while others, such as Laurens Blijdorp [36], are rather tall with a lot of floors. In the case of Laurens Blijdorp the psychogeriatric department distributed over the second until the fourth floor.

In most buildings the basic room layout is symmetric, meaning that neighbouring rooms are equal in size and structure. Not only is the layout and size of the rooms itself important, the materials that walls and floors are made of, and their thickness is important, especially concerning RSSI manners.

A schematic overview of the eight floor of Laurens Blijdorps is shown in Figure 3.5. As most of the measurements were performed at this location, this map serves as a foundation for some critical design decisions for the localization system:

- All rooms are equal in size (a regular phenomena as there is no specific reason to grand residents a different room size then others).
- The rooms are precisely located in front of, and next to each other. This allows for discretization of the map, explained in section 4.3.1.
- For the greater part there is a single hallway with rooms on either sides of it, or on only one side.
- Floors are identical in layout.

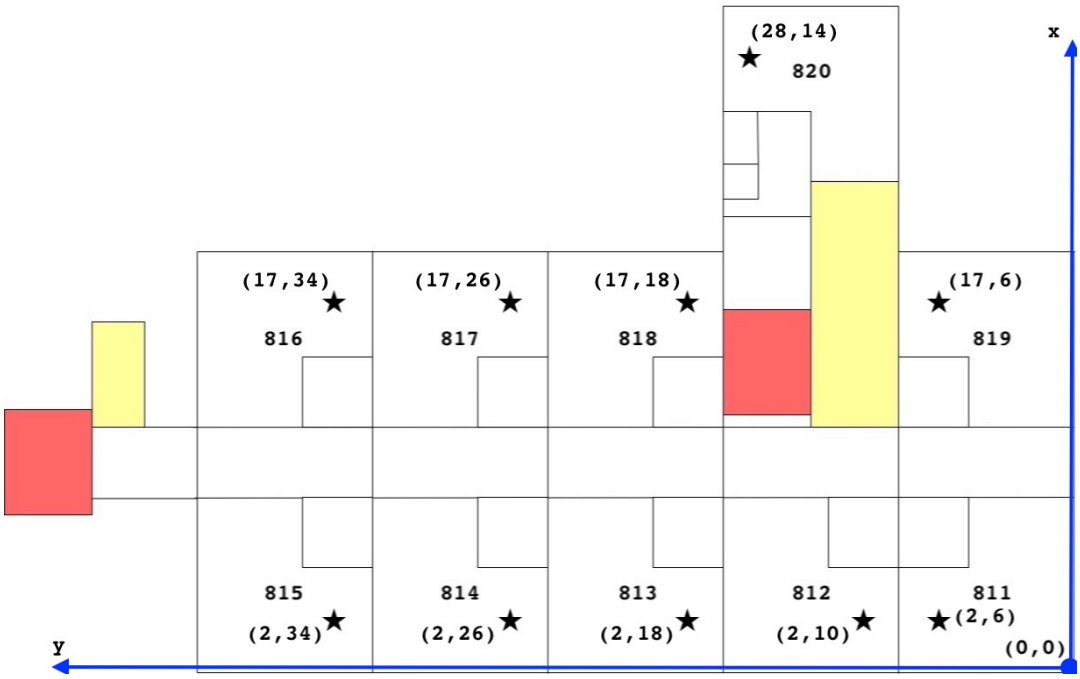


Figure 3.5: Schematic map of the eight floor of nursing home Laurens in Blijdorp. The stars indicate locations of BedSenses

3.4. RSSI in a Nursing Home

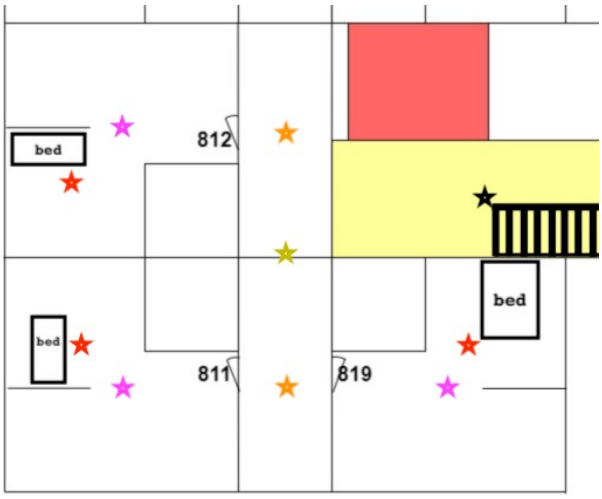
The signal propagation within nursing homes as explained above, has been measured and interpreted. This behaviour might differ from the expected behaviour by Section 3.2 as the combination of environmental factors described in that section can lead to cumulative effects.

The measurements have been taken in a nursing home: the Blijdorp location from Laurens, a nursing homes organization in the Netherlands. Laurens Blijdorp has ten floors: the ground and first floors are open areas,

the second, third, and fourth floors contain psychogeriatric departments, and the remaining floors contain somatic departments. The measurements where taken on the sixth, eighth, ninth, and tenth floor, mainly on the hallway, but also inside several rooms, both in the middle of the room and closer to the bed where the BedSense is situated. The nursing home was operational during the all measurements, meaning residents and care takers were present in the building. Furthermore, the panic button used for testing was programmed to keep transmitting "alive signals" with one second intervals. These alive signals were received by all BedSenses within reach in the nursing home; each room has one BedSense located under each mattress. Also, the panic button has a 60 cm cord attached at its top for it to be used as a necklace. For reporting, two laptops were used.

The test team existed of three people: Person one moved the panic button in a systematic and methodical manner. Persons two and three manually determined button's actual location for all timestamps and recorded this. In order to not heavily influence the transmitted signal from the button, it was attached with its cord on the one end of a one-meter long, metal pole. The other end of the pole was held by the person moving the button. As the button transmitted, the RSSI measurements by each BedSense are automatically stored in a database to which the BedSenses are connected by the internet. After the measurement session, these data were collected via Grafana, an online tool used to access and process all data contained in the database. Within the project, the measurement data represents a panic button's discrete movement through the building which is encoded in RSSI values measured by the BedSenses.

A number of the visited locations are selected to be studied more thoroughly, so more samples are taken there by holding the button steady in place for a longer duration of 20 to 30 seconds. The locations of interest, see the stars in Figure 3.6a, are: in the middle of two opposite rooms' doors (orange), in between two of these points (yellow), in the middle a room (purple), within a radius of one meter to the bed (red), and in the middle of the stairwell (black).



(a)

```
[[[ nan  nan  nan  nan 620  nan]
  [ nan  nan  nan  nan   0  nan]
  [ nan  20   14   7    0  619]
  [  0   0    0    0    0   0]
  [ nan  615  614  613  612  611]]

 [[ nan  nan  nan  nan 720  nan ]
  [ nan  nan  nan  nan   0  nan ]
  [ nan  19   8    6    0   23 ]
  [  0   0    0    0    0   0  ]
  [ nan  715  714  15   712  711 ]]

 [[ nan  nan  nan  nan 820  nan ]
  [ nan  nan  nan  nan   0  nan ]
  [ nan  10   9    4    0   26 ]
  [  0   0    0    0    0   0  ]
  [ nan  815  13   21   18  811 ]]

 [[ nan  nan  nan  nan 920  nan ]
  [ nan  nan  nan  nan   0  nan ]
  [ nan  17   1    2    0   25 ]
  [  0   0    0    0    0   0  ]
  [ nan  915  914  11   912  911 ]]

 [[ nan  nan  nan  nan 1020 nan ]
  [ nan  nan  nan  nan   0  nan ]
  [ nan  22   5    3    0  1019]
  [  0   0    0    0    0   0  ]
  [ nan  1015 16   24   1012 1011]]]
```

(b)

Figure 3.6:
3.6a: Locations of interest during the measurements.
3.6b: Output of the ratings_on_map algorithm for a single transmitted signal from room 917 (yellow 2).

Raw RSSI measurements were processed with the ratings_on_map algorithm (Appendix A.6) that maps all received RSSI's on a floor plan of the building and rates them from high to low. An example of this is shown in

Figure 3.6b. Here, The output of the ratings_on_map algorithm is shown for a single transmitted signal from within room 917, located at the yellow 2, in the form of a three dimensional array. From top to bottom the matrices represent floors six to ten. Zero entries in the matrix represent hallways, 611 to 1020 are rooms with the corresponding room numbers, and 1 to 40 are RSSI ratings ordered from strongest to weakest. They have taken the place of the room numbers that used to be present there. For each measurement, such a map has been generated. Several results have been extracted from these maps:

- It is clear that not all BedSenses received the signals. Some seem to have been offline.
- BedSenses that did receive the signal often significantly differ from each other in RSS.
- There seems to be a trend in the ratings of rooms (with a BedSense) around and neighbouring the room of the actual location.
- Azimuth orientation of the person carrying the button significantly influences the RSSI measured by BedSenses located behind the person.
- When standing static at a location for an interval of time, each BedSense's RSSI measurements fluctuate significantly during this interval, up to 10 dBm with a roughly calculated variance of about 5 dBm.

While the difference in received RSS is largely the effect of environmental factors, it is also due to differences or offsets between the devices themselves: an effect that was confirmed by looking at measurements taken close to the bed, where environmental factors are minimized, for several rooms and comparing those. The trend in ratings of neighbours of a room X has manually been interpreted, see Table 3.1. To elaborate, take X as room 814 in Figure 3.5, Table 3.2 shows what neighbours of 814 belong to what group. The BedSenses in the remaining rooms are often out of the signal's range, show only low RSSI measurements, and are influenced too much by noise to be properly rated in this manner.

Table 3.1: General rating of neighbours by RSSI

Group	Type
0	Present room (X)
1	Direct neighbours of X sideways (D), Neighbours directly above and below (B)
2	Direct neighbours of B sideways (BD), Neighbours two floors directly above and below (BB)
3	Second direct neighbours of X and direct neighbours of D sideways (S), Neighbours of X sideways with hallway in between (I), Neighbours of B sideways with hallway in between (BI)
4	Neighbours of B sideways and diagonally with hallway in between (direct neighbours of BI) (BY)

Table 3.2: Example of the observed trend in Table 3.1 applied to room 814

Type	Rooms
X	814
D	813, 815
B	714, 914
BD	713, 715, 913, 915
BB	614, 1014
S	812
I	816, 817, 818
BI	717, 917
BY	716, 718, 916, 918

3.5. Evaluation of RTLS Improvement Techniques

Section 1.4.2 proposes two widely used techniques that can be used to improve localization: fingerprinting, and the Kalman filter. This section gives an evaluation of the extent to which they can be used.

3.5.1. Fingerprinting

A solution for the described, main problems using the fingerprinting method is implementing a trainable algorithm using machine learning. However, training, as described in section 2, is out of this project's scope and undesirable for the eventual application, since installation of the system should be as non invasive as possible. On top of that, creating an accurate floor plan and fingerprint of the building is incredibly time consuming and labour intensive.

From the application point of view, nursing homes are not suited to create a clean, accurate radio map as these places are highly dynamic. On one hand, they are influenced by residents, care takers and other persons that move around full time. On the other hand, receivers, BedSenses, are located under the mattress of each bed. This results in short term effects from people getting in and out of bed, people moving around the bed, and nursing beds being adjustable in position. It also results in long term effects due to alterations in the room layout and changes in location of the bed itself.

Though the method of fingerprinting is not considered an option during this project, a similar, but smaller scale solution is implemented that also uses a radio map, in Section 4.4.1 called the neighbour model. This solution avoids machine learning and its radio map is created differently and more easily.

3.5.2. Kalman Filter

The Kalman filter estimates a variable for the current time k - here that variable represents the location - by using measurements of that time instance k and a predicted value for the variable based on its previous values. This technique requires insight into various parameters and evaluations of the situation at hand. For example, prediction of the state requires insight into the speed of an elderly person when he or she is moving: walking through a hallway, passing by one room after the other, the elderly person should for an interval of time be localized at the nearest room before the system predicts the location being the successive room. This has everything to do with the elderly person's speed.

The greatest value of using the Kalman filter comes from looking at previous states of the panic button's location. However, because of its accuracy, the main system that is constructed in this project as described in Section 4.3, will not make much use of a Kalman filter. Localization by the system is accurate up to one room. Even though that room could as big as 20 by 20 meters, a person walking in or around that room will each time be localized at that same location, namely, inside that room. However, in later stages of the system, where the localization should also work for hallways, the Kalman filter could add more value as the accuracy of the overall system increases.

4

Prototype and Validation

This chapter describes the prototype of the localization system and the choices made during programming. First the data pre-processing stage is described in Section 4.1. Secondly, the multilateration algorithm is described in Section 4.2. Furthermore, in Section 4.3 the main localization algorithm is explained. Moreover, the two reliability checks are described in 4.4. Finally, the results of the localization algorithm are discussed in Section 4.5.

4.1. Data Pre-processing

The pre-processing of data is a vital part of any program working with any data. When data is obtained, it is seldom exactly in a shape or form which can immediately be used. Pre-processing data is done for many different reasons, such as for removing imperfect or applying dimensionality reduction, etc. [37]. In the case of the data used for localization, pre-processing is mostly done because of imperfect data, meaning it has a lot of missing values and noise that need to be reduced. The data consists of two datasets, the RSSI data received by the BedSenses and the data received by the Access Control System (ACS) (whether a panic button is seen by a sensor placed at a door or not). The python code used for pre-processing the data can be found in Appendix A.2.2.

4.1.1. BedSense Data

The data received by the BedSenses contains at what time, which BedSense has received a signal from a panic button. A panic button sends a signal to all BedSenses close enough to receive it. This signal is sent every minute (an alive signal to indicate that the battery is not dead) and whenever the panic button is pressed. However, the BedSenses do not all receive this signal at the same time, which results in a lot of lines of data where only a few BedSenses receive a signal. This means that each dataset contains a lot of missing values. Therefore, the data can be grouped per minute. This however, introduces the problem that sometimes multiple signals are received by the same BedSense within the same minute and sometimes there is a minute in which the same BedSense does not receive anything from a certain panic button.

Chosen is to group the data for every 60 seconds. This seems similar to grouping per minute, however grouping per minute means from , for example, 12:00:00 till 12:01:00. With grouping per 60 seconds is meant for example grouping the data from 12:00:25 till 12:01:25. For the grouping to happen, at least 60 seconds should have passed since the first received signal. In this way all the BedSenses close enough to the panic button will have received the signal within those 60 seconds.

As for the problem of receiving multiple signals within the same 60 seconds, the maximum value is taken from all the received signals by one BedSense. The reason for doing this is mainly because of reflections. As is described in Section 3.2.1, power is lost due to reflections. Therefore, the maximum value received by a BedSense will be the most reliable as this received signal will most likely have the least amount of reflections. Another reason for taking the maximum value is the automatic gain control (AGC) of the receiver on the BedSense. The AGC can decrease the RSSI but cannot increase it, which further emphasises that a high RSSI is more reliable than a low RSSI.

Sometimes it is also possible for some of the BedSenses to receive a very low RSSI (lower than -90 dBm). It is assumed that every signal below -90 dBm is influenced by so much noise that it is no longer reliable. Therefore, all received signals of which the RSSI value is lower than -90 dBm are not taken into account during localization.

4.1.2. Access Control System Data

The ACS outputs data in a similar way to the data coming from the BedSenses [2]. In other words, the data coming from ACS states at what time, which door has detected a specific panic button. Only the range of detection of the ACS is significantly shorter that that of a BedSense. Therefore, the data is also grouped in a similar way. It is not useful, for every time the panic button is detected by an ACS, to give a location as this can result in giving a location multiple times within a second. Therefore, as the panic button should be localized once every minute, this data is also grouped by 60 seconds, for the same reasons described in Section 4.1.1.

If a panic button is detected multiple times by multiple doors, this needs to be filtered out. As the data is grouped by 60 seconds, it is no longer useful to look at data from the beginning of the time frame. If the button is detected in the beginning of the time frame on a certain floor, but at the end of the time frame the same button is detected at another floor, the first detection is no longer useful. Therefore, only the last time a panic button is detected near a door within the time frame is used for localization.

4.2. Multilateration

A transmitter can be localized as a point in space represented by its spatial coordinates. In trilateration, a series of receivers with known locations and measured distances from the transmitter can be used to calculate the location of the transmitter relative to these receivers. Multilateration is the extension of this system, which is noting more then systematically combining two or more trilaterations from four or more receivers. For this project, the transmitter is a panic button and the receivers are the BedSenses. Trilateration is performed with three receivers and one transmitter. Around each of these receivers, a sphere is created with the distance between that receiver and the transmitter as its radius. If the three distances are error free, they all intersect at exactly one point: the location of the transmitter. However, due to fluctuations and effects on the signal propagation, the measured distances are only approximations of the real distances. That introduces another two possibilities for the three spheres: they have no common point of intersection, or they have two points of intersection, see Figure 4.1. In three dimensional space this requires a fourth receiver to reduce the problem to a one intersection.

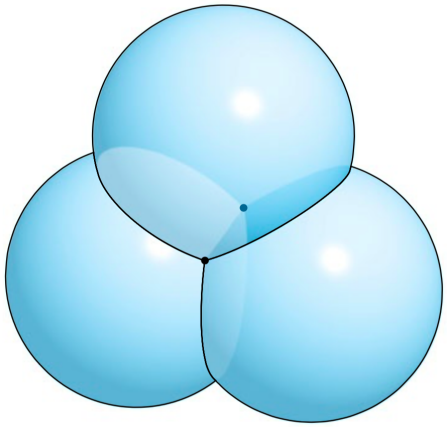
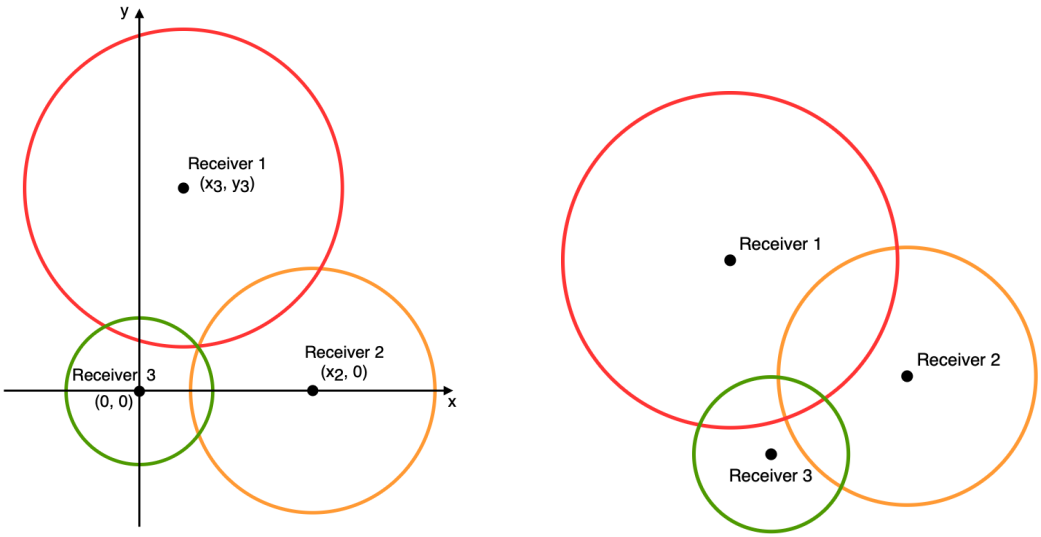


Figure 4.1: Three spheres with two points of common intersecting. Note. Adapted from [38]

In the first version of the multilateration algorithm only a single floor is considered. It is assumed that all receivers and the transmitter are located at the same height, meaning the z-component is abandoned. The problem reduces to two dimensional space, and the spheres around the receivers become circles instead of spheres; their common point of intersection gives a two dimensional location. The three types of common intersection are now as follows: ideally there is a single point point of common intersection, but in reality

there will often either be no such point or the intersection covers an area, see Figure 4.2,



(a) Ideal situation (b) Reality

Figure 4.2: Intersection of three circles

Each of these circles is defined by the general equation for a circle, Equation 4.1,

$$r^2 = (x - x_c)^2 + (y - y_c)^2 \tag{4.1}$$

With (x_c, y_c) the centre of the circle. The location of the transmitter is analytically found by formulating the three circles and solving for (x, y). Using [39] the problem is simplified by setting the centre of one circle at the origin and the centre of another circle on the x-axis, as in Figure 4.2a. This gives the following formulas for the three circles:

$$r_1^2 = x^2 + y^2 \tag{4.2}$$

$$r_2^2 = (x - x_2)^2 + y^2 \tag{4.3}$$

$$r_3^2 = (x - x_3)^2 + (y - y_3)^2 \tag{4.4}$$

Where r_i is the distance between receiver i and the transmitter located at (x_t, y_t). Combining Equation 4.2 until 4.4 and solving for x and y in case of a single intersection gives,

$$x_t = \frac{r_1^2 - r_2^2 + x_2^2}{2x_2} \tag{4.5}$$

$$y_t = \frac{r_1^2 - r_3^2 + x_3^2 + y_3^2 - (2x_3 x)}{2y_3} \tag{4.6}$$

In reality, where often no single point of intersection exists, [13] proposes to take the minimum value of an error function as the location of the transmitter.

A simple algorithm to perform multilateration on the incoming BedSense RSSI data has been made and tested (Appendix A.7). This algorithm uses the path loss model of Equation 3.2 to translate the RSSI into distance used as radius for each circle. Multiple collected data sets were given as an input to the algorithm, the result of one of them is discussed in more detail.

A particular measurement was taken at the eight floor of Laurens Blijdorp. A map of this floor including dimensions and relative locations of BedSenses is shown in Figure 3.5. For this measurement, the algorithm predicts the locations of the panic button as if it draws random values from a range of locations that lie within a radius of 500 meters around the building, meaning it does not give any valuable results at all. The reason

for this erroneous localization is that the RSSI values are highly effected by reflection- and shadowing effects, as described in Section 3.2, which makes them fluctuate a lot.

To increase the accuracy of the model several improvements are suggested and some of those also tested. Firstly, in an ideal situation, receivers located far from a transmitter measure a lower RSSI value then receivers located closer to the transmitter. This follows from the path loss model, which also shows that the decrease in RSSI close to the transmitter is steeper then the decrease at larger distance, see Figure 3.3. Thus, at larger distances the normal distributed fluctuation error that is contained in the RSSI creates a stronger error in calculated distance. Ignoring measured RSSI values below a certain threshold then lowers the effect of high distance fluctuations. To test, this threshold has been set to $-75dB$; a compromise between ignoring too many measurements and including too many erroneous values. This has significantly improved the results. As expected, the locations vary within a smaller radius of 80 meters around the building. In addition to that the localization has became more stable as the determined locations lie relatively closer to each other. This later conclusion defines that the distance related error is indeed higher at larger distances as ignoring those larger distances leads to relatively more stable results.

Some other possible improvements. Calibrating the BedSenses to get more equal measurements of RSSI as there is an offset between BedSenses in the measured RSSI due to device differences. Taking into consideration the effects of reflections and shadowing in translating the RSSI into distance. Lastly, applying the Kalman filter on the RSSI values or locations to get a more stable localization. However, as described by Section 2.2.2 reporting happens through textual format telling the end-user in which room the person is. The accuracy is therefor a maximum of one room. Furthermore, the RSSIs from the BedSenses show a reoccurring trend, to be explained in Section 4.3. By the combination of those two it was chosen to approach localization in a different manner.

4.3. Nearest BedSense Algorithm

An easy way to determine an approximate location of the panic button, is to simply evaluate all the RSSI values received by all different BedSenses within the given time frame and take the largest RSSI value recorded. Then it is reasonable to assume that the panic button is either in or close to the room containing this BedSense. This is close to what the Nearest BedSense Algorithm (NBSA) does. The localization of the panic button by the NBSA is performed in two parts, the prediction of the floor and the prediction of the room in a two dimensional space. Both of these are done separately as it is more accurate to determine the room than it is to determine the correct floor just by looking at the RSSI. Therefore, first the floor is determined and after the room is determined. The python code used for localization can be found in Appendix A.1.

4.3.1. Map Implementation

To be able to perform these checks, a simplification of the map of a nursing home is required which can be transformed into a simple array. This is useful, because in Python, working with arrays becomes fairly straightforward. In Figure 4.3 a simplified version, a .xlsx (spreadsheet) file, of the map in Figure 3.5 is shown. In this simplification each room is simply reduced to one square in a grid and indicated by its room number. Squares containing zeroes represent the hallways in between rooms. Squares containing staircases will be represented by a -0.5 and squares containing lifts will be filled with a -1.5. These values were taken negative to make the easily differentiable from room numbers, which are taken to never be negative. They are not integer numbers, because those are used by the History Check, described in Subsection 4.4.2, to indicate distance values. This schematic simplification can then simply be translated into an array. This is done in map_matrix (Appendix A.3.1) by creating an array with the same dimensions as this schematic map and placing the values within the squares into the corresponding floor plan array entries. Any empty entries in Figure 4.3 will be filled up with a NaN (not a number) value in the floor plan array.

					820	
				-1.5	0	-0.5
		816	817	818	0	819
-1.5	0	0	0	0	0	0
		815	814	813	812	811

Figure 4.3: Simplification of the map from Figure 3.5.

For each floor of a nursing home, such an array is created. These arrays are then stacked on top of each other to create a three dimensional array, creating a simplified three dimensional blueprint of the entire nursing home. Since every BedSense is coupled to a room number, this blueprint can be used to make predictions about the interrelationships of various measured data points. These predictions are subsequently used in the check functions described below in order to improve the error-rate of the overall system.

4.3.2. Prediction of the Floor

To determine the floor on which the panic button is, the data coming from the ACS is analysed. If the panic button is detected by an ACS, it is confirmed that the panic button is at the same floor as the ACS that has detected the panic button. This can be assumed as the ACS can only detect the panic button if it is within a radius of one meter to the ACS's antenna. As the ACS is used for the determination of the location, the floor can only be determined when the panic button passed an ACS. If this is not the case, the determined location of the panic button stays stuck on the floor where the ACS has last seen the panic button. Therefore, it is required that at every stairwell and every elevator an ACS is placed.

As a security measure if the ACS fails, the floor can also be predicted by taking the mean of the floors on which the five BedSenses are located which have received the strongest signal. This is done because the floors of the rooms in which the BedSenses receive the strongest signal are assumed to even out to the floor on which the panic button is. This can be said as a nursing home can have many floors or just a couple and in both cases taking the mean of the floors of the five strongest receivers will result in the correctly predicted floor. If the panic button is on one of the middle floors of a nursing home with many floors, both floors above and below will receive a strong signal and the mean would result in the actual floor. However, this concept also works if the button is on the top floor of a nursing home, as the actual floor will always receive some of the stronger signals, which average out so that the actual floor equals the determined floor.

4.3.3. Prediction of the Room

To predict the room in which the panic button is, first, the five rooms in which the BedSenses receive the strongest signal are determined. Then the three dimensional location of these five rooms is determined using the map explained in Section 4.3.1. Using these locations and the predicted floor, the rooms of the strongest receivers are projected onto the two dimensional map of the predicted floor. This means that the same rooms are picked as the strongest receivers only sometimes on a different floor (the predicted floor), see Figures 4.4a and 4.4b. This is done because it was found during testing, see Section 3.4, that in multiple nursing homes the rooms directly above and below the actual room receive the signal stronger than the neighbours sideways of the room.

After having selected the most likely room locations on the predicted floor, the next step is to predict a specific room. This is done with a number of reliability checks described in Section 4.4. The first check performed is the history check (Section 4.4.2), which determines the probability of moving from the previously determined location to the currently predicted location. If the outcome of the history check states that is is likely (> 50%) for the person to have moved to one of the possible rooms, the rooms meeting this criteria are all saved. For the rooms that are saved, the second reliability check is performed, the neighbour comparison (Section 4.4.1). The neighbour comparison compares the RSSI values of the neighbours of the predicted room to each

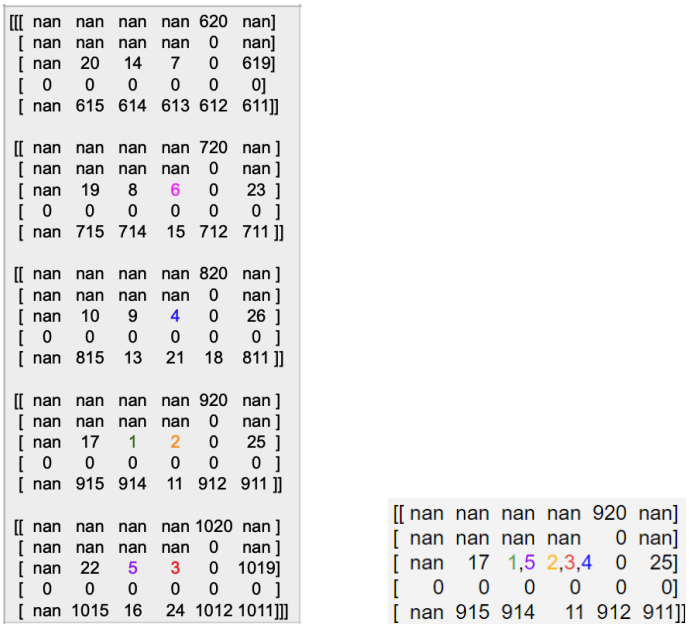


Figure 4.4: 4.4a: The ranking of the RSSI (where 1 is the highest and 40 the lowest, room numbers are omitted) from a single transmission. 0 represents the hallway and 611 up to 1020 are the room numbers.4.4b: The five strongest receivers mirrored to the predicted floor.

other to determine if they are as expected. Using these comparisons, the room that has the highest neighbour comparison result will be the determined room.

Furthermore, there are some contingencies in place to prevent the algorithm from failing for example when none of the most likely locations score higher than 50% on the history check. If this is the case, a multiplication will be performed between the outcomes of the history check and the neighbour check, so that a negative history check can balance with a positive neighbour check and therefore give a more accurate prediction. From the results of the multiplication, the highest value is taken and the room belonging to that measurement becomes the predicted room.

At last, the location of the panic button is outputted in a statement. This statement consists of the determined room and the time at which this determination was made. Chosen is to have the statement look like: "The panic button is closest to room [room number], at time [date and time]. This is done because to any care giver working in a nursing home, the layout is known. However, if it would be unclear on which floor the rooms are located, the statement can easily be adjusted to contain the room as well.

To clarify an overview of the total system is shown in Figure 4.5. Here, it can be seen how the NBSA fetches the required data, processes that data, finds the floor and room, and uses the reliability checks to estimate the location of the panic button, and report that back to the database.

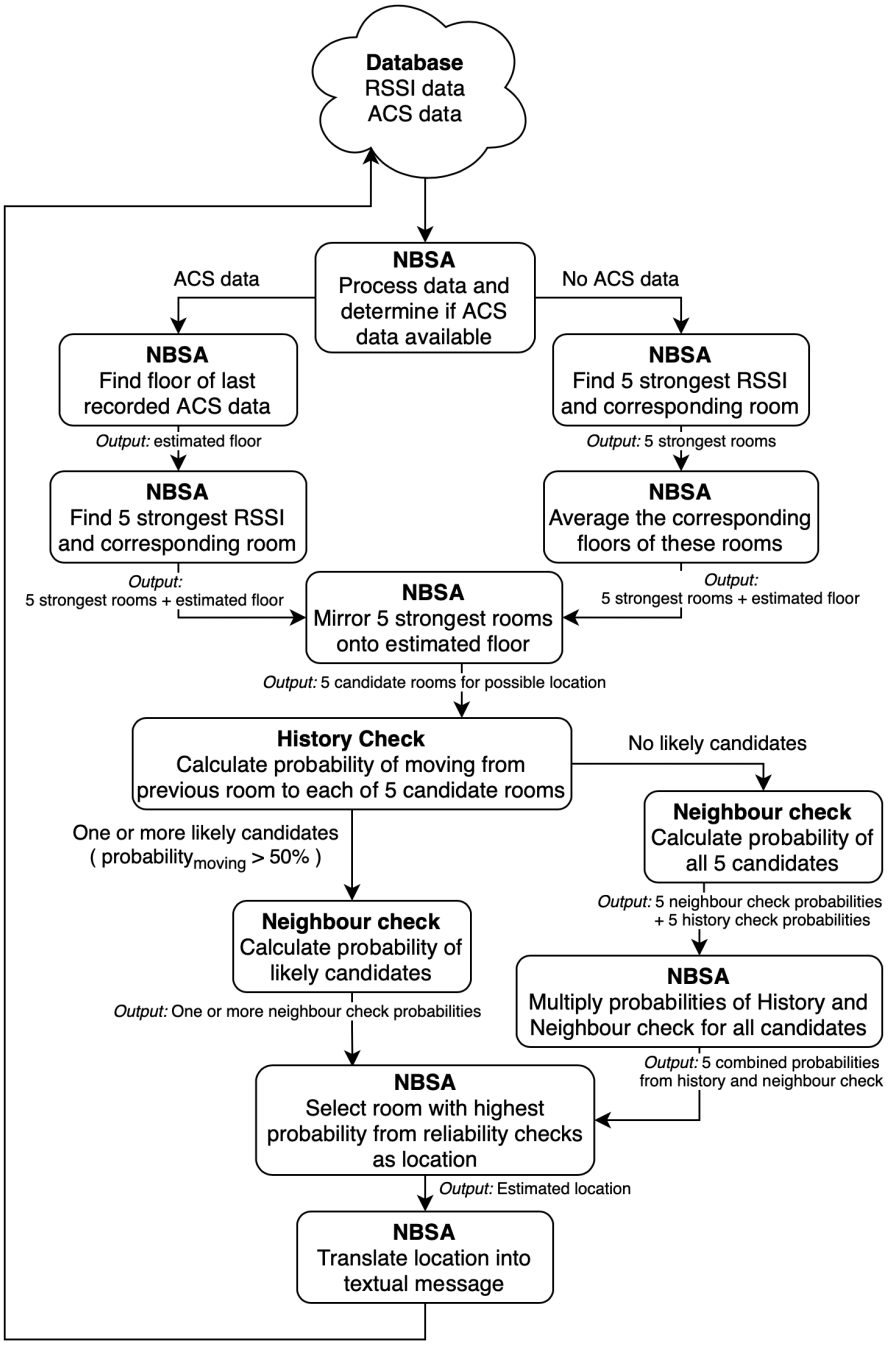


Figure 4.5: Overview of the working principle of the total localization system

4.4. Reliability Checks

In this section, the reliability checks mentioned in Section 4.3 are explained in more detail. This also includes various assumptions and validations performed to ensure a smooth and proper functioning of these checks in tandem with the main algorithm.

4.4.1. Neighbour Comparison

The trend of certain neighbour groups often receiving the transmitted signal different than others, as described in Section 3.4, can be used to check the likelihood of being at a certain location, of all rooms selected by the NBSA (Section 4.3). The neighbour comparison (Appendix A.4) checks a model of neighbours on the

simple condition that members of one neighbour group are expected to receive the signal weaker or stronger than the other neighbour group.

Neighbour model
To this manner the neighbour model has been created and extended several times based on better insight on this trend between neighbour groups, see Figure 4.6. Here, around the room for which the map is made, room X, all possible neighbours, of the types introduced in Table 3.1, have been mapped. Orientation is preserved in this setup, that is, the nursing home map, in the format of an array, explained in Section 4.3.1, shares its orientation with each room's neighbour model. Thus, looking at room 814 as in Section 3.4, the rooms 816, 817, and 818 are on the top of 814 in the map of Figure 3.5. In the neighbour model of room 814 that orientation is preserved by placing room 816 until 818 on the top of room X (room 814). For convenience, in the algorithm, these five matrices in Figure 4.6 are translated to a single, 70 elements long array according to five layout matrices in Figure 4.7. In this translation the number of a neighbour in the layout matrices corresponds to the element's index in the array. Index 69 - because in Python coding indexes run from 0 to 69 for a 70 long array - represents the floor number.

From the floor plan array it extracts from a building's map, the map_matrix algorithm (Appendix A.3.1) creates for each room a neighbour model array. During Real-Time Localizing (RTL) the NBSA, Section 4.3, substitutes each RSSI for the neighbour's room number in the floor plan array of a considered room X.

```
|----|-----------------|---------------|-----------------|----|
|    |                 |     i i i     |                 |    |
|    |    by bi by     |       s       |    ay ai ay     |    |
|    | by bd bd bd by | i   d d d   i | ay ad ad ad ay |    |
| bb | bi bd b  bd bi | i s d X d s i | ai ad a  ad ai | aa |
|    | by bd bd bd by | i   d d d   i | ay ad ad ad ay |    |
|    |    by bi by     |       s       |    ay ai ay     |    |
|    |                 |     i i i     |                 |    |
|----|-----------------|---------------|-----------------|----|
```

Figure 4.6: Neighbour grouping per floor. The blocks represent floors, from left to right: two floors above room X, one floor above X, floor of X, floor below X, and two floors below X.

```
|---|----------------|----------------------|----------------|----|
|   |                |       46 35 36       |                |    |
|   |    21 10 11    |          31          |    67 56 57    |    |
|   | 20 9  2  3  12 | 45    30 23 24    37 | 66 55 47 49 58 |    |
| 0 | 19 8  1  4  13 | 44 34 29 22 25 32 38 | 65 54 47 50 59 | 68 |
|   | 18 7  6  5  14 | 43    28 27 26    39 | 64 53 52 51 60 |    |
|   |    17 16 15    |          33          |    63 62 61    |    |
|   |                |       42 41 40       |                |    |
|---|----------------|----------------------|----------------|----|
```

Figure 4.7: Layout of neighbours and their mapping to the neighbour array.

Neighbour model check
By deployment of a neighbour model after the RSSI's are substituted by the NBSA, the neighbour comparison algorithm checks all the to-be-compared neighbour pairs. As explained, there is an expectation about whether a neighbour receives the signal stronger, about as strong, or weaker than certain other neighbours. Specifically, to-be-compared neighbour pairs should meet two requirements: firstly, their location separation

has to be limited, and secondly, the expectations of RSSI's - the groups in Table 3.1 to which they belong - differs enough. The maximum separation follows from the phenomenon that BedSenses in the rooms behind the person measure a lower RSSI then rooms to its front. Neighbours on each side of a person (front and back) measure RSSI with differences up to 19 dBm on average. Therefore, it is chosen to compare neighbours, of the neighbour model in Figures 4.6 and 4.7, only to neighbours lying on a maximum separation of 90 degrees around room X. Vertically this is implemented by not comparing rooms from higher floors than room X to rooms from lower floors than room X. For example, room 2 should be compared to rooms 1, 3, 4, 8, 9, 19-21, 11-13, 0, etc., but, by the second requirement, as rooms 3, 4, 8, 9, and 0 are not different (enough) in expected RSS, these are not compared to room 2 after all. This way the differences due to the effect of the person's body between rooms lying further separated then 90 degrees do cause wrongly false comparisons. Within this 90 degrees separation these differences tend to be less significant and disappear in the RSSI fluctuation effects.

With respect to these two requirements, all to-be-compared neighbour pairs are linked in a square table in which a 0 indicates that the corresponding neighbour pair is to-be-compared. This table is shown in Appendix A.4.4. Two separate functions, comparison_dictionary and comparison_weights (Appendix A.4.2 and A.4.3) arrange a lookup table (named "dictionary" in Python) according to this table. This dictionary defines for each neighbour in the neighbour model, what are the to-be-compared neighbours supplemented with appropriate weights for the comparisons, explained further in Section 4.4.1.

If an RSSI check of two to-be-compared neighbours is as expected, the appropriate weight is summed to a general variable that keeps track of the check's score. If the RSSI check is different than expected, the appropriate weight is subtracted from the general variable. Finally, the score is normalized by division of the general variable by the maximum achievable score for this neighbour model's neighbour comparison. The normalized score is a number between -1 and 1. A score of -1 means the RSSI-containing neighbour model is totally against the expectation, and a score of 1 means it is totally in agreement with the expectation.

Neighbour model check weights

Each to-be-compared neighbour pair has two neighbours from a different group in Table 3.1. These groups have been formed by interpretation of a trend, and can physically be explained by looking at the transmission medium. The main influence on the signal propagation that leads to this grouping, is the traveled distance and the effect of reflections and shadowing, as explained in Section 3.2. Thus, if a comparison is performed there is only limited certainty that the comparison, whether it is as expected or not, is actually valid when possible errors in the RSSI are taken into account.

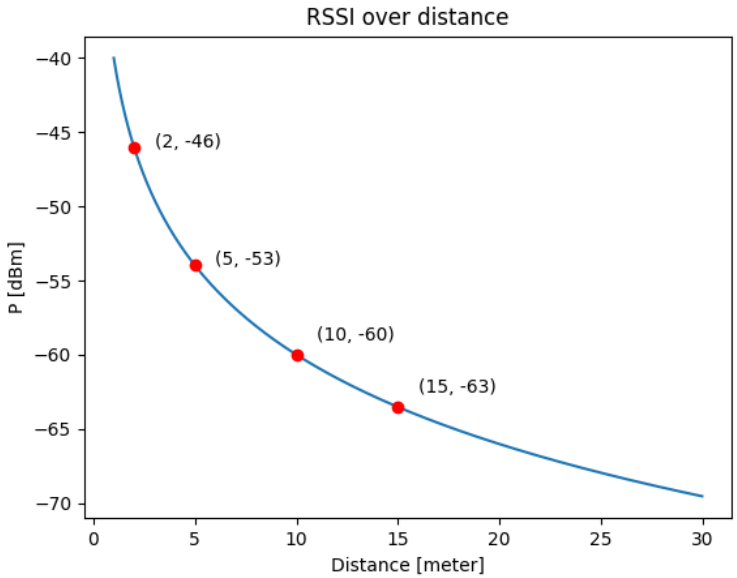
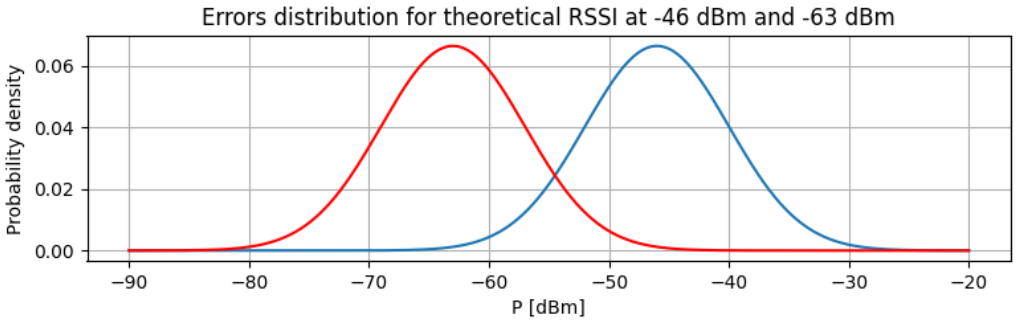


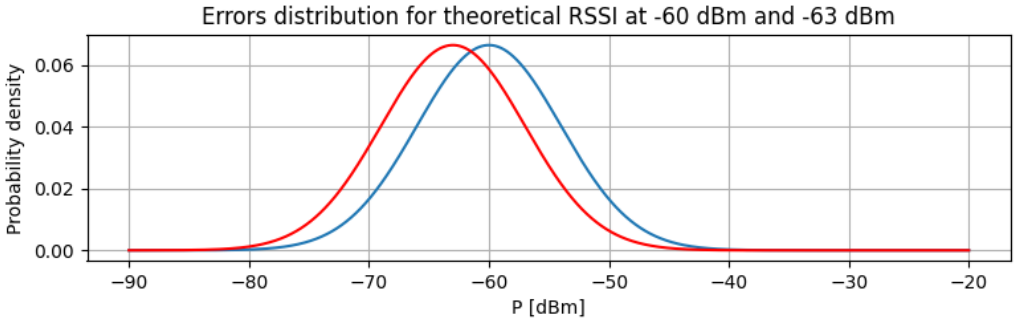
Figure 4.8: Theoretical distance-power curve of the RSSI with several highlighted points representing fictional neighbours of room X

By Equation 3.8, the further the receiver is removed from the transmitter the weaker the signal becomes, see Figure 4.8. Due to possible fluctuation, which are assumed to be Gaussian distributed, Section 3.2.1, the RSSI could at each distance run lower or higher. Visually this would look like plotting a normal distribution vertically on top of any point in the graph of Figure 4.8. The mean of this normal being the theoretical power for that distance, and the standard deviation is found by measurements for that distance. On the other hand, this means that for a certain measured RSSI there is an error in the corresponding distance that is a function of this fluctuation.

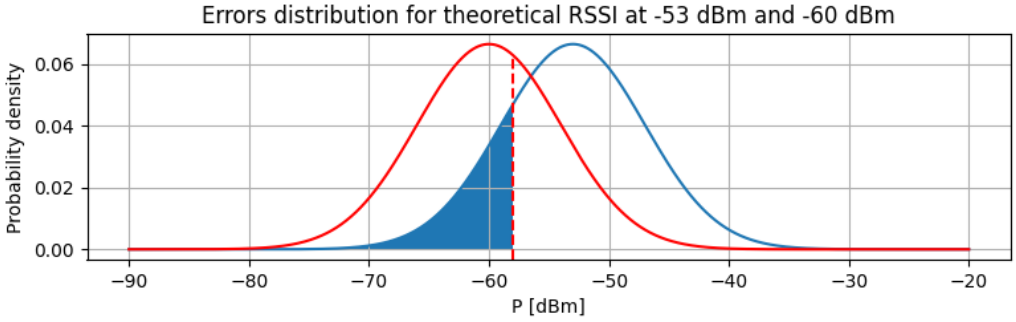
Comparison is done between two neighbours, at different distances, that both received the signal and both contain this fluctuation error. If the distance from the transmitter to both neighbours becomes larger, the two normal distributed probability curves move closer to each other. To visualise this, for three combinations of the fictional neighbours in Figure 4.8, the RSSI distribution is shown in Figure 4.9. Each distribution is centered at the theoretical RSSI for that neighbour's distance. The variance is taken as 6. This value was calculated from measurements inside an office for a system operating at 900 mHz by [24]. Two effects are visible: Firstly, the error probability curves of neighbours located closer to each other overlap more, compare Figures 4.9a and 4.9b. Secondly, from two neighbour pairs that have their neighbours separated equally far, the pair that is located the furthest from the transmitter has its error probability curves overlapping the most, compare Figures 4.9c and 4.9b.



(a) Overlapping normal distributions for neighbours at 2 and 15 meter



(b) Overlapping normal distributions for neighbours at 10 and 15 meter



(c) Overlapping normal distributions for neighbours at 5 and 10 meter

Figure 4.9: Three examples of two overlapping normal distributions

As said, if a comparison is performed there is only a certain probability p that it is actually correct. For convenience the neighbour located further away from the receiver is called the "far neighbour", and the neighbour located closer to the receiver the "close neighbour". The conditional probability of the close neighbour receiving a lower RSSI, $RSSI_{close}$, than the far neighbour, given that the far neighbour receives a certain RSSI, $RSSI_{far}$, is,

$$P(RSSI_{close} < RSSI_{far} \mid RSSI_{far}) = p_{far}(x = RSSI_{far}) \cdot p_{close}(x < RSSI_{far}) \tag{4.7}$$

Where $p_{far}(x = RSSI_{far})$ is the probability density of the far neighbour (in red) receiving $RSSI_{far}$ (indicated by the dashed line), and $p_{close}(x < RSSI_{far})$ is probability of the close neighbour (in blue) receiving any RSSI smaller than $RSSI_{far}$ (indicated by the blue area), in Figure 4.9c.

The probability of a comparison being incorrect is equal to the sum of Equation 4.7 for all possible $RSSI_{far}$,

$$\begin{aligned} P(RSSI_{close} < RSSI_{far}) &= \sum P(RSSI_{close} < RSSI_{far} \mid RSSI_{far}) \\ &= \sum P_{far}(x = RSSI_{far}) \cdot P_{close}(x < RSSI_{far}) \\ &= \int_{-\infty}^{\infty} f_{far}(x) \cdot F_{close}(x)\ dx \end{aligned} \tag{4.8}$$

The probability of the comparison being incorrect is related to it being correct as,

$$p(correct) = 1 - p(incorrect) \tag{4.9}$$

For each neighbour group in Table 3.1 a predefined distance is given as an input for the comparison_weights algorithm. These distances are known for, or measured in, the nursing home where the RTLS is installed. The comparison_dictionary algorithm uses the comparison_weights algorithm for each neighbour pair it adds to the lookup table (the dictionary) the acquire the appropriate comparison weight. This dictionary then is used by neighbour_comparison to check what are the to-be-compared neighbours and what weights to use for each of these comparisons in assigning a general score of a room's neighbour model.

Testing the algorithm

For the purpose of discussing test results, a summary of the actual purpose of the neighbour comparison algorithm: This algorithm checks for a room X to what extend the trend in RSSIs from neighbouring BedSenses is present expressed in a score between -1 and 1. Room X is selected by the NBSA, and, in fact, that algorithm selects multiple candidates for room X. Thus, NBSA uses the neighbour comparison algorithm to "choose" between its selected candidates; according to the neighbour comparison algorithm the candidates with the highest score has the largest probability of being the estimated location.

Two different tests have been performed, a theoretical test and a test with real data pre-processed by the NBSA. The first test serves to confirm the working of the algorithm. Being mainly a debugging tool, this test also gives insight into ratings that are given to different sort of setups. For example, if many neighbours far away from room X show the expected behaviour and only a few close neighbours show unexpected behaviour, the score is heavily lowered by these closer rooms. Furthermore, if only neighbours that are not to-be-compared to each other are set to have reported an RSSI the comparison returns score "0". This all is, of course, the intended behaviour of the algorithm, showing it theoretically works.

In the second test, the NBSA selects, for every received ping by the panic button, two candidates. In particular these two rooms, as they reported the highest and the one but highest BedSense's RSSI. The following parameters are used to check the performance of the algorithm when working such real input data: Actual location: Number of times the correct location is selected, when the correct location is one of the two selected rooms; closest location: Number of times it selected the closest room, of the two selected rooms, to the correct location; and total checks: Total number of checks performed. Hallways are not differentiated in the algorithm, meaning that if the button is in the hallway, in front of two rooms, either of those rooms is considered a correct location estimation. The results are shown in Table 4.1.

The "actual location" and the "closest location" parameters are considered equally strong in the validation of the algorithm. This and the results in Table 4.1 lead to the overall accuracy of the algorithm for this data

Table 4.1: Neighbour comparison test results

Parameter	Value
Actual location	106
Closest location	84
Total checks	281

to be 67.6%. In 37.8% of the estimations the correct room was estimated, and in 29.9% of the estimations the closest of the two select rooms was estimated. In the other cases the other of the two selected rooms was estimated. Often this turned out to be the room located above or below the actual room. Relying on other checks performed or used by the NBSA, this problem should be solved as only rooms from a one floor are then selected as candidates. This solution is tested and further discussed in Section 4.5.

Another, possible improvement could be to collect measurements of multiple pings and average these together. Taking the highest two RSSIs from this set eliminates possible fluctuations in RSSI. These fluctuation became visible when looking at the rating on map measurements in Section 3.4.

4.4.2. History Check

The second reliability check that is implemented looks at previous localization points and compares those to the current predicted location. Firstly, the shortest distance between the current measured point and the previous registered point is taken. This is then compared to the maximum distance that can theoretically be traveled by a resident of a nursing home within this time frame. The main function of this check is to enable the localization algorithm to more easily identify measurements where the strongest received signal comes from a place on a different floor from the actual location of the panic button. This is relevant because when this is the case, the resident would often have had to move at speeds which are practically impossible, especially since residents are often of elderly age. The History Check can filter away these erroneous measurements and improve the reliability of the NBSA. The code for the History Check was written in Python and can be found in Appendix A.5.1.

Distance measurements

The history check algorithm needs two distances. These distances are measured in squares instead of meters to make every different nursing home more general from the algorithms point of view. The first one is the maximum distance that can be traveled by a nursing home resident within the time since the previous measurement. This distance is acquired by simply multiplying the time that has passed since the last measurement with the maximum velocity a nursing home resident can move. During experimental testing, an average top speed of 0,4 m/s was found. This testing, however, was done very roughly and without observing actual nursing home residents. The estimation of the average top speed of nursing home residents can therefore quite easily be improved upon by testing with actual residents. Within the nursing home where most of our data was recorded, the estimated average top speed translates to 0,05 squares per second. This speed is based on slow moving elderly, possibly using a walker. There exists however quite some variance in the different top speeds even without taking into account that some residents have mobility scooters or wheelchairs that can be pushed by other more mobile people and therefore drastically increasing their top speed. This measurement may therefore be improved by implementing a variable, person dependent top speed or, at the very least, increasing this top speed when the resident uses a mobility scooter or wheelchair.

The second distance value needed by the history check is the minimum travel distance between the two measured points. This is calculated via a function in Python which uses the map implementation as described in Section 4.3.1. The function is loosely based on Dijkstra's algorithm [40] and finds the two measured points on the map and checks whether or not they are located on the same floor. If this is the case, the distance between those two points as shown in Figure 4.10 is measured. This is done by starting at the previously measured point. From there, all horizontally and vertically adjacent hallway squares are filled with a minus one value. This minus one represents that these squares are a distance of one square away from the start point. A negative value is used in order to be able to differentiate between a distance value of one and a room numbered '1'. The next step is to fill out every hallway square horizontally and vertically adjacent to the squares containing a minus one with a minus two and so on until the the currently measured point is reached. When this is the case, the corresponding distance is made positive again and returned.

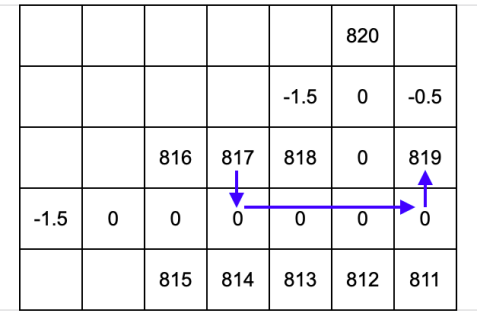


Figure 4.10: The shortest route from room 817 to 819.

When the two compared location points are not located on the same floor, the algorithm looks for the closest staircase which leads either up or down, depending on the position of the currently measured point in relation to that of the previously measured point. It does this using the same methodology it uses when looking for its end point on the same floor, but instead of looking for the target location, it looks for the closest viable staircase. When the distance to this staircase is reached, the algorithm looks to the next floor whichever way is required to get closer to the floor where the target location is and adds one square to the total distance. It does this until it either reaches the right floor or until the top or bottom of the staircase has been reached. In the first case, the history check algorithm measures the distance between the staircase and the target point. When the staircase ends before the target floor has been reached, the algorithm looks for the next staircase, measures the distance to it in the same way as before and adds it to the total distance. Then the algorithm looks at the next floor as before. These steps are repeated until the target floor, which contains the currently measured point, has been reached. It will then measure the distance to the currently measured point and add it to the total distance to reach the full minimum traveled distance.

When lifts are present in the building, they are handled in the exact same way as stairs with the exception that whenever a lift is present, it is always prioritized over the stairs, since most of the residents experience many difficulties taking stairs or are simply unable to do so because of supporting equipment like wheelchairs or walkers.

There are some assumptions and simplifications leading to an imperfect physical distance measurement. These are listed below:

- When a lift is present, the algorithm will always prioritize it over taking the stairs.
- When looking for lifts or staircases, the algorithm always looks for the closest of each to the starting point instead of the most efficient overall.
- using lifts or stairs adds the same distance tot the total distance as moving one square (namely one). In practice, a different (higher) value would be more realistic.
- Stairs can only be traveled in the direction that goes to a floor which lies closer to the floor of the end point. This may lead to problems when for example two wings are only connected by the ground floor, meaning that in order to reach the other wing, one needs to go down and then back up again.
- Since the array representation of a nursing home is very simplified, the distances determined using it will be less accurate.

The first of these simplifications is an assumption. Even though from observations in the nursing homes this assumption seems to be valid, no proper testing was performed as of yet to validate it. The second third and fourth listings are simplifications to make development easier. In later iterations of the product these may yet be properly implemented. However, since these simplifications will most often lead to a longer measured shortest path, they will make the filter wider instead of narrower. This means that no valid measurements are lost and the ones that unjustly pass through the check, may still be ignored through other means. The last simplification is an imperfection caused by the general map structure and can therefore not be solved in the history check itself. To fully eliminate this, an entirely different map structure must be implemented. As this is not practical for several reasons, it is suggested as a future work.

Estimating likelihood
The next step in the history check is to determine the likelihood that a measured point is reliable or not using the aforementioned maximum distance and the minimum traveled distance. Due to measurement assumptions, imperfections in the model and measurement errors, these distance values are not precise. Since all errors are assumed to be equally distributed, both curves were assigned a normal distribution. This means that for both measurements a variance value needs to be determined.

First, the shortest distance calculation is evaluated. There are two contributors to the variance of this calculation. The first is the error due to simplification of the model and the second is the error in measurement. The measurement error is easiest to determine by simply comparing the predicted locations at a number of times to the actual location the test subject was at that time. Then the standard procedure is followed of averaging the sum of squares to reach a variance of 1.3 squares. The variance due to simplification error is linearly dependant on the shortest distance. The multiplication constant for an estimated set of deviations with an average of 0.14 squares was empirically determined to be 0.09. Since these two error sources do not influence each other, the corresponding variances can simply be added.

Secondly, the measured distance is evaluated. Here too, there exist two major contributors to the measurement error, namely the simplification of the model and an inaccuracy in the estimation of the maximum distance of the resident. The simplification of the model is handled the same way as in the shortest distance calculation resulting in the same deviation. This velocity error is fairly straightforward to determine based on the experiment performed in determining the max velocity. The average velocity deviation is then multiplied with the estimated minimum distance to transform it into a distance deviation. Once again, these two errors are independent, so their variances can simply be added to arrive at the overall variance. All variance metrics are presented in Table 4.2. The resulting distributions both have nonzero values at negative distance values, which at first glance seems to be a problem. These negative distance values can be interpreted as distances in the negative direction. and be processed accordingly later on.

Table 4.2: Different error sources and their associated variances. d represents the relevant distance values and d_{min} and d_{max} represent the shortest route distance and the calculated maximum distance respectively. All distances are in squares

Error	Variance $[squares^2]$
Velocity error	$0.069 \cdot d^2$
Simplification error	$0.09 \cdot d$
Measurement error	1.3
Shortest route	$0.069 \cdot d_{min}^2 + 0.09 \cdot d_{min}$
Maximum distance	$0.11 \cdot d_{max} + 1.3$

Using these two normal distributions, a reliability metric can be produced. The first step is determining how likely it is that the true maximum distance has not yet been reached based on the measured maximum distance. This is then evaluated for the full range of possible input distance values. In order to accomplish this, the Probability Density Function (PDF) of the maximum distance distribution is translated into its corresponding Cumulative Density Function (CDF) to get a figure representing the likelihood that the maximum distance has been reached. Because the probability is needed that this is not the case, we subtract this curve from one resulting in the blue line in figure 4.11. Since the purpose of this curve is to suppress distance measurements that lie outside of the realistic maximum distance travelled, this can be thought of as a low pass filter in the distance domain with the calculated maximum distance as the cut-off distance. This filter can then be used to suppress the measured minimum distances which are unlikely to be valid. Since the distribution for the shortest travel distance also has nonzero values at negative distances, the filter is made symmetric by mirroring it around the $distance = 0$ axis. This is because these negative values can be interpreted as distances in the opposite direction from the measured one. Therefore, the same theoretical maximum distance holds in the negative direction.

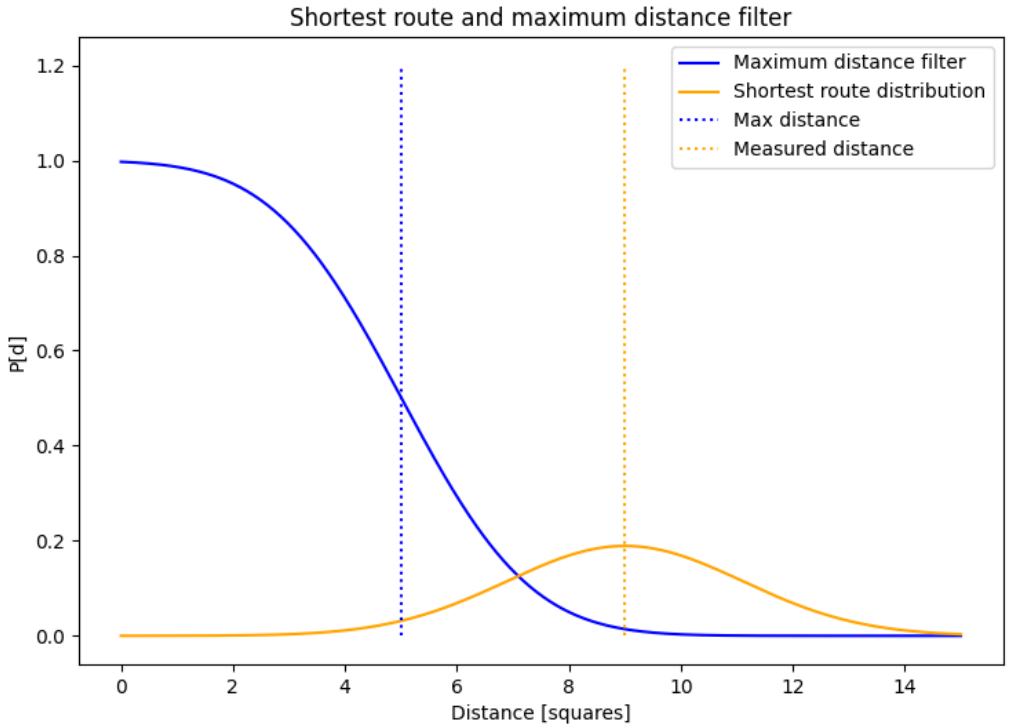


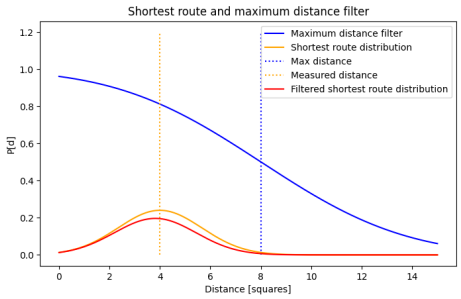
Figure 4.11: The two measures determining the confidence of the history Check

The next step is to determine the probability that the shortest distance traveled is lies within the maximum distance limit. To achieve this, the CDF based filter figure is multiplied by the shortest distance normal distribution. Four examples for this are given in Figure 4.12. In these figures, four scenario's are shown where in each case the maximum distance is kept constant and the measured distance is varied. For a measured distance that is much smaller then the calculated maximum distance, the figure is mostly unchanged. When the measured distance increases approaching the the calculated maximum distance, the height of the curve decreases. When the distance increases even further, the height of the curve eventually approaches zero. To translate this curve into a probability that the the measured distance value is realistic, the red line, which is the result of the multiplication of the shortest distance distribution and the maximum distance filter, is integrated. A value close to one will be present at places where the Gaussian fits neatly underneath the curve, as in Figure 4.12a, down to a value close to zero when the Gaussian lies mostly outside of the filter, as in Figure 4.12d.

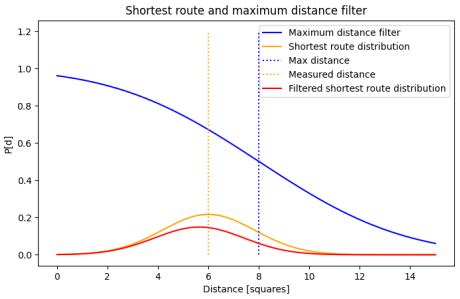
Testing the algorithm

When testing on real pre-recorded data it became immediately clear that an maximum distance offset was needed. The first reason for this, was that the maximum traveled distance should always be at least one square, regardless of the time difference between measurements. This was for the simple reason that at high update frequencies, the person was indicated to not be able to move at all, since the time difference would not get high enough for the maximum distance to exceed one square at the given maximum velocity. Another reason an maximum distance offset is needed is because of the fact that the model is not yet able to to differentiate between hallway and rooms. Therefore when on the map in Figure 4.3 the panic button moves from the hallway in between rooms 813 and 818 to the hallway between rooms 814 and 817, which is a distance of one square, this would be registered as moving from room 813 to room 814, which is a distance of three squares. Together these result in a maximum distance offset of three squares.

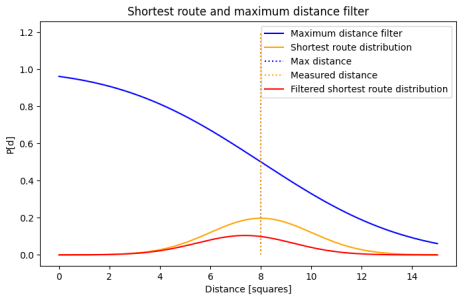
Furthermore, when testing, it became apparent that the history check outcome should not play a large role in validating short distance shifts. Therefore it was chosen to only take the history check into account when its result becomes lower than 50%, as discussed in Section 4.3.3. This is due to the fact that at small displace-



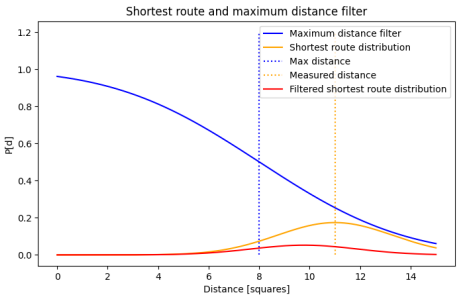
(a) Measured distance « Maximum Distance. $P_{hist} = 0.79$



(b) Measured distance < Maximum Distance. $P_{hist} = 0.65$



(c) Measured distance = Maximum Distance. $P_{hist} = 0.50$



(d) Measured distance > Maximum Distance. $P_{hist} = 0.30$

Figure 4.12: History check probability distributions for varying Measured Distance values. P_{hist} represents the chance that the measured distance value is possible using the history check

ment, the other localization steps play a much more vital role, since it is roughly as plausible for a resident to have moved half is maximum distance as it is to have moved three quarters of it. This is partly already taken into account in the outcome of the check, but because the deviation in the maximum speed of the elderly residents is relatively high, the variance of the maximum distance is fairly large as well. This result in a steeper slope in the maximum distance Figure 4.11 far away from the measured maximum distance than ideal. This can be corrected by reducing the maximum distance variance as suggested in Section 4.4.2, but for now it was decided to not take the actual outcome of the history check into account if its result is lower than 50%.

Once these adaptations were put in place, the history check performed as expected. The localized floor becomes much more consistent. Especially the two floor jumps are almost completely removed. especially far away from a staircase or lift, the floor this effect is very noticeable. This of course also has the drawback that once a wrong decision has been made, it echoes through in the following decisions. Since it was assumed that all lifts and staircases are equipped with an ACS, the decision regarding the floor the panic button is on is based fully on ACS data, since these are assumed to be more reliable. In field testing for the ACS has yet to be performed however, so this assumption may turn out to be false. If this assumption turns out to be false or no ACS is installed in a given nursing home, the history check will offer a good alternative. This means that in the current situation, the history check only filters out large distance differences in a flat area. These do, however, seldom occur within the used data, meaning that the validity of the history check in this case is difficult to determine.

4.5. Validation of the Prototype

In Sections 4.4.1 and 4.4.2 the results of testing the NBSA with the individual reliability checks are discussed. The results of testing the total system, the NBSA with both reliability checks implemented, is discussed in this section.

The algorithm was tested with a dataset containing an hour of data recorded in nursing home Laurens in Blijdorp. How this data was obtained is explained in Section 3.4. Several elements were used for this validation. Two coupling files: one to see which BedSense is coupled to which room and another to see which ACS is coupled to which floor. The map described in Section 4.3.1. Lastly, a test file containing data from the ACS as the ACS is not yet in place. However, for testing purposes it was vital that this has also been tested.

As the prototype is not able to distinguish between being in a room or being in the hallway in front of the room, this error is not accounted for. For validating the results, only the room to which the panic button is closest to will be checked. In Table 4.3 is stated when a room is seen as correct and when not in combination with the actual location of the panic button. The decision of when a determined room is indeed the correct location is based on the map of the nursing home (see Figure 3.5. When the panic button is in the hallway between for example rooms 813 and 814, rooms 813, 818, 814 and 817 are all correct. But when the panic button is in the hallway between 813 and 818, only 813 and 818 are correct. Whenever the panic button is inside of a room, only the room where it is inside, is seen as a correct determination.

Table 4.3: The determined location in comparison to the actual location and whether the combination of both is seen as correct or incorrect.

Determined location	Actual location	Correct or incorrect
811	811 in room	Correct
811	819 in room	Incorrect
811	811-819 hallway	Correct
811	811-812 hallway	Correct
819	819 in room	Correct
819	811-819 hallway	Correct
819	811-812 hallway	Correct
812	812 in room	Correct
812	811-812 hallway	Correct
812	812-813 hallway	Correct
812	813-814 hallway	Incorrect
813	813 in room	Correct
813	813-818 hallway	Correct
813	813-814 hallway	Correct
813	814-817 hallway	Incorrect

In Table 4.4, the results of the NBSA can be found. As can be seen in this table, the NBSA often determines the correct room on the correct floor (with an accuracy of 79.9%). This accuracy is determined by Equation 4.10. The accuracy of the algorithm with a maximum error of one room on the correctly determined floor is 95.6%, and is calculated by Equation 4.11.

Table 4.4: The error on the determined room and floor on the dataset from Section 3.4.

Error	Amount
Correct room on correct floor	183
One room error on correct floor	35
Two rooms error on correct floor	3
Correct room on one floor above/below	3
Correct room on two floors above/below	5
Total outputs	229
Accuracy (correct room and correct floor)	79.9%
Accuracy (within one room error and correct floor)	95.6%

$$\text{Accuracy} = \frac{\text{Correct room on correct floor}}{\text{Total outputs}} \tag{4.10}$$

$$\text{Accuracy} = \frac{\text{Correct room on correct floor} + \text{One room error on correct floor}}{\text{Total outputs}} \tag{4.11}$$

5

Discussion

The primary goal of this project was to create an extension to the existing Momo Medical BedSense design, that allows for location tracking on residents using panic buttons that are already deployed in the field. Therefore, an algorithm has been designed that performs localization, using the RSSI of signals transmitted by these panic buttons. The result is a free of additional hardware system which can attain this location. Chapter 2 presents the full system requirements and localization system requirements, both of these are discussed in further detail.

Full system requirements

- No trouble is caused by RF interference due to both systems being operative in the same nursing home. The solution to this requirement is stated in [2], because the solution to this problem is achieved by them entirely.
- Both systems output their data at least once per minute. For the ACS, this data consists of the tags registered by each door sensor. For the localization system, this data consists of locations of all panic buttons that were within the system's range.
- For the system to work in a nursing home, some installation has to be done. For example, the ACS sensors need to be placed at every room door and all other passageways. Also, the layout of the rooms needs to be translated to the map format used by the system. However, no (extensive) calibration is needed. The ACS does not need any calibration at all and the localization system has been designed in such a way that it works after giving it general layout parameters, such as the distances to different types of neighbouring rooms.
- Both systems do not require any training. Once the door sensors are installed and the panic buttons and tags are coupled to the right rooms, both systems works in an uninterrupted manner.
- Lastly, the ACS system works in every nursing home. The localization system works in the majority of nursing homes. Due to possible abnormal building layouts, the current map features might not support functionality of the reliability check algorithms.

Localization system requirements

The localization system requirements are divided into functional and non-functional requirements, as described in Chapter 2,

- Every minute, or any other interval that is set between sending alive signals, the system creates a textual message with a resident's room number, the current location of that resident, and the current time.
- The location format is clear to any reader familiar with the layout of the nursing home.
- As the localization takes place every minute, it can be sent to the Momo Medical database and thus be requested by the Momo Medical app.

- To improve the accuracy of the algorithm, the localization system makes good use of the ACS' data, that it fetches from the Momo Medical database where the ACS stores its data.
- The localization system works three dimensional, thus across multiple floors.

Regarding the non-functional requirements,

- Of all performed localizations, 95.6% is within the required accuracy of one room variance around the actual room.
- 79.9% of all performed localizations are determined correctly, meaning the actual room where the panic button is present, is estimated by the localization system.
- As said, the messages that are stored in the database, contain a resident's room number that functions as their identification.

6

Conclusion

The goal of this Bachelor Graduation Project was to design a real-time localization system to be able to find residents within the nursing home that they live in. For this, two different approaches were worked out, tested, and further improved.

Firstly, multilateration was explored. In theory, this seemed very promising, but it turned out to be inadequate for implementation within nursing homes. The primary causes of this are the effects by shadowing and reflections in the transmission medium, the nursing home. It is learned that with these phenomena there is a need of additional systems, such as filters and other radio frequency signal noise reducing techniques, to smooth the RSSI measurements for trilateration, and later multilateration purposes.

Secondly, the nearest BedSense algorithm was developed. Due to its different approach to localization compared to multilateration, namely its reliability checks, resulted this system to be much more robust to variances in RSSI, the main parameter this systems uses for localization. By combining the history check and the neighbour comparison algorithms within the nearest BedSense algorithm, this system reached an accuracy of 95.6%, that is, to localize a person within a maximum error of one room next to the actual location.

Acceptable accurate localization with the use of BedSenses, from Momo Medical, within nursing homes, using a the type of panic button presented in this project, is most easily achieved by translating the RSSI values, measured by BedSenses, using the NBSA developed throughout this thesis report. For Momo Medical, to further implement this system it should be used as peripheral software within their existing nurse supporting system.

7

Recommendations and Future Work

In the previous sections the developed localization system is described. As this system has been developed during a short period of time there is still room to improve the systems performance and widen its employability. In this section, some suggested recommendations and future work are shorty described.

7.1. Recommendations

Extend the implementation of the path loss model to better account for walls and objects
Buildings can vary in type of materials used. For example, walls can be made of reinforced concrete or plaster. These materials have different influence on the propagation of a signal. The way in which the RSSI from BedSenses is interpreted, by the different algorithms, can be adjusted to the information on these materials.

Adapt to the difference in the standard RSSI of each BedSense
In a baseline measurement at a single distance each BedSense shows a slightly different RSSI, meaning that there is an offset between the BedSenses in the ability to receive signals. To solve this, each BedSense should either be calibrated to receive equal strong RSSI as others, or its offset, in respect to a mean value, should be registered in a "Room-BedSense coupling" alike file.

Experiment with sending a burst of signals from the panic button to the BedSense for a single localization
An attempt to calculate more accurate distances from the RSSI by sending a burst of signals and averaging all RSSIs that follow from this burst.

Experiment with sending signals from the panic button to the BedSense at different strength
Another attempt to calculate more accurate distances from the RSSI. By the path loss model, close to the transmitter the signal decreases more heavily than further away from it. By sending two different signal strengths, two different path loss models are expected, in the sense that one has lower RSSI values than the other. This may result in different reflections and shadowing effects in both signals, which, by comparison of the two, could to some extent be recognized and filtered out.

7.2. Future Work

Differentiate between an alive signal and signal due to pressing the panic button
A panic button's location is computed and stored every x amount of seconds or minutes - depending on the interval setting for sending an alive signals by the button. However, in case a resident needs help and therefore presses the button, this situation should only lead to computing a location and storing that. It should also alert the appropriate caregiver. This later functionality is still to be implemented.

Differentiate between hallway and rooms when localizing a person
The way NBSA uses the reliability checks does not allow for differentiating between the location being in a hallway or inside a room. Additional steps in interpreting the outcomes from these checks might enable this

option.

Omnidirectional antenna in panic button
Effects due to polarization mismatches could be reduced by making use of a panic button that has a omnidirectional antenna.

Improve the map implementation to allow for different sizes of rooms and hallways
Currently, the map is build up by squares. Each single square represents a complete room or a (part of a) hallway irrespective of its size. Locations where sizes of rooms and hallways are significantly different from each other, might experience a better accuracy if the map allows for a these different sizes. Naturally, the algorithms that make use of the map will have to be adapted to this change of map layout.

Couple the maximum and average speed of a resident to the panic button he or she carries
While the greatest part of all residents are elderly people that walk at about the same speed, some might deviate in speed from others. For example, residents moving with a walker tend to be slower than average, and residents sitting in a wheelchair or even a mobility scooter tend to be faster than average.

Create an functionality for care givers to wear a button as well
This idea came from the care givers themselves during an interview. They suggested to not only have residents carry these panic buttons, but have care givers carry them too. This way those care givers can, by simply pressing their button, bleep each other in case they need help. For example after someone fell and needs to be lifted upright.

Create panic button alike devices for utilities so these can also be localized
Another idea that came from the care givers during the interview. Not only do care givers seek to more easily get information on the whereabouts of a resident, they would like to more easily find utilities (such as lifting aids to move people that have to be carried) within the building.

Bibliography

[1] Momo Medical: Home. url: https://momomedical.nl/.

[2] A. Bennebroek and S. Ramezani. "Access control system based on UHF RFID technology". In: (2022).

[3] A. Yassin et al. "Recent Advances in Indoor Localization: A Survey on Theoretical Approaches and Applications". In: IEEE Communications Surveys Tutorials 19.2 (2017), pp. 1327–1346. doi: 10.1109/COMST.2016.2632427.

[4] N.A. Azmi et al. "A Survey of Localization using RSSI and TDoA Techniques in Wireless Sensor Network: System Architecture". In: 2018 2nd International Conference on Telematics and Future Generation Networks (TAFGEN). 2018, pp. 131–136. doi: 10.1109/TAFGEN.2018.8580464.

[5] F. Zafari, A. Gkelias, and K.K. Leung. "A Survey of Indoor Localization Systems and Technologies". In: IEEE Communications Surveys Tutorials 21.3 (2019), pp. 2568–2599. doi: 10.1109/COMST.2019.2911558.

[6] F. Gustafsson and F. Gunnarsson. "Positioning using time-difference of arrival measurements". In: 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). Vol. 6. 2003, pp. VI–553. doi: 10.1109/ICASSP.2003.1201741.

[7] M.A. Spirito and A.G. Mattioli. "On the hyperbolic positioning of GSM mobile stations". In: 1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No.98EX167). 1998, pp. 173–177. doi: 10.1109/ISSSE.1998.738060.

[8] S. Maddio, A. Cidronali, and G. Manes. "RSSI/DOA Based Positioning System for Wireless Sensor Network". In: New Approach of Indoor and Outdoor Localization Systems. 2012, pp. 139–162. doi: http://dx.doi.org/10.5772/50380.

[9] M. Passafiume et al. "An enhanced triangulation algorithm for a distributed RSSI-DoA positioning system". In: 2016 European Radar Conference (EuRAD). 2016, pp. 185–188.

[10] B. Gaffney. "Considerations and Challenges in Real Time Locating Systems Design". In: 2008.

[11] M. Shchekotov. "Indoor localization methods based on Wi-Fi lateration and signal strength data collection". In: 2015 17th Conference of Open Innovations Association (FRUCT). 2015, pp. 186–191. doi: 10.1109/FRUCT.2015.7117991.

[12] Y. Chen and H. Kobayashi. "Signal strength based indoor geolocation". In: 2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No.02CH37333). Vol. 1. 2002, 436–439 vol.1. doi: 10.1109/ICC.2002.996891.

[13] B. Yang et al. "A Novel Trilateration Algorithm for RSSI-Based Indoor Localization". In: IEEE Sensors Journal 20.14 (2020), pp. 8164–8172. doi: 10.1109/JSEN.2020.2980966.

[14] H. Obeidat et al. "Indoor environment propagation review". In: Computer Science Review 37 (2020), p. 100272. issn: 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2020.100272.

[15] F. Zargoun, I. El-henawy, and N. Ziedan. "Effects of Walls and Floors in Indoor Localization Using Tracking Algorithm". In: International Journal of Advanced Computer Science and Applications 7 (Mar. 2016). doi: 10.14569/IJACSA.2016.070305.

[16] A. Moreira et al. "Wi-Fi fingerprinting in the real world - RTLS@UM at the EvAAL competition". In: 2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN). 2015, pp. 1–10. doi: 10.1109/IPIN.2015.7346967.

[17] Y. Kim and H. Bang. "Introduction to Kalman Filter and Its Applications". In: Introduction and Implementations of the Kalman Filter. Ed. by Felix Govaers. Rijeka: IntechOpen, 2018. Chap. 2. doi: 10.5772/intechopen.80600. url: https://doi.org/10.5772/intechopen.80600.

[18] M. Yavari and B.G. Nickerson. Ultra Wideband Wireless Positioning Systems. Tech. rep. TR14-230. University of New Brunswick, Mar. 2014, p. 27.

[19] DS-PDEBP1-EG2-we. url: https://www.hikvision.com/europe/products/Alarm-Products/Hikvision-Intrusion-Detector/Wireless-Detector/ds-pdebp1-eg2-we/.

[20] DS-PDEBP1-EG2-WE. DS-PDEBP1-EG2-WE. 2020.

[21] Datasheet, Ultra-low power, high performance, sub-1 GHz transceiver. DS11896. June 2022.

[22] J.A. Shaw. “Radiometry and the Friis transmission equation”. In: American Journal of Physics 81.1 (2013), pp. 33–37. doi: 10.1119/1.4755780. eprint: https://doi.org/10.1119/1.4755780. url: https://doi.org/10.1119/1.4755780.

[23] G. Mao, B.D.O. Anderson, and B. Fidan. “WSN06-4: Online Calibration of Path Loss Exponent in Wireless Sensor Networks”. In: IEEE Globecom 2006. 2006, pp. 1–6. doi: 10.1109/GLOCOM.2006.497.

[24] Z Şahinoğlu, S. Gezici, and I. Güvenç. Ultra-wideband positioning systems: Theoretical limits, ranging algorithms, and protocols. Cambridge University Press, 2010.

[25] S. Lee et al. “Kalman Filter-Based Indoor Position Tracking with Self-Calibration for RSS Variation Mitigation”. In: International Journal of Distributed Sensor Networks 11.8 (2015), p. 674635. doi: 10.1155/2015/674635. url: https://doi.org/10.1155/2015/674635.

[26] P. Barsocchi et al. “Forecast-Driven Enhancement of Received Signal Strength (RSS)-Based Localization Systems”. In: ISPRS International Journal of Geo-Information 2.4 (2013), pp. 978–995. issn: 2220-9964. doi: 10.3390/ijgi2040978. url: https://www.mdpi.com/2220-9964/2/4/978.

[27] J. Pyun, S. Subedi, and E. Llobet. “Practical Fingerprinting Localization for Indoor Positioning System by Using Beacons”. In: Journal of Sensors (Dec. 2017). doi: 10.1155/2017/9742170. url: https://doi.org/10.1155/2017/9742170.

[28] F.T. Ulaby and U. Ravaioli. Fundamentals of applied electromagnetics. 7 Global. Pearson, 2015.

[29] F. Delfino et al. “Electromagnetic plane wave scattering from building surfaces”. In: The international journal for computation and mathematics in electrical and electronic engineering 25.4 (2006), pp. 1007–1018. doi: https://doi.org/10.1108/03321640610684132.

[30] O. Landron, M.J. Feuerstein, and T.S. Rappaport. “In situ microwave reflection coefficient measurements for smooth and rough exterior wall surfaces”. In: IEEE 43rd Vehicular Technology Conference. 1993, pp. 77–80. doi: 10.1109/VETEC.1993.510972.

[31] C.R. Lomba, R.T. Valadas, and A.M. de Oliveira Duarte. “Experimental characterisation and modelling of the reflection of infrared signals on indoor surfaces”. In: IEE Proceedings-Optoelectronics 145.3 (1998), pp. 191–197.

[32] P. Singam, S. Narendranath, and P. Sreekumar. “Design and development of soft x-ray multi-layer optics”. In: (Nov. 2015). doi: 10.13140/RG.2.1.1828.8082.

[33] Á. De La Llana Calvo et al. “Modeling Infrared Signal Reflections to characterize indoor multipath propagation”. In: Sensors 17.4 (2017). doi: https://doi.org/10.3390/s17040847.

[34] K. Kaemarungsi and P. Krishnamurthy. “Properties of indoor received signal strength for WLAN location fingerprinting”. In: The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. 2004, pp. 14–23. doi: 10.1109/MOBIQ.2004.1331706.

[35] ANTX150P116B08683Pulse Electronics: Mouser. url: https://nl.mouser.com/ProductDetail/Pulse-Electronics/ANTX150P116B08683?qs=MLItCLRbWsxhiIxzWM7gXA%5C%3D%5C%3D.

[36] Laurens Blijdorp. url: https://laurens.nl/locatieoverzicht/blijdorp.

[37] S. Garcia et al. “Big data preprocessing: methods and prospects”. In: Big Data Analytics 1.9 (2016), pp. 191–197. doi: https://doi.org/10.1186/s41044-016-0014-0.

[38] T. Sauer. “Least Squares”. In: Numerical Analysis. Hoboken New Jersey: Pearson, 2018. Chap. 4.

[39] B.T. Fang. “Simple solutions for hyperbolic and related position fixes”. In: IEEE Transactions on Aerospace and Electronic Systems 26.5 (1990), pp. 748–753. doi: 10.1109/7.102710.

[40] A. Javaid. “Understanding Dijkstra’s algorithm”. In: Available at SSRN 2340905 (2013).

Appendices

A

Python Code

A.1. Localization

```
# BAP Q1 2022-2023 Subgroup 2
# Localization Algorithm Iteration 7
# Enables us to determine to which room the person is closest to (in 3D, by executing both the history and neighbour comparison check. Grouped by 10 locations and the most frequent one is picked.
# Version 2022/12/18 11:30

import math
from statistics import mode
import numpy as np
from datetime import datetime

import Coupling_bed_bedsense_Version_2 as coup
import Classes_Room_Version_5 as Class
import Data_Version_5
import Neighbour_Values_Version_3 as values
import Neighbour_Comparison_4_0 as check
import History_Check_3 as history


def loc_1():
    # This function has no inputs
    # It loads multiple variables from other functions
    # Predicts the floor on which the panic button is.
    # For every row in the dataset, take the 5 maximum values received
    # For all these values, locate then on the map (layout)
    # For all these locations, find the same room on the predicted floor.
    # Make a prediction about the location of the panic button
    # Print this prediction
    # This function has no outputs

    # Initializing values
    last_prediction = None
    last_timestamp = None

    # State directories of the couplings and the map
    loc = "C:\\Users\\borin\\Documents\\BAP\\koppelingen_laurens_2.csv"
```

```
    loc2 = "C:\\Users\\borin\\Documents\\BAP\\Test data subgroup 1 - koppelingen.csv"
    map_loc = "C:\\Users\\borin\\Documents\\BAP\\textual_map_laurens_blijdorp - Floor 6-10.csv"

    # Calling the function Couplings to obtain the beds coupled to the bedsenses
    bed_bedsense, door_floor = coup.Couplings(loc, loc2)

    # Calling the function Classes_Room to obtain a dictionary of all rooms with as key the bedsense number belongin to the room, and a map of the building
    dictionary, layout = Class.import_rooms(loc, map_loc)

    # Calling the function Data to obtain a grouped by 1 sec dataframe, an array with bedsense numbers, the data coming from the Access Control System (ACS) and an array with ACS numbers
    df, bedsenses, door_data, doors = Data_Version_5.Data()

    time = df[:, 0]  # Time column
    data = df[:, 1:]  # Data columns for all bedsenses
    doortime = door_data[:, 0]
    door_data = door_data[:, 1:]

    locations = []

    j = 0
    # Iterate through the data
    for row in data:
        nan = np.isnan(row.astype(float))  # Check if a row contains NaN values
        x = np.all(nan == True)  # Check if all values are NaN in a row

        if not x:  # If row contains at least one non NaN value then
            new_row = []
            for i in range(len(row)):
                if not math.isnan(row[i]):  # If the value is not NaN
                    new_row.append([i, row[i]])  # Add both value and index to a new list
            listvalues = sorted(new_row, key=lambda x: x[1], reverse=True)  # Sort the list based on the rssi values
            index_array = []
            for k in listvalues:
                index_array.append(k[0])  # Add indices to another list

            max_values = index_array[0:5]  # Take the 8 strongest bedsenses
            strongest_bedsenses = []
            # Predict the floor on which the person is existing
            floors = []
            for value in max_values: # For one of the highest values find the corresponding bedsensenumber and bed
                strongest_bedsense = bedsenses[value]
                strongest_bedsenses.append(strongest_bedsense)
                bed = dictionary[str(strongest_bedsense)]
                floors.append(bed.floor) # From all highest values collect the floor

            door_time_small = []
            for t in range(len(doortime)):
                if doortime[t] < time[j]:
                    door_time_small.append(doortime[t])
```

```
85
86              door_data_small = door_data[0:len(door_time_small)]
87
88              door_index = np.where(door_data_small == 1)
89              door_index = list(zip(door_index[0], door_index[1]))
90              if not len(door_index) == 0:
91                  door = doors[door_index[-1][1]]
92                  floor = door_floor.loc[door_floor['ACS'] == door, 'door'].iloc[0] #
                     ↪ The floor on which we are is the floor on which the ACS is
93              else:
94                  floor = round(sum(floors) / len(floors))
95
96              row_copy = row.copy()  # create a copy of the data row
97              for key in dictionary:  # iterate through dictionary
98                  if not len(np.where(bedsenses == int(key))[0]) == 0:  # if the key
                     ↪ has collected data and therefore is in bedsenses
99                      bed = dictionary[key]  # collect the instance of the key
100                     if bed.floor != floor:  # if the instance is not on the floor
101                         row_copy[np.where(bedsenses == int(key))[0][0]] = np.nan  #
                             ↪ set the value registered by the bedsense to nan
102             if np.all(np.isnan(row_copy.astype(float))): # If the row now only
                 ↪ contains NaN values, the floor is the mean of the array floors
103                 floor = round(sum(floors) / len(floors))
104                 row_copy = row.copy()  # create a copy of the data row
105                 for key in dictionary:  # iterate through dictionary
106                     if not len(np.where(bedsenses == int(key))[0]) == 0:  # if the
                         ↪ key has collected data and therefore is in bedsenses
107                         bed = dictionary[key]  # collect the instance of the key
108                         if int(bed.floor) is not floor:  # if the instance is not on
                             ↪ the floor
109                             row_copy[np.where(bedsenses == int(key))[0][0]] = np.nan
                                 ↪ # set the value registered by the bedsense to nan
110             new_copy = []
111             for i in range(len(row_copy)):
112                 if not math.isnan(row_copy[i]): # If the value in row_copy is not NaN
                     ↪ (is on the predicted floor and received data) add the data to a
                     ↪ new array
113                     new_copy.append([i, row_copy[i]])
114             sorted_copy = sorted(new_copy, key=lambda x: x[1], reverse=True) #sort
                 ↪ the new array based on RSSI value from highest to lowest
115
116             bed_floor = []
117             room_floor = []
118             if len(sorted_copy) <= 5: # If less than 5 bedsenses received the signal
                 ↪ from the panic button
119                 for i in range(len(sorted_copy)): # Find the corresponding instance
                     ↪ and roomnumber
120                     bed_floor.append(dictionary[str(bedsenses[sorted_copy[i][0]])])
121                     room_floor.append(bed_floor[i].roomnumber)
122             elif len(sorted_copy) > 5:
123                 for i in range(5): # Find the corresponding instance and roomnumber
124                     bed_floor.append(dictionary[str(bedsenses[sorted_copy[i][0]])])
125                     room_floor.append(bed_floor[i].roomnumber)
126
127             bed = []
128             room = []
```

```
129             location = []
130             if len(listvalues) <= 5:
131                 for i in range(len(listvalues)): # Find the corresponding room,
                     ↪ neighbours, neighbour_values and confidence of the strongest
                     ↪ receivers
132                     bed.append(dictionary[str(bedsenses[listvalues[i][0]])])
133                     room.append(bed[i].roomnumber)
134                     location_array = np.where(layout == float(room[i])) # Find the
                         ↪ location of the room on the layout of the building
135                     for element in location_array:
136                         for i in range(len(element)):
137                             location.append(element[i])
138             else:
139                 for i in range(5): # Find the corresponding room, neighbours,
                     ↪ neighbour_values and confidence of the strongest receivers
140                     bed.append(dictionary[str(bedsenses[listvalues[i][0]])])
141                     room.append(bed[i].roomnumber)
142                     location_array = np.where(layout == float(room[i])) # Find the
                         ↪ location of the room on the layout of the building
143                     for element in location_array:
144                         for i in range(len(element)):
145                             location.append(element[i])
146
147             location = [location[n:n + 3] for n in range(0, len(location), 3)] #
                 ↪ Group the location of each bedsense as 3D coordinates
148             location_room_floor = []
149             if any(room_floor): # If any room on the predicted floor received a
                 ↪ signal
150                 location_floor_index = np.where(layout == float(room_floor[0])) #
                     ↪ Find the location of one of those rooms
151                 # Mirror the strongest receivers to the same rooms on the predicted
                     ↪ floor
152                 for element in location_floor_index:
153                     for i in range(len(element)):
154                         location_room_floor.append(element[i])
155                 floor_index = location_room_floor[0]
156                 floor_room = []
157                 for lst in location:
158                     lst[0] = floor_index
159                     floor_room.append(layout[lst[0]][lst[1]][lst[2]])
160                 floor_bed = []
161                 neighbours_floor = []
162                 confidence_floor = []
163                 for i in range(len(floor_room)):
164                     if not np.isnan(floor_room[i]):
165                         index =
                             ↪ bed_bedsense[bed_bedsense['Bed'].isin([int(floor_room[i])])].to_numpy
                             ↪ 1]  # Find the bedsense in dataframe belonging to the
                             ↪ roomnumber
166                         room_bedsense = index
167                         floor_bed.append(dictionary[str(room_bedsense)]) # Find the
                             ↪ instance belonging to the bedsense number
168                         neighbours_floor.append(floor_bed[i].map) # Find the
                             ↪ neighbours belonging to the instance
```

```
169                         values.Neighbour_values(row, floor_bed[i],
                             ↪ neighbours_floor[i], bed_bedsense, bedsenses) # Find the
                             ↪ neighbour values of the instance
170
                             ↪ confidence_floor.append(check.neigh_comp(floor_bed[i].values))
                             ↪ # Find its confidence
171                     else: # If the room number has no bedsense and thus the room has
                         ↪ number NaN, add low values (just for indexing)
172                         floor_bed.append(0)
173                         neighbours_floor.append(0)
174                         confidence_floor.append(-100)
175                 timestamp = time[j] / 1000  # Translate time from ms to s
176                 date_time = datetime.fromtimestamp(timestamp)  # Switch time in
                     ↪ seconds to readable date and time
177                 if last_timestamp is not None and last_prediction is not None:  # if
                     ↪ we have a previous prediction
178                     histories_floor = []
179                     deltat = timestamp - last_timestamp # Calculate the difference in
                         ↪ time from last measurement to current measurement
180                     for i in range(len(floor_room)):
181                         if not np.isnan(floor_room[i]): #Calculate the probability
                             ↪ that the prediction could be possible
182                             histories_floor.append(history.threeDhistory(deltat,
                                 ↪ layout, floor_room[i], last_prediction))
183                         else:
184                             histories_floor.append(0.1) # If roomnnumber is NaN add
                                 ↪ value just for indexing
185
186                     confident_history = []
187                     confident_room = []
188                     confident_confidence = []
189                     for i in range(len(histories_floor)):
190                         if histories_floor[i] > 0.5: #Check if the history is
                             ↪ possible (> 0.5)
191                             confident_history.append(histories_floor[i])
192                             confident_room.append(floor_room[i])
193                             confident_confidence.append([i, confidence_floor[i]])
194
195                     if not len(confident_confidence) == 0: # If there are values for
                         ↪ history larger than 0.5
196                         sorted_confidence = sorted(confident_confidence, key=lambda
                             ↪ x: x[1], reverse=True) # Sort the confidence for highest
                             ↪ to lowest
197                         prediction = floor_room[sorted_confidence[0][0]] #The room
                             ↪ prediction is the one with the highest confidence
198                     else: # If no history is higher than 0.5
199                         multiple = []
200                         for i in range(len(histories_floor)):
201                             multiple.append([i, histories_floor[i] *
                                 ↪ confidence_floor[i]]) # Multiply the history and the
                                 ↪ confidence
202                         multiple_sorted = sorted(multiple, key=lambda x: x[1],
                             ↪ reverse=True) # Sort the multiplication array
203                         prediction = floor_room[multiple_sorted[0][0]] # The room
                             ↪ prediction is the one with the highest multiplication
                             ↪ value
```

```
                else: # If is was no previous prediction
                    confident_confidence = []
                    for i in range(len(confidence_floor)):
                        if i > 0: # If the confidence is higher than 0
                            confident_confidence.append([i, confidence_floor[i]])
                    if not len(confident_confidence) == 0: # If there is at least one confidence higher than 0
                        sorted_confidence = sorted(confident_confidence, key=lambda x: x[1], reverse=True) # Sort the confidences from highest to lowest
                        prediction = floor_room[sorted_confidence[0][0]] # The room prediction is the one with the highest confidence
                    else: # If there is no confidence higher than 0
                        max_confidence = max(confidence_floor) # Take the highest confidence
                        max_index = np.where(confidence_floor == max_confidence)[0][0] # Find the index of the highest confidence
                        prediction = floor_room[max_index] # The room prediction is the one with the highest confidence

            else: # If there is no room on the predicted floor receiving a signal
                prediction = room[0] # The room prediction is the room receiving the strongest signal

            locations.append(prediction)

            last_prediction = prediction
            last_timestamp = timestamp
            j += 1

        else: #If row contains only NaN value
                    data = np.delete(data, j) #Delete all NaN row
                    time = np.delete(time, j) #Delete the time

        if len(locations) == 10: # If the list locations has 10 elements, give an output
            localization = mode(locations) # Find the value in locations which is present most
            print("De drukknop is het dichtst in de buurt bij kamer", int(localization), "op tijd", date_time)  # Print statement
            locations = []

    return
```

A.2. Data

A.2.1. Data Extraction

```
#BAP Q1 2022-2023 Subgroup 2
#Data extraction algorithm version 1
#Enables us to load data from the mijn-dev-joey server for the bedsenses of the 8th floor of Laurens
#Version 2022/11/23 15:21PM
#An example of this code was provided by Momo Medical

```

```

def extract_data():

    import datetime
    import pytz

    from momo_data_analysis.io.tsdb import MomoTSDB

    tsdb = MomoTSDB(url="https://mijn-dev-joey.momomedical.com/", api_key="PASSWORD")

    starttime = int(datetime.datetime(2022, 11, 15, 15, 34, 48).timestamp())
    endtime = int(datetime.datetime(2022, 11, 15, 15, 49, 2).timestamp())

    query = f"""SELECT rssi, device_id
            FROM "momo"."autogen"."peripheralState"
            WHERE ("peripheral_id" = '04680808' AND ("device_id" = '11025609' or "device_id" = '11025610' or "device_id" = '11025572' or "device_id" = '11025571' or "device_id" = '11025612' or "device_id" = '11025611' or "device_id" = '11025608' or "device_id" = '11025575' or "device_id" = '11025574' or "device_id" = '11025573'))
            AND time >= {starttime * 1000000000} AND time <= {endtime * 1000000000}"""
    df = tsdb._api.query_influx(query)
    df_2 = df.pivot_table(index='time', columns='device_id')
    df_2.columns = df_2.columns.droplevel(0)
    df_final = df_2.rename_axis(None, axis=1).reset_index()
    return df_final
```

A.2.2. Data Pre-processing

```
#BAP Q1 2022-2023 Subgroup 2
#Data processing algorithm version 5
#Enables us to preproces and group data being send within a timeframe from the data loaded from the server
#Also it enables us to group and filter a dataframe coming from the access control system
#Version 2022/12/18 11:30

import pandas as pd
import numpy as np
import re

import Extract_data_Version_1 as extract


def Data():
    #This function does not have an input
    #Function that extracts the dataframe of bedsense data (from the server) from the function extract data
    #It creates a numpy array containing all bedsenseID's that collected data
    #It creates a numpy array containing all DoorID's that collected data
    #It groups the bedsense data based on a timeframe, if multiple datapoints are collected in timeframe, the maximum value of each bedsense will be used
```

```
20      #It groups the door data based on a timeframe, if multiple datapoints are
        ↪  collecteed in timeframe, the ever last recorded value will be stored (all
        ↪  other data in timeframe is discarded only the last door to see button is
        ↪  saved)
21      #It creates a new numpy array containing the grouped bedsense data
22      #it creates a new numpy array containing the grouped door data
23      #This function has as output the new data (grouped), the array of bedsenseID's,
        ↪  the data of the door (grouped en filtered) and an array of doorID's
24
25      # To load the data
26      df = extract.extract_data()
27      df = df.apply(pd.to_numeric, errors='coerce')  # To convert all unknowns to NaN
        ↪  values
28
29      data = open("C:\\Users\\borin\\Documents\\BAP\\Test data subgroup 1 - full.csv")
30      sub1 = pd.read_csv(data)
31
32      # Creating the bedsense names
33      column_names = list(df.columns)  # Extracting headers from dataframe
34      bedsenses = []  # Create empty list for bedsenses
35      for name in column_names:
36          new_name = re.sub('\D', '', name)  # Delete all non numeric characters from
            ↪  list of headers
37          bedsenses.append(new_name)  # Add new headernames to bedsenses list
38
39      bedsenses[:] = [bed for bed in bedsenses if bed]  # Delete all empty headers
40      bedsenses = [eval(bed) for bed in bedsenses]
41      bedsenses = np.array(bedsenses)  # Transform bedsenses list into numpy array
42      df = df.to_numpy()  # Converting dataframe to a numpy array
43
44      #Creating door names
45      door_ids = list(sub1.columns)
46      doors = []
47      for door in door_ids:
48          new_door = re.sub('\D', '', door)
49          doors.append(new_door)
50      doors[:] = [door for door in doors if door]
51      doors = [eval(door) for door in doors]
52      doors = np.array(doors)
53      sub1 = sub1.to_numpy()
54
55      # Setting all values below -90 to zero, as it is no longer useful to work with
56      df[df < -90] = np.nan
57
58      # Variables
59      timestamp = [0]
60      door_timestamp = [0]
61      timeframe = 1000
62      time = df[:, 0]
63      door_time = sub1[:, 0]
64      new_door_data = []
65      new_data = []
66      j = False
67      m = False
68      k = 0
69      l = 0
```

```
70
71      for t in time:
72          i = 1  # Counter
73          t_index = np.where(time == t)  # Find the index of t
74          t_index = int(t_index[0])  # Setting the index of t to an integer value
75
76          if j == True:  # If we ran out of space j is set to true so that we break out
            ↪  of the for loop because we're out of space
77              break  # Break out of the loop
78          elif t_index - 1 < timestamp[-1]:  # When the index is lower than the index
            ↪  value in the timestamp list, pass the iteration
79              pass  # Skip iteration
80          else:
81              while (time[t_index + i - 1]) - t < timeframe:  # While the time
                ↪  difference between time t and the time at index of t + i:
82                  if t_index + i + 2 > len(time):  # To determine if we run out of the
                    ↪  column
83                      j = True
84                      break  # If statement above is true break out of while loop
85                  else:
86                      i += 1
87              else:  # If the while condition is no longer true, we add the latest
                ↪  index t_index + i to the timestamp list for saving
88                  timestamp.append(t_index + i)  # Add index to timestamp list
89                  minute = df[timestamp[k]: timestamp[k + 1]]  # Take a slice of the
                    ↪  dataframe containing 1 sec of data
90                  k += 1
91                  new_row = []  # A new row for all maximum values per bedsense per
                    ↪  timestamp
92                  for column in minute.T:  # Iterate through columns of matrix minutes
93                      max = np.nanmax(column)  # Find maximum value of a column
94                      new_row.append(max)  # Add value to list
95              new_data.append(new_row)  # Add row of maximum values per bedsense to a
                ↪  new list containing more rows like this for every timestamp
96      new_data = np.array(new_data, dtype=object)  # Transform data to numpy array
97
98      for t in door_time:
99          i = 1  # Counter
100         t_index = np.where(door_time == t)  # Find the index of t
101         t_index = int(t_index[0])  # Setting the index of t to an integer value
102
103         if m == True:  # If we ran out of space j is set to true so that we break out
            ↪  of the for loop because we're out of space
104             break  # Break out of the loop
105         elif t_index - 1 < door_timestamp[-1]:  # When the index is lower than the
            ↪  index value in the timestamp list, pass the iteration
106             pass  # Skip iteration
107         else:
108             while (door_time[t_index + i - 1]) - t < timeframe:  # While the time
                ↪  difference between time t and the time at index of t + i:
109                 if t_index + i + 2 > len(door_time):  # To determine if we run out of
                    ↪  the column
110                     m = True
111                     break  # If statement above is true break out of while loop
112                 else:
113                     i += 1
```

```
            else:  # If the while condition is no longer true, we add the latest index t_index + i to the timestamp list for saving
                door_timestamp.append(t_index + i)  # Add index to timestamp list
                door_minute = sub1[door_timestamp[l]: door_timestamp[l + 1]]  # take a slice of door dataframe containing 1 sec of data
                l += 1
                new_door_row = []  # a new row for all maximum values per door per timestamp

                for column in door_minute.T: #for each doorID in door_minute
                    max = np.nanmax(column)  # Find maximum value of a column
                    new_door_row.append(max)  # Add value to list
                new_door_data.append(new_door_row)
    new_door_data = np.array(new_door_data, dtype=object) #transfrom new_door_data to numpy array

    del timestamp[0]  # Delete start value
    del door_timestamp[0]

    return new_data, bedsenses, new_door_data, doors
```

A.2.3. Coupling

```
#BAP Q1 2022-2023 Subgroup 2
#Coupling between bed and bedsense and between ACS and floor
#Enables us to determine which bedsense is coupled to which bed and which ACS is coupled to which floor
#Version 2022/12/09 11:43AM

import pandas as pd

def Couplings(Directory, sub1directory):
    #This function has as input the directory to both the coupling files
    #Function that extracts the columns containing the roomnumber and the bedsenseID from the file
    #Function extracts the dataframe containing floor and the ID of the ACS
    #This funtion has as output a dataframe containing only the roomnumbers and the corresponding bedsenseID's and the dataframe containing the floors and corresponding ACS ID's

    #Open the file at the given directory
    data = open(Directory)
    df = pd.read_csv(data)

    sub1data = open(sub1directory)
    sub1df = pd.read_csv(sub1data)

    #Extracting columns containing beds and bedsenses from complete dataframe
    coupling = df.iloc[:, :2]

    return coupling, sub1df
```

A.3. Map

A.3.1. Map Matrix

```
# BAP Q1 2022-2023 Subgroup 2
# Map to matrix
# Find neighbour model for all rooms on a map with multiple floors
# Version 5 (each neighbour model is a 1-D array of length 70, floor at index -1)
# 2022/12/09 10:45

import numpy as np
import pandas as pd


def map_matrix(data):
    global floor_plan  # this becomes a global variable
    data = open(data)
    df_blueprint = pd.read_csv(data, header=None)  # load building's "blueprint" from csv into dataframe
    blueprint = df_blueprint.to_numpy()  # transform dataframe to np.array

    floor_height = (np.shape(blueprint)[0] - 1) / blueprint[0, 0]  # height of each individual map
    floor_plan = []
    for i in range(int(blueprint[0, 0])):
        floor_plan.append(blueprint[int(i * floor_height + 1):int((i + 1) * floor_height + 1), :])  # stack floor maps
    floor_plan = np.array(floor_plan)  # 3-D array of entire floor plan (stacked 2-D floor maps of each floor)

    # The neighbour map (per floor):
    # - X:      the room itself
    # - d:      direct neighbour
    # - s:      second direct neighbour
    # - i:      indirect neighbours (with hallway)
    # - b, a:   neighbour directly below/above the room itself
    # - bd, ad: direct neighbour on floor below/above
    # - bi, ai: indirect neighbours on floor below/above (with hallway)
    # - bb, aa: neighbour two floors directly below/above the room itself
    # |----|----------------|---------------|----------------|----|
    # |    |                |     i i i     |                |    |
    # |    |    bi bi bi    |       s       |    ai ai ai    |    |
    # |    | bi bd bd bd bi | i   d d d   i | ai ad ad ad ai |    |
    # | bb | bi bd b  bd bi | i s d X d s i | ai ad a  ad ai | aa |
    # |    | bi bd bd bd bi | i   d d d   i | ai ad ad ad ai |    |
    # |    |    bi bi bi    |       s       |    ai ai ai    |    |
    # |    |                |     i i i     |                |    |
    # |----|----------------|---------------|----------------|----|
    #
    # Lay-out matrix (3-D) (2-below - below - level - above - 2-above)
    # |---|----------------|---------------------|----------------|----|
    # |   |                |       46 35 36      |                |    |
    # |   |    21 10 11    |          31         |    67 56 57    |    |
    # |   | 20 9  2  3  12 | 45    30 23 24    37 | 66 55 47 49 58 |    |
    # | 0 | 19 8  1  4  13 | 44 34 29 22 25 32 38 | 65 54 47 50 59 | 68 |
    # |   | 18 7  6  5  14 | 43    28 27 26    39 | 64 53 52 51 60 |    |
    # |   |    17 16 15    |          33         |    63 62 61    |    |
```

```
    # |   |                |       42 41 40       |                |    |
    # |---|----------------|----------------------|----------------|----|
    #
    # Lay-out array
    # [0-68 (rooms), 69 (floor)]


    rooms = []  # map for each room at each floor (4-D)
    m = np.shape(floor_plan)[0]  # size of floor plan
    r = np.shape(floor_plan)[1]
    c = np.shape(floor_plan)[2]
    for l in range(m):  # iterate matrices (floors)
        for i in range(r):  # iterate rows (y)
            for j in range(c):  # iterate columns (x)
                neighbours = [np.nan]*70  # empty neighbours list of a room at (l, i, j)
                if not(np.isnan(floor_plan[l, i, j])) and (floor_plan[l, i, j] > 0):  # if element is room find neighbours
                    neighbours[69] = l + blueprint[0, 1]  # floor number

                    ### check on floor
                    # current room
                    neighbours[22] = floor_plan[l, i, j]

                    ## check diagonal, direct or indirect
                    # check upper left
                    if -1 < i - 1 < r and -1 < j - 1 < c:
                        if floor_plan[l, i - 1, j - 1] == 0:
                            if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l, i - 2, j - 1] > 0:
                                neighbours[46] = floor_plan[l, i - 2, j - 1]
                            if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l, i - 1, j - 2] > 0:
                                neighbours[45] = floor_plan[l, i - 1, j - 2]
                        elif floor_plan[l, i - 1, j - 1] > 0:
                            neighbours[30] = floor_plan[l, i - 1, j - 1]

                    # check upper right
                    if -1 < i - 1 < r and -1 < j + 1 < c:
                        if floor_plan[l, i - 1, j + 1] == 0:
                            if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l, i - 2, j + 1] > 0:
                                neighbours[36] = floor_plan[l, i - 2, j + 1]
                            if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l, i - 1, j + 2] > 0:
                                neighbours[37] = floor_plan[l, i - 1, j + 2]
                        elif floor_plan[l, i - 1, j + 1] > 0:
                            neighbours[24] = floor_plan[l, i - 1, j + 1]

                    # check lower right
                    if -1 < i + 1 < r and -1 < j + 1 < c:
                        if floor_plan[l, i + 1, j + 1] == 0:
                            if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l, i + 2, j + 1] > 0:
                                neighbours[40] = floor_plan[l, i + 2, j + 1]
```

```
98                          if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l, i
                             ↪  + 1, j + 2] > 0:
99                              neighbours[39] = floor_plan[l, i + 1, j + 2]
100                     elif floor_plan[l, i + 1, j + 1] > 0:
101                         neighbours[26] = floor_plan[l, i + 1, j + 1]
102
103                 # check lower left
104                 if -1 < i + 1 < r and -1 < j - 1 < c:
105                     if floor_plan[l, i + 1, j - 1] == 0:
106                         if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l, i
                             ↪  + 2, j - 1] > 0:
107                             neighbours[42] = floor_plan[l, i + 2, j - 1]
108                         if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l, i
                             ↪  + 1, j - 2] > 0:
109                             neighbours[43] = floor_plan[l, i - 1, j - 2]
110                     elif floor_plan[l, i + 1, j - 1] > 0:
111                         neighbours[28] = floor_plan[l, i + 1, j - 1]
112
113                 ## check horizontal/vertical, direct, second-direct or indirect
114                 # check upper middle
115                 if -1 < i - 1 < r and -1 < j < c:
116                     if floor_plan[l, i - 1, j] == 0:
117                         if -1 < i - 2 < r and -1 < j < c and floor_plan[l, i - 2,
                             ↪  j] > 0:
118                             neighbours[35] = floor_plan[l, i - 2, j]
119                         if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l, i
                             ↪  - 2, j - 1] > 0:
120                             neighbours[46] = floor_plan[l, i - 2, j - 1]
121                         if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l, i
                             ↪  - 2, j + 1] > 0:
122                             neighbours[36] = floor_plan[l, i - 2, j + 1]
123                     elif floor_plan[l, i - 1, j] > 0:
124                         neighbours[23] = floor_plan[l, i - 1, j]
125                         if -1 < i - 2 < r and -1 < j < c and floor_plan[l, i - 2,
                             ↪  j] > 0:
126                             neighbours[31] = floor_plan[l, i - 2, j]
127
128                 # check middle right
129                 if -1 < i < r and -1 < j + 1 < c:
130                     if floor_plan[l, i, j + 1] == 0:
131                         if -1 < i < r and -1 < j + 2 < c and floor_plan[l, i, j +
                             ↪  2] > 0:
132                             neighbours[38] = floor_plan[l, i, j + 2]
133                         if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l, i
                             ↪  - 1, j + 2] > 0:
134                             neighbours[37] = floor_plan[l, i - 1, j + 2]
135                         if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l, i
                             ↪  + 1, j + 2] > 0:
136                             neighbours[39] = floor_plan[l, i + 1, j + 2]
137                     elif floor_plan[l, i, j + 1] > 0:
138                         neighbours[25] = floor_plan[l, i, j + 1]
139                         if -1 < i < r and -1 < j + 2 < c and floor_plan[l, i, j +
                             ↪  2] > 0:
140                             neighbours[32] = floor_plan[l, i, j + 2]
141
142                 # check lower middle
```

```
143                     if -1 < i + 1 < r and -1 < j < c:
144                         if floor_plan[l, i + 1, j] == 0:
145                             if -1 < i + 2 < r and -1 < j < c and floor_plan[l, i + 2,
                                ↪  j] > 0:
146                                 neighbours[41] = floor_plan[l, i + 2, j]
147                             if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l, i
                                ↪  + 2, j + 1] > 0:
148                                 neighbours[40] = floor_plan[l, i + 2, j + 1]
149                             if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l, i
                                ↪  + 2, j - 1] > 0:
150                                 neighbours[42] = floor_plan[l, i + 2, j - 1]
151                         elif floor_plan[l, i + 1, j] > 0:
152                             neighbours[27] = floor_plan[l, i + 1, j]
153                             if -1 < i + 2 < r and -1 < j < c and floor_plan[l, i + 2,
                                ↪  j] > 0:
154                                 neighbours[33] = floor_plan[l, i + 2, j]
155
156                     # check middle right
157                     if -1 < i < r and -1 < j - 1 < c:
158                         if floor_plan[l, i, j - 1] == 0:
159                             if -1 < i < r and -1 < j - 2 < c and floor_plan[l, i, j -
                                ↪  2] > 0:
160                                 neighbours[44] = floor_plan[l, i, j - 2]
161                             if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l, i
                                ↪  + 1, j - 2] > 0:
162                                 neighbours[43] = floor_plan[l, i - 1, j - 2]
163                             if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l, i
                                ↪  - 1, j - 2] > 0:
164                                 neighbours[45] = floor_plan[l, i - 1, j - 2]
165                         elif floor_plan[l, i, j - 1] > 0:
166                             neighbours[29] = floor_plan[l, i, j - 1]
167                             if -1 < i < r and -1 < j - 2 < c and floor_plan[l, i, j -
                                ↪  2] > 0:
168                                 neighbours[34] = floor_plan[l, i, j - 2]
169
170
171                     ### check on floor below
172                     if -1 < l - 1 < m:
173                         # above current room
174                         if -1 < i < r and -1 < j < c and floor_plan[l - 1, i, j] > 0:
175                             neighbours[1] = floor_plan[l - 1, i, j]
176
177                         ## check diagonal, direct or indirect
178                         # check upper left
179                         if -1 < i - 1 < r and -1 < j - 1 < c:
180                             if floor_plan[l - 1, i - 1, j - 1] == 0:
181                                 if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l
                                    ↪  - 1, i - 2, j - 1] > 0:
182                                     neighbours[21] = floor_plan[l - 1, i - 2, j - 1]
183                                 if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l
                                    ↪  - 1, i - 1, j - 2] > 0:
184                                     neighbours[20] = floor_plan[l - 1, i - 1, j - 2]
185                             elif floor_plan[l - 1, i - 1, j - 1] > 0:
186                                 neighbours[9] = floor_plan[l - 1, i - 1, j - 1]
187
188                         # check upper right
```

```
                if -1 < i - 1 < r and -1 < j + 1 < c:
                    if floor_plan[l - 1, i - 1, j + 1] == 0:
                        if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l
                         ↪  - 1, i - 2, j + 1] > 0:
                            neighbours[11] = floor_plan[l - 1, i - 2, j + 1]
                        if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l
                         ↪  - 1, i - 1, j + 2] > 0:
                            neighbours[12] = floor_plan[l - 1, i - 1, j + 2]
                    elif floor_plan[l - 1, i - 1, j + 1] > 0:
                        neighbours[3] = floor_plan[l - 1, i - 1, j + 1]

                # check lower right
                if -1 < i + 1 < r and -1 < j + 1 < c:
                    if floor_plan[l - 1, i + 1, j + 1] == 0:
                        if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l
                         ↪  - 1, i + 2, j + 1] > 0:
                            neighbours[15] = floor_plan[l - 1, i + 2, j + 1]
                        if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l
                         ↪  - 1, i + 1, j + 2] > 0:
                            neighbours[14] = floor_plan[l - 1, i + 1, j + 2]
                    elif floor_plan[l - 1, i + 1, j + 1] > 0:
                        neighbours[5] = floor_plan[l - 1, i + 1, j + 1]

                # check lower left
                if -1 < i + 1 < r and -1 < j - 1 < c:
                    if floor_plan[l - 1, i + 1, j - 1] == 0:
                        if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l
                         ↪  - 1, i + 2, j - 1] > 0:
                            neighbours[17] = floor_plan[l - 1, i + 2, j - 1]
                        if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l
                         ↪  - 1, i + 1, j - 2] > 0:
                            neighbours[18] = floor_plan[l - 1, i - 1, j - 2]
                    elif floor_plan[l - 1, i + 1, j - 1] > 0:
                        neighbours[7] = floor_plan[l - 1, i + 1, j - 1]

                ## check horizontal/vertical, direct, second direct or
                 ↪  indirect
                # check upper middle
                if -1 < i - 1 < r and -1 < j < c:
                    if floor_plan[l - 1, i - 1, j] == 0:
                        if -1 < i - 2 < r and -1 < j < c and floor_plan[l -
                         ↪  1, i - 2, j] > 0:
                            neighbours[10] = floor_plan[l - 1, i - 2, j]
                        if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l
                         ↪  - 1, i - 2, j - 1] > 0:
                            neighbours[21] = floor_plan[l - 1, i - 2, j - 1]
                        if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l
                         ↪  - 1, i - 2, j + 1] > 0:
                            neighbours[11] = floor_plan[l - 1, i - 2, j + 1]
                    elif floor_plan[l - 1, i - 1, j] > 0:
                        neighbours[2] = floor_plan[l - 1, i - 1, j]

                # check middle right
                if -1 < i < r and -1 < j + 1 < c:
                    if floor_plan[l - 1, i, j + 1] == 0:
```

```
234                             if -1 < i < r and -1 < j + 2 < c and floor_plan[l -
                                ↪  1, i, j + 2] > 0:
235                                 neighbours[13] = floor_plan[l - 1, i, j + 2]
236                             if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l
                                ↪  - 1, i - 1, j + 2] > 0:
237                                 neighbours[12] = floor_plan[l - 1, i - 1, j + 2]
238                             if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l
                                ↪  - 1, i + 1, j + 2] > 0:
239                                 neighbours[14] = floor_plan[l - 1, i + 1, j + 2]
240                         elif floor_plan[l - 1, i, j + 1] > 0:
241                             neighbours[4] = floor_plan[l - 1, i, j + 1]
242
243                     # check lower middle
244                     if -1 < i + 1 < r and -1 < j < c:
245                         if floor_plan[l - 1, i + 1, j] == 0:
246                             if -1 < i + 2 < r and -1 < j < c and floor_plan[l -
                                ↪  1, i + 2, j] > 0:
247                                 neighbours[16] = floor_plan[l - 1, i + 2, j]
248                             if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l
                                ↪  - 1, i + 2, j + 1] > 0:
249                                 neighbours[15] = floor_plan[l - 1, i + 2, j + 1]
250                             if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l
                                ↪  - 1, i + 2, j - 1] > 0:
251                                 neighbours[17] = floor_plan[l - 1, i + 2, j - 1]
252                         elif floor_plan[l - 1, i + 1, j] > 0:
253                             neighbours[6] = floor_plan[l - 1, i + 1, j]
254
255                     # check middle right
256                     if -1 < i < r and -1 < j - 1 < c:
257                         if floor_plan[l - 1, i, j - 1] == 0:
258                             if -1 < i < r and -1 < j - 2 < c and floor_plan[l -
                                ↪  1, i, j - 2] > 0:
259                                 neighbours[19] = floor_plan[l - 1, i, j - 2]
260                             if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l
                                ↪  - 1, i + 1, j - 2] > 0:
261                                 neighbours[18] = floor_plan[l - 1, i - 1, j - 2]
262                             if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l
                                ↪  - 1, i - 1, j - 2] > 0:
263                                 neighbours[20] = floor_plan[l - 1, i - 1, j - 2]
264                         elif floor_plan[l - 1, i, j - 1] > 0:
265                             neighbours[8] = floor_plan[l - 1, i, j - 1]
266
267                 ### check on floor above
268                 if -1 < l + 1 < m:
269                     # above current room
270                     if -1 < i < r and -1 < j < c and floor_plan[l + 1, i, j] > 0:
271                         neighbours[47] = floor_plan[l + 1, i, j]
272
273                     ## check diagonal, direct or indirect
274                     # check upper left
275                     if -1 < i - 1 < r and -1 < j - 1 < c:
276                         if floor_plan[l + 1, i - 1, j - 1] == 0:
277                             if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l
                                ↪  + 1, i - 2, j - 1] > 0:
278                                 neighbours[67] = floor_plan[l + 1, i - 2, j - 1]
```

```
279                         if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l
                             ↪  + 1, i - 1, j - 2] > 0:
280                             neighbours[66] = floor_plan[l + 1, i - 1, j - 2]
281                     elif floor_plan[l + 1, i - 1, j - 1] > 0:
282                         neighbours[55] = floor_plan[l + 1, i - 1, j - 1]
283
284                 # check upper right
285                 if -1 < i - 1 < r and -1 < j + 1 < c:
286                     if floor_plan[l + 1, i - 1, j + 1] == 0:
287                         if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l
                             ↪  + 1, i - 2, j + 1] > 0:
288                             neighbours[57] = floor_plan[l + 1, i - 2, j + 1]
289                         if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l
                             ↪  + 1, i - 1, j + 2] > 0:
290                             neighbours[58] = floor_plan[l + 1, i - 1, j + 2]
291                     elif floor_plan[l + 1, i - 1, j + 1] > 0:
292                         neighbours[49] = floor_plan[l + 1, i - 1, j + 1]
293
294                 # check lower right
295                 if -1 < i + 1 < r and -1 < j + 1 < c:
296                     if floor_plan[l + 1, i + 1, j + 1] == 0:
297                         if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l
                             ↪  + 1, i + 2, j + 1] > 0:
298                             neighbours[61] = floor_plan[l + 1, i + 2, j + 1]
299                         if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l
                             ↪  + 1, i + 1, j + 2] > 0:
300                             neighbours[60] = floor_plan[l + 1, i + 1, j + 2]
301                     elif floor_plan[l + 1, i + 1, j + 1] > 0:
302                         neighbours[51] = floor_plan[l + 1, i + 1, j + 1]
303
304                 # check lower left
305                 if -1 < i + 1 < r and -1 < j - 1 < c:
306                     if floor_plan[l + 1, i + 1, j - 1] == 0:
307                         if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l
                             ↪  + 1, i + 2, j - 1] > 0:
308                             neighbours[63] = floor_plan[l + 1, i + 2, j - 1]
309                         if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l
                             ↪  + 1, i + 1, j - 2] > 0:
310                             neighbours[64] = floor_plan[l + 1, i - 1, j - 2]
311                     elif floor_plan[l + 1, i + 1, j - 1] > 0:
312                         neighbours[53] = floor_plan[l + 1, i + 1, j - 1]
313
314                 ## check horizontal/vertical, direct, second direct or
                     ↪  indirect
315                 # check upper middle
316                 if -1 < i - 1 < r and -1 < j < c:
317                     if floor_plan[l + 1, i - 1, j] == 0:
318                         if -1 < i - 2 < r and -1 < j < c and floor_plan[l +
                             ↪  1, i - 2, j] > 0:
319                             neighbours[56] = floor_plan[l + 1, i - 2, j]
320                         if -1 < i - 2 < r and -1 < j - 1 < c and floor_plan[l
                             ↪  + 1, i - 2, j - 1] > 0:
321                             neighbours[67] = floor_plan[l + 1, i - 2, j - 1]
322                         if -1 < i - 2 < r and -1 < j + 1 < c and floor_plan[l
                             ↪  + 1, i - 2, j + 1] > 0:
323                             neighbours[57] = floor_plan[l + 1, i - 2, j + 1]
```

```
                        elif floor_plan[l + 1, i - 1, j] > 0:
                            neighbours[48] = floor_plan[l + 1, i - 1, j]

                    # check middle right
                    if -1 < i < r and -1 < j + 1 < c:
                        if floor_plan[l + 1, i, j + 1] == 0:
                            if -1 < i < r and -1 < j + 2 < c and floor_plan[l +
                             ↪  1, i, j + 2] > 0:
                                neighbours[59] = floor_plan[l + 1, i, j + 2]
                            if -1 < i - 1 < r and -1 < j + 2 < c and floor_plan[l
                             ↪  + 1, i - 1, j + 2] > 0:
                                neighbours[58] = floor_plan[l + 1, i - 1, j + 2]
                            if -1 < i + 1 < r and -1 < j + 2 < c and floor_plan[l
                             ↪  + 1, i + 1, j + 2] > 0:
                                neighbours[60] = floor_plan[l + 1, i + 1, j + 2]
                        elif floor_plan[l + 1, i, j + 1] > 0:
                            neighbours[50] = floor_plan[l + 1, i, j + 1]

                    # check lower middle
                    if -1 < i + 1 < r and -1 < j < c:
                        if floor_plan[l + 1, i + 1, j] == 0:
                            if -1 < i + 2 < r and -1 < j < c and floor_plan[l +
                             ↪  1, i + 2, j] > 0:
                                neighbours[62] = floor_plan[l + 1, i + 2, j]
                            if -1 < i + 2 < r and -1 < j + 1 < c and floor_plan[l
                             ↪  + 1, i + 2, j + 1] > 0:
                                neighbours[61] = floor_plan[l + 1, i + 2, j + 1]
                            if -1 < i + 2 < r and -1 < j - 1 < c and floor_plan[l
                             ↪  + 1, i + 2, j - 1] > 0:
                                neighbours[63] = floor_plan[l + 1, i + 2, j - 1]
                        elif floor_plan[l + 1, i + 1, j] > 0:
                            neighbours[52] = floor_plan[l + 1, i + 1, j]

                    # check middle right
                    if -1 < i < r and -1 < j - 1 < c:
                        if floor_plan[l + 1, i, j - 1] == 0:
                            if -1 < i < r and -1 < j - 2 < c and floor_plan[l +
                             ↪  1, i, j - 2] > 0:
                                neighbours[65] = floor_plan[l + 1, i, j - 2]
                            if -1 < i + 1 < r and -1 < j - 2 < c and floor_plan[l
                             ↪  + 1, i + 1, j - 2] > 0:
                                neighbours[64] = floor_plan[l + 1, i - 1, j - 2]
                            if -1 < i - 1 < r and -1 < j - 2 < c and floor_plan[l
                             ↪  + 1, i - 1, j - 2] > 0:
                                neighbours[66] = floor_plan[l + 1, i - 1, j - 2]
                        elif floor_plan[l + 1, i, j - 1] > 0:
                            neighbours[54] = floor_plan[l + 1, i, j - 1]


                ### check on two floors below
                if -1 < l - 2 < m:
                    # check middle middle
                    if -1 < i < r and -1 < j < c and floor_plan[l - 2, i, j] > 0:
                        neighbours[0] = floor_plan[l - 2, i, j]

                ### check on two floors above
```

```
                    if -1 < l + 2 < m:
                        # check middle middle
                        if -1 < i < r and -1 < j < c and floor_plan[l + 2, i, j] > 0:
                            neighbours[68] = floor_plan[l + 2, i, j]

                    # neighbours.insert(0, l + 6)  # register current floor at index
                     ↪  0

                    rooms.append(neighbours)  # save list

    rooms = np.array(rooms)  # convert list of lists to numpy array
    return rooms
```

A.3.2. Class Room

```
#BAP Q1 2022-2023 Subgroup 2
#Room classes Version 5
#Coupling between bed, bedsense, organization and the room's neighbours
#Enables us to see which room is connected to which bedsense and what the room's
 ↪  neighbours are.
#Version 2022/12/09 15:00

import csv
import numpy as np

import Map_Matrix_5


class Room:
    #Define the variables which are directly given as input to the instances from the
     ↪  file
    def __init__(self, roomnumber, bedsensenumber, organization):
        self.roomnumber = roomnumber
        self.bedsense = bedsensenumber
        self.organization = organization

    #Define the variable floor which contains the floor of the room
    def set_floor(self, floor):
        self.floor = floor

    #Define the variable map which contains the map with the neighbours of the room
    def set_neighbours(self, neighbour):
        self.map = neighbour

    # Define the variable bedsenses which contains the map with the neighbours of the
     ↪  room in bedsense numbers
    def set_neighbourbedsenses(self, bedsenses):
        self.bedsenses = bedsenses

    # Define the variable values which contains the map with the values received by
     ↪  the neighbours of the room
    def set_neighbourvalues(self, values):
        self.values = values


def import_rooms(Directory, map):
```

```
    #This function has as input the directory to the coupling file and the directory to the map file
    #Function that creates an instance for each room, containing the roomnumber, bedsensenumber and organization
    #Also the variable map is defined for every instance based on the list of maps coming from Map_Matrix
    #These instances are zipped in a dictionary with as key the bedsensenumber
    #As output has this function the dictionary and the complete layout of the building

    maps = Map_Matrix_5.map_matrix(map)
    layout = Map_Matrix_5.floor_plan
    all_rooms = []
    i = 0
    with open(Directory) as d:
        reader = csv.reader(d)
        data = list(reader)
        data = data[1:]
        bedsensenumbers = np.array(data)[:, 1]
        for row in data:
            all_rooms.append(Room(row[0], row[1], row[2]))
            for neighbour_map in maps:
                if neighbour_map[22] == float(row[0]):
                    all_rooms[i].set_floor(neighbour_map[69])
                    all_rooms[i].set_neighbours(np.delete(neighbour_map, 69))
            i += 1
        all_rooms_dict = dict(zip(bedsensenumbers, all_rooms))
    return all_rooms_dict, layout
```

A.3.3. Neighbour Values

```
import math
import numpy as np


def Neighbour_values(row, bed, neighbours, bed_bedsense, bedsenses):
    #This function has as input a row of the grouped data, a Room instance, the neighbour matrix of the instance, the coupling between bed and bedsense and the numpy array containing all bedsenseID's
    #Function that places the neighbour bedsenseID's in a new matrix at the same place as its corresponding roomnumber
    #Is then places the value received by a bedsense at the same place in a new matrix as its bedsenseID was
    #Both these matrices are stored in the instance
    #This function has no outputs

    neighbour_bedsenses = neighbours.copy() #create a copy of the neighbours array
    for o in range(len(neighbours)):
        if not math.isnan(neighbours[o]):  #check if neighbours[o] is existing
            index = bed_bedsense[bed_bedsense['Bed'].isin([int(neighbours[o])])].to_numpy()[0, 1]  # Find the rows in dataframe belonging to the neighbours
            neighbour_bedsenses[o] = index  # Add bedsense number to numpy array at the same location as the roomnumber was
    bed.set_neighbourbedsenses(neighbour_bedsenses)  # Add matrix to class instance bed
```

```

    neighbour_values = neighbours.copy() #create a copy of the neighbours array
    for j in range(len(neighbour_bedsenses)):
            bed_index = np.where(bedsenses == neighbour_bedsenses[j]) #find index of
             ↪ neighbour_bedsenses[j] in bedsenses array, which corresponds to index
             ↪ of rssi received by bedsense in row
            if np.any(bed_index[0]): #check if bedsense collected any data at all and
             ↪ thus is existing in row
                neighbour_values[j] = row[bed_index[0]] #add value to new matrix at
                 ↪ same location as bedsense number was
    bed.set_neighbourvalues(neighbour_values) #add matrix to class

    return
```

A.4. Neighbour Comparison

A.4.1. Neighbour Comparison

```
# BAP Q1 2022-2023 Subgroup 2
# Neighbour comparison
# Compare measured neighbour RSSI values with each other against the expected
 ↪ behaviour of neighbour RSSI's
# Version 4.0 (works with (RSSI-filled) array from Map_Matrix_5_0)
# 2022/12/09 10:45

import numpy as np


def neigh_comp(neighbour_rssi):
    neighbour_rssi = neighbour_rssi[:69]  # make sure floor number is removed
    neighbour_rssi = [-200 if 0 < i != np.nan else i for i in neighbour_rssi]  # if
     ↪ value is roomnumber, room received signal weakly, say -200dB

    dict_neighbours = {
    0: [[1, -0.5479832539398328], [10, 0.6466971013818766], [11, 0.7220582450858175],
     ↪ [12, 0.7220582450858175],
         [13, 0.6466971013818766], [14, 0.7220582450858175], [15,
          ↪ 0.7220582450858175], [16, 0.6466971013818766],
         [17, 0.7220582450858175], [18, 0.7220582450858175], [19,
          ↪ 0.6466971013818766], [20, 0.7220582450858175],
         [21, 0.7220582450858175], [23, -0.5479832539398328], [24,
          ↪ -0.5479832539398328], [25, -0.5479832539398328],
         [26, -0.5479832539398328], [27, -0.5479832539398328], [28,
          ↪ -0.5479832539398328], [29, -0.5479832539398328],
         [30, -0.5479832539398328], [31, 0.6466971013818766], [32,
          ↪ 0.6466971013818766], [33, 0.6466971013818766],
         [34, 0.6466971013818766], [35, 0.6466971013818766], [36,
          ↪ 0.6466971013818766], [37, 0.6466971013818766],
         [38, 0.6466971013818766], [39, 0.6466971013818766], [40,
          ↪ 0.6466971013818766], [41, 0.6466971013818766],
         [42, 0.6466971013818766], [43, 0.6466971013818766], [44,
          ↪ 0.6466971013818766], [45, 0.6466971013818766],
         [46, 0.6466971013818766]],
     1: [[2, 0.5479832539398328], [3, 0.5479832539398328], [4, 0.5479832539398328],
      ↪ [5, 0.5479832539398328],
```

```
26          [6, 0.5479832539398328], [7, 0.5479832539398328], [8, 0.5479832539398328],
            ↪  [9, 0.5479832539398328],
27          [10, 0.6904005902711063], [11, 0.7610035763163452], [12,
            ↪  0.7610035763163452], [13, 0.6904005902711063],
28          [14, 0.7610035763163452], [15, 0.7610035763163452], [16,
            ↪  0.6904005902711063], [17, 0.7610035763163452],
29          [18, 0.7610035763163452], [19, 0.6904005902711063], [20,
            ↪  0.7610035763163452], [21, 0.7610035763163452],
30          [31, 0.6904005902711063], [32, 0.6904005902711063], [33,
            ↪  0.6904005902711063], [34, 0.6904005902711063],
31          [35, 0.6904005902711063], [36, 0.6904005902711063], [37,
            ↪  0.6904005902711063], [38, 0.6904005902711063],
32          [39, 0.6904005902711063], [40, 0.6904005902711063], [41,
            ↪  0.6904005902711063], [42, 0.6904005902711063],
33          [43, 0.6904005902711063], [44, 0.6904005902711063], [45,
            ↪  0.6904005902711063], [46, 0.6904005902711063]],
34      2: [[10, 0.6466971013818766], [11, 0.7220582450858175], [12,
        ↪  0.7220582450858175], [13, 0.6466971013818766],
35          [19, 0.6466971013818766], [20, 0.7220582450858175], [21,
            ↪  0.7220582450858175], [31, 0.6466971013818766],
36          [32, 0.6466971013818766], [34, 0.6466971013818766]],
37      3: [[10, 0.6466971013818766], [11, 0.7220582450858175], [12,
        ↪  0.7220582450858175], [13, 0.6466971013818766],
38          [31, 0.6466971013818766], [32, 0.6466971013818766]],
39      4: [[10, 0.6466971013818766], [11, 0.7220582450858175], [12,
        ↪  0.7220582450858175], [13, 0.6466971013818766],
40          [14, 0.7220582450858175], [15, 0.7220582450858175], [16,
            ↪  0.6466971013818766], [31, 0.6466971013818766],
41          [32, 0.6466971013818766], [33, 0.6466971013818766]],
42      5: [[13, 0.6466971013818766], [14, 0.7220582450858175], [15,
        ↪  0.7220582450858175], [16, 0.6466971013818766],
43          [32, 0.6466971013818766], [33, 0.6466971013818766]],
44      6: [[13, 0.6466971013818766], [14, 0.7220582450858175], [15,
        ↪  0.7220582450858175], [16, 0.6466971013818766],
45          [17, 0.7220582450858175], [18, 0.7220582450858175], [19,
            ↪  0.6466971013818766], [32, 0.6466971013818766],
46          [33, 0.6466971013818766], [34, 0.6466971013818766]],
47      7: [[16, 0.6466971013818766], [17, 0.7220582450858175], [18,
        ↪  0.7220582450858175], [19, 0.6466971013818766],
48          [33, 0.6466971013818766], [34, 0.6466971013818766]],
49      8: [[16, 0.6466971013818766], [17, 0.7220582450858175], [18,
        ↪  0.7220582450858175], [19, 0.6466971013818766],
50          [20, 0.7220582450858175], [21, 0.7220582450858175], [31,
            ↪  0.6466971013818766], [33, 0.6466971013818766],
51          [34, 0.6466971013818766]], 9: [[10, 0.6466971013818766], [31,
            ↪  0.6466971013818766], [34, 0.6466971013818766]],
52      10: [[11, 0.5841603716167574], [12, 0.5841603716167574], [20,
        ↪  0.5841603716167574], [21, 0.5841603716167574],
53           [23, -0.6904005902711063], [24, -0.6904005902711063], [25,
             ↪  -0.6904005902711063], [29, -0.6904005902711063],
54           [30, -0.6904005902711063]],
55      11: [[13, -0.5841603716167574], [23, -0.7610035763163452], [24,
        ↪  -0.7610035763163452], [25, -0.7610035763163452],
56           [35, -0.5841603716167574], [36, -0.5841603716167574], [37,
             ↪  -0.5841603716167574], [38, -0.5841603716167574]],
```

```
57      12: [[13, -0.5841603716167574], [23, -0.7610035763163452], [24,
        ↪  -0.7610035763163452], [25, -0.7610035763163452],
58           [35, -0.5841603716167574], [36, -0.5841603716167574], [37,
             ↪  -0.5841603716167574], [38, -0.5841603716167574]],
59      13: [[14, 0.5841603716167574], [15, 0.5841603716167574], [23,
        ↪  -0.6904005902711063], [24, -0.6904005902711063],
60           [25, -0.6904005902711063], [26, -0.6904005902711063], [27,
             ↪  -0.6904005902711063]],
61      14: [[16, -0.5841603716167574], [25, -0.7610035763163452], [26,
        ↪  -0.7610035763163452], [27, -0.7610035763163452],
62           [38, -0.5841603716167574], [39, -0.5841603716167574], [40,
             ↪  -0.5841603716167574], [41, -0.5841603716167574]],
63      15: [[16, -0.5841603716167574], [25, -0.7610035763163452], [26,
        ↪  -0.7610035763163452], [27, -0.7610035763163452],
64           [38, -0.5841603716167574], [39, -0.5841603716167574], [40,
             ↪  -0.5841603716167574], [41, -0.5841603716167574]],
65      16: [[17, 0.5841603716167574], [18, 0.5841603716167574], [25,
        ↪  -0.6904005902711063], [26, -0.6904005902711063],
66           [27, -0.6904005902711063], [28, -0.6904005902711063], [29,
             ↪  -0.6904005902711063]],
67      17: [[19, -0.5841603716167574], [27, -0.7610035763163452], [28,
        ↪  -0.7610035763163452], [29, -0.7610035763163452],
68           [41, -0.5841603716167574], [42, -0.5841603716167574], [43,
             ↪  -0.5841603716167574], [44, -0.5841603716167574]],
69      18: [[19, -0.5841603716167574], [27, -0.7610035763163452], [28,
        ↪  -0.7610035763163452], [29, -0.7610035763163452],
70           [41, -0.5841603716167574], [42, -0.5841603716167574], [43,
             ↪  -0.5841603716167574], [44, -0.5841603716167574]],
71      19: [[20, 0.5841603716167574], [21, 0.5841603716167574], [23,
        ↪  -0.6904005902711063], [27, -0.6904005902711063],
72           [28, -0.6904005902711063], [29, -0.6904005902711063], [30,
             ↪  -0.6904005902711063]],
73      20: [[23, -0.7610035763163452], [29, -0.7610035763163452], [30,
        ↪  -0.7610035763163452], [35, -0.5841603716167574],
74           [44, -0.5841603716167574], [45, -0.5841603716167574], [46,
             ↪  -0.5841603716167574]],
75      21: [[23, -0.7610035763163452], [29, -0.7610035763163452], [30,
        ↪  -0.7610035763163452], [35, -0.5841603716167574],
76           [44, -0.5841603716167574], [45, -0.5841603716167574], [46,
             ↪  -0.5841603716167574]],
77      22: [[23, 0.7610035763163446], [24, 0.7610035763163446], [25,
        ↪  0.7610035763163446], [26, 0.7610035763163446],
78           [27, 0.7610035763163446], [28, 0.7610035763163446], [29,
             ↪  0.7610035763163446], [30, 0.7610035763163446],
79           [68, 0.7967594858586314]],
80      23: [[35, 0.6904005902711063], [36, 0.6904005902711063], [37,
        ↪  0.6904005902711063], [38, 0.6904005902711063],
81           [44, 0.6904005902711063], [45, 0.6904005902711063], [46,
             ↪  0.6904005902711063], [56, 0.6904005902711063],
82           [57, 0.7610035763163452], [58, 0.7610035763163452], [59,
             ↪  0.6904005902711063], [65, 0.6904005902711063],
83           [66, 0.7610035763163452], [67, 0.7610035763163452], [68,
             ↪  0.5479832539398328]],
84      24: [[35, 0.6904005902711063], [36, 0.6904005902711063], [37,
        ↪  0.6904005902711063], [38, 0.6904005902711063],
```

```
85          [56, 0.6904005902711063], [57, 0.7610035763163452], [58,
              ↪ 0.7610035763163452], [59, 0.6904005902711063],
86          [68, 0.5479832539398328]],
87      25: [[35, 0.6904005902711063], [36, 0.6904005902711063], [37,
          ↪ 0.6904005902711063], [38, 0.6904005902711063],
88          [39, 0.6904005902711063], [40, 0.6904005902711063], [41,
              ↪ 0.6904005902711063], [56, 0.6904005902711063],
89          [57, 0.7610035763163452], [58, 0.7610035763163452], [59,
              ↪ 0.6904005902711063], [60, 0.7610035763163452],
90          [61, 0.7610035763163452], [62, 0.6904005902711063], [68,
              ↪ 0.5479832539398328]],
91      26: [[38, 0.6904005902711063], [39, 0.6904005902711063], [40,
          ↪ 0.6904005902711063], [41, 0.6904005902711063],
92          [59, 0.6904005902711063], [60, 0.7610035763163452], [61,
              ↪ 0.7610035763163452], [62, 0.6904005902711063],
93          [68, 0.5479832539398328]],
94      27: [[38, 0.6904005902711063], [39, 0.6904005902711063], [40,
          ↪ 0.6904005902711063], [41, 0.6904005902711063],
95          [42, 0.6904005902711063], [43, 0.6904005902711063], [44,
              ↪ 0.6904005902711063], [59, 0.6904005902711063],
96          [60, 0.7610035763163452], [61, 0.7610035763163452], [62,
              ↪ 0.6904005902711063], [63, 0.7610035763163452],
97          [64, 0.7610035763163452], [65, 0.6904005902711063], [68,
              ↪ 0.5479832539398328]],
98      28: [[41, 0.6904005902711063], [42, 0.6904005902711063], [43,
          ↪ 0.6904005902711063], [44, 0.6904005902711063],
99          [62, 0.6904005902711063], [63, 0.7610035763163452], [64,
              ↪ 0.7610035763163452], [65, 0.6904005902711063],
100         [68, 0.5479832539398328]],
101     29: [[35, 0.6904005902711063], [41, 0.6904005902711063], [42,
          ↪ 0.6904005902711063], [43, 0.6904005902711063],
102         [44, 0.6904005902711063], [45, 0.6904005902711063], [46,
              ↪ 0.6904005902711063], [56, 0.6904005902711063],
103         [62, 0.6904005902711063], [63, 0.7610035763163452], [64,
              ↪ 0.7610035763163452], [65, 0.6904005902711063],
104         [66, 0.7610035763163452], [67, 0.7610035763163452], [68,
              ↪ 0.5479832539398328]],
105     30: [[35, 0.6904005902711063], [44, 0.6904005902711063], [45,
          ↪ 0.6904005902711063], [46, 0.6904005902711063],
106         [56, 0.6904005902711063], [65, 0.6904005902711063], [66,
              ↪ 0.7610035763163452], [67, 0.7610035763163452],
107         [68, 0.5479832539398328]],
108     31: [[47, -0.6904005902711063], [48, -0.6466971013818766], [49,
          ↪ -0.6466971013818766], [50, -0.6466971013818766],
109         [54, -0.6466971013818766], [55, -0.6466971013818766], [68,
              ↪ -0.6466971013818766]],
110     32: [[47, -0.6904005902711063], [48, -0.6466971013818766], [49,
          ↪ -0.6466971013818766], [50, -0.6466971013818766],
111         [51, -0.6466971013818766], [52, -0.6466971013818766], [68,
              ↪ -0.6466971013818766]],
112     33: [[47, -0.6904005902711063], [50, -0.6466971013818766], [51,
          ↪ -0.6466971013818766], [52, -0.6466971013818766],
113         [53, -0.6466971013818766], [54, -0.6466971013818766], [68,
              ↪ -0.6466971013818766]],
114     34: [[47, -0.6904005902711063], [48, -0.6466971013818766], [52,
          ↪ -0.6466971013818766], [53, -0.6466971013818766],
```

```
115             [54, -0.6466971013818766], [55, -0.6466971013818766], [68,
                ↪  -0.6466971013818766]],
116         35: [[47, -0.6904005902711063], [57, 0.5841603716167574], [58,
            ↪  0.5841603716167574], [66, 0.5841603716167574],
117             [67, 0.5841603716167574], [68, -0.6466971013818766]],
118         36: [[47, -0.6904005902711063], [57, 0.5841603716167574], [58,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
119         37: [[47, -0.6904005902711063], [57, 0.5841603716167574], [58,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
120         38: [[47, -0.6904005902711063], [57, 0.5841603716167574], [58,
            ↪  0.5841603716167574], [60, 0.5841603716167574],
121             [61, 0.5841603716167574], [68, -0.6466971013818766]],
122         39: [[47, -0.6904005902711063], [60, 0.5841603716167574], [61,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
123         40: [[47, -0.6904005902711063], [60, 0.5841603716167574], [61,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
124         41: [[47, -0.6904005902711063], [60, 0.5841603716167574], [61,
            ↪  0.5841603716167574], [63, 0.5841603716167574],
125             [64, 0.5841603716167574], [68, -0.6466971013818766]],
126         42: [[47, -0.6904005902711063], [63, 0.5841603716167574], [64,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
127         43: [[47, -0.6904005902711063], [63, 0.5841603716167574], [64,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
128         44: [[47, -0.6904005902711063], [63, 0.5841603716167574], [64,
            ↪  0.5841603716167574], [66, 0.5841603716167574],
129             [67, 0.5841603716167574], [68, -0.6466971013818766]],
130         45: [[47, -0.6904005902711063], [66, 0.5841603716167574], [67,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
131         46: [[47, -0.6904005902711063], [66, 0.5841603716167574], [67,
            ↪  0.5841603716167574], [68, -0.6466971013818766]],
132         47: [[48, 0.5479832539398328], [49, 0.5479832539398328], [50,
            ↪  0.5479832539398328], [51, 0.5479832539398328],
133             [52, 0.5479832539398328], [53, 0.5479832539398328], [54,
                ↪  0.5479832539398328], [55, 0.5479832539398328],
134             [56, 0.6904005902711063], [57, 0.7610035763163452], [58,
                ↪  0.7610035763163452], [59, 0.6904005902711063],
135             [60, 0.7610035763163452], [61, 0.7610035763163452], [62,
                ↪  0.6904005902711063], [63, 0.7610035763163452],
136             [64, 0.7610035763163452], [65, 0.6904005902711063], [66,
                ↪  0.7610035763163452], [67, 0.7610035763163452],
137             [68, 0.5479832539398328]],
138         48: [[56, 0.6466971013818766], [57, 0.7220582450858175], [58,
            ↪  0.7220582450858175], [59, 0.6466971013818766],
139             [65, 0.6466971013818766], [66, 0.7220582450858175], [67,
                ↪  0.7220582450858175]],
140         49: [[56, 0.6466971013818766], [57, 0.7220582450858175], [58,
            ↪  0.7220582450858175], [59, 0.6466971013818766]],
141         50: [[56, 0.6466971013818766], [57, 0.7220582450858175], [58,
            ↪  0.7220582450858175], [59, 0.6466971013818766],
142             [60, 0.7220582450858175], [61, 0.7220582450858175], [62,
                ↪  0.6466971013818766]],
143         51: [[59, 0.6466971013818766], [60, 0.7220582450858175], [61,
            ↪  0.7220582450858175], [62, 0.6466971013818766]],
144         52: [[59, 0.6466971013818766], [60, 0.7220582450858175], [61,
            ↪  0.7220582450858175], [62, 0.6466971013818766],
```

```
145             [63, 0.7220582450858175], [64, 0.7220582450858175], [65,
                ↪ 0.6466971013818766]],
146         53: [[62, 0.6466971013818766], [63, 0.7220582450858175], [64,
            ↪ 0.7220582450858175], [65, 0.6466971013818766]],
147         54: [[56, 0.6466971013818766], [62, 0.6466971013818766], [63,
            ↪ 0.7220582450858175], [64, 0.7220582450858175],
148             [65, 0.6466971013818766], [66, 0.7220582450858175], [67,
                ↪ 0.7220582450858175]],
149         55: [[56, 0.6466971013818766], [65, 0.6466971013818766], [66,
            ↪ 0.7220582450858175], [67, 0.7220582450858175]],
150         56: [[57, 0.5841603716167574], [58, 0.5841603716167574], [66,
            ↪ 0.5841603716167574], [67, 0.5841603716167574],
151             [68, -0.6466971013818766]], 57: [[59, -0.5841603716167574], [68,
                ↪ -0.7220582450858175]],
152         58: [[59, -0.5841603716167574], [68, -0.7220582450858175]],
153         59: [[60, 0.5841603716167574], [61, 0.5841603716167574], [68,
            ↪ -0.6466971013818766]],
154         60: [[62, -0.5841603716167574], [68, -0.7220582450858175]],
155         61: [[62, -0.5841603716167574], [68, -0.7220582450858175]],
156         62: [[63, 0.5841603716167574], [64, 0.5841603716167574], [68,
            ↪ -0.6466971013818766]],
157         63: [[65, -0.5841603716167574], [68, -0.7220582450858175]],
158         64: [[65, -0.5841603716167574], [68, -0.7220582450858175]],
159         65: [[66, 0.5841603716167574], [67, 0.5841603716167574], [68,
            ↪ -0.6466971013818766]],
160         66: [[68, -0.7220582450858175]],
161         67: [[68, -0.7220582450858175]],
162         68: []
163     }
164
165     comparison_max = 0
166     comparison = 0
167
168     for i in range(len(dict_neighbours)):  # for all possible rooms
169         if not(np.isnan(neighbour_rssi[i])):  # ... if they exist in current
            ↪ neighbour model
170             for j in range(len(dict_neighbours[i])):  # compare them with the
                ↪ to-be-compared neighbours
171                 if not(np.isnan(neighbour_rssi[dict_neighbours[i][j][0]])):  # ... if
                    ↪ those exist
172                     weight = dict_neighbours[i][j][1]  # appropriate weight for
                        ↪ comparison
173                     comparison_max += abs(weight)  # count for max achievable weight
174
175                     if neighbour_rssi[i] >= neighbour_rssi[dict_neighbours[i][j][0]]:
                        ↪ # compare room...
176                         comparison += weight  # comparison as expected -> associate
                            ↪ appropriate weighted TRUE
177                     else:
178                         comparison -= weight  # comparison not as expected ->
                            ↪ associate appropriate weighted FALSE
179
180     if comparison_max > 0:
181         return comparison/comparison_max  # normalized comparison
182     else:
```

```
183         return 0  # if no neighbours exist or could be compared
```

A.4.2. Comparison Dictionary

```
# BAP Q1 2022-2023 Subgroup 2
# Comparison dictionary
# Create dictionary to specify which neighbours should be compared with which neighbours and what weight
# Version 2.0 (works for Neighbour_Comparison_5_0)
# 2022/12/09 00:00

import numpy as np
import pandas as pd

from Comparison_Weights_4_1 import get_weight


def get_dict_neighbours(data):
    data = open(data)
    df_data = pd.read_csv(data, header=None)
    combi = df_data.to_numpy()  # matrix with what neighbours should be compared to which neighbours

    dict_neigh = {}  # dictionary containing each neighbour complemented with the to-be-compared-to neighbours
    for i in range(1, np.shape(combi)[0] - 1):  # for each neighbour i
        rooms = []  # list of neighbours to-be-compared with this neighbour i
        for j in range(1, np.shape(combi)[1] - 1):  # check which neighbours j should be compared to this neighbour i
            if combi[i, j] == 0:  # if it should be compared
                weight = get_weight(combi[i, -1] + 1, combi[-1, j] + 1)  # get weight of this comparison (roomnumber+1 due to layoutmatrix in get_weight)
                if weight:  # if neighbours are not expected to be compared at equal strength -> compare this neighbour
                    rooms.append([int(combi[-1, j]), weight])  # save that neighbour and corresponding comparison weight

        dict_neigh[int(combi[i, -1])] = rooms  # key for neighbour j with all the to-be-compared neighbours in "rooms"

        print("process = %d%%" % (int(i / (np.shape(combi)[0] - 2) * 100)))  # print process as percentage

    return dict_neigh

```

A.4.3. Comparison Weights

```
# BAP Q1 2022-2023 Subgroup 2
# Compute comparison weights
# Compute weight for a comparison of two points depending on their distance to midpoint using normal distribution for error in RSS
# Version 4.1 (works with Comparison_Dictionary_2_0) (log-normal instead of normal distribution)
```

```python
# 2022/12/09 00:00

import math
import numpy as np
from scipy.stats import norm
from scipy.integrate import quad


def probability(dis1, dis2):  # it is supposed to be dis1 < dis2

    # [P]
    # ^  "
    # |
    # |   "
    # 0 ---"--dis1---_dis2-------> [x]
    # |      " |       |
    # | - - - -*       |
    # |           "    |
    # | - - - - - - - -*
    # |                        "

    s1 = 1
    mean1 = -20 * math.log(dis2, 10)  # bigger distance means smaller power
    var1 = 3
    s2 = 1
    mean2 = -20 * math.log(dis1, 10)  # smaller distance means larger power
    var2 = 3

    prob_wrong = quad(lambda x: norm.pdf(x, s1, loc=mean1, scale=var1) * norm.cdf(x, s2, loc=mean2, scale=var2), -np.inf, np.inf)  # probability dis1 > dis2
    prob_correct = 1 - prob_wrong[0]  # probability dis1 < dis2

    return prob_correct


def get_weight(room1, room2):
    # Lay-out matrix (3-D) (2-below - below - level - above - 2-above)
    # |---|----------------|----------------------|----------------|----|
    # |   |                |       46 35 36       |                |    |
    # |   |    21 10 11    |          31          |    67 56 57    |    |
    # |   | 20 9  2  3  12 | 45    30 23 24    37 | 66 55 47 49 58 |    |
    # | 0 | 19 8  1  4  13 | 44 34 29 22 25 32 38 | 65 54 47 50 59 | 68 |
    # |   | 18 7  6  5  14 | 43    28 27 26    39 | 64 53 52 51 60 |    |
    # |   |    17 16 15    |          33          |    63 62 61    |    |
    # |   |                |       42 41 40       |                |    |
    # |---|----------------|----------------------|----------------|----|

    # Expectation matrix
    # - X             : 0
    # - d, a&b        : 1
    # - bd&ad, bb&aa  : 2
    # - s, bi&ai, i   : 3
    # - by&ay         : 4
    # |----|----------------|---------------|----------------|----|
    # |    |                |     i i i     |                |    |
    # |    |    by bi by    |       s       |    ay ai ay    |    |
```

```
    # |    | by bd bd bd by | i   d d d   i | ay ad ad ad ay |    |
    # | bb | bi bd b  bd bi | i s d X d s i | ai ad a  ad ai | aa |
    # |    | by bd bd bd by | i   d d d   i | ay ad ad ad ay |    |
    # |    |    by bi by    |       s       |    ay ai ay    |    |
    # |    |                |     i i i     |                |    |
    # |----|----------------|---------------|----------------|----|


    layout_matrix = np.array([[[0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 1, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0]],

                              [[0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 22, 11, 12, 0, 0],
                               [0, 21, 10, 3, 4, 13, 0],
                               [0, 20, 9, 2, 5, 14, 0],
                               [0, 19, 8, 7, 6, 15, 0],
                               [0, 0, 18, 17, 16, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0]],

                              [[0, 0, 47, 36, 37, 0, 0],
                               [0, 0, 0, 32, 0, 0, 0],
                               [46, 0, 31, 24, 25, 0, 38],
                               [45, 35, 30, 23, 26, 33, 39],
                               [44, 0, 29, 28, 27, 0, 40],
                               [0, 0, 0, 34, 0, 0, 0],
                               [0, 0, 43, 42, 41, 0, 0]],

                              [[0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 68, 57, 58, 0, 0],
                               [0, 67, 56, 49, 50, 59, 0],
                               [0, 66, 55, 48, 51, 60, 0],
                               [0, 65, 54, 53, 52, 61, 0],
                               [0, 0, 64, 63, 62, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0]],

                              [[0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 69, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0],
                               [0, 0, 0, 0, 0, 0, 0]]])

    expectation_matrix = np.array([[[0, 0, 0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0, 0, 0],
                                    [0, 0, 0, 2, 0, 0, 0],
                                    [0, 0, 0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0, 0, 0],
                                    [0, 0, 0, 0, 0, 0, 0]],
```

```
                                    [[0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 4, 3, 4, 0, 0],
                                     [0, 4, 2, 2, 2, 4, 0],
                                     [0, 3, 2, 1, 2, 3, 0],
                                     [0, 4, 2, 2, 2, 4, 0],
                                     [0, 0, 4, 3, 4, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0]],

                                    [[0, 0, 3, 3, 3, 0, 0],
                                     [0, 0, 0, 3, 0, 0, 0],
                                     [3, 0, 1, 1, 1, 0, 3],
                                     [3, 3, 1, 0, 1, 3, 3],
                                     [3, 0, 1, 1, 1, 0, 3],
                                     [0, 0, 0, 3, 0, 0, 0],
                                     [0, 0, 3, 3, 3, 0, 0]],

                                    [[0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 4, 3, 4, 0, 0],
                                     [0, 4, 2, 2, 2, 4, 0],
                                     [0, 3, 2, 1, 2, 3, 0],
                                     [0, 4, 2, 2, 2, 4, 0],
                                     [0, 0, 4, 3, 4, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0]],

                                    [[0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 2, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 0]]])  # expectation in the
                                      ↪ order of distance

    distances = [1, math.sqrt(2), 1.5, math.sqrt(3.25), 2]  # distance from X at
     ↪ expectation [0, 1, 2, 3, 4] respectively

    exp_room1 = expectation_matrix[np.where(layout_matrix == room1)][0]  # find
     ↪ expectation of neighbour...
    exp_room2 = expectation_matrix[np.where(layout_matrix == room2)][0]  # ...
    # print(room1, exp_room1, room2, exp_room2)

    if exp_room1 < exp_room2:
        return probability(distances[exp_room1], distances[exp_room2])  # prob of
         ↪ room1 is closer/stronger then room2 (is pos)
    elif exp_room1 > exp_room2:
        return -probability(distances[exp_room2], distances[exp_room1])  # prob of
         ↪ room1 is closer/stronger then room2 (is neg)
    else:
        return False
```

A.4.4. The Comparison Table

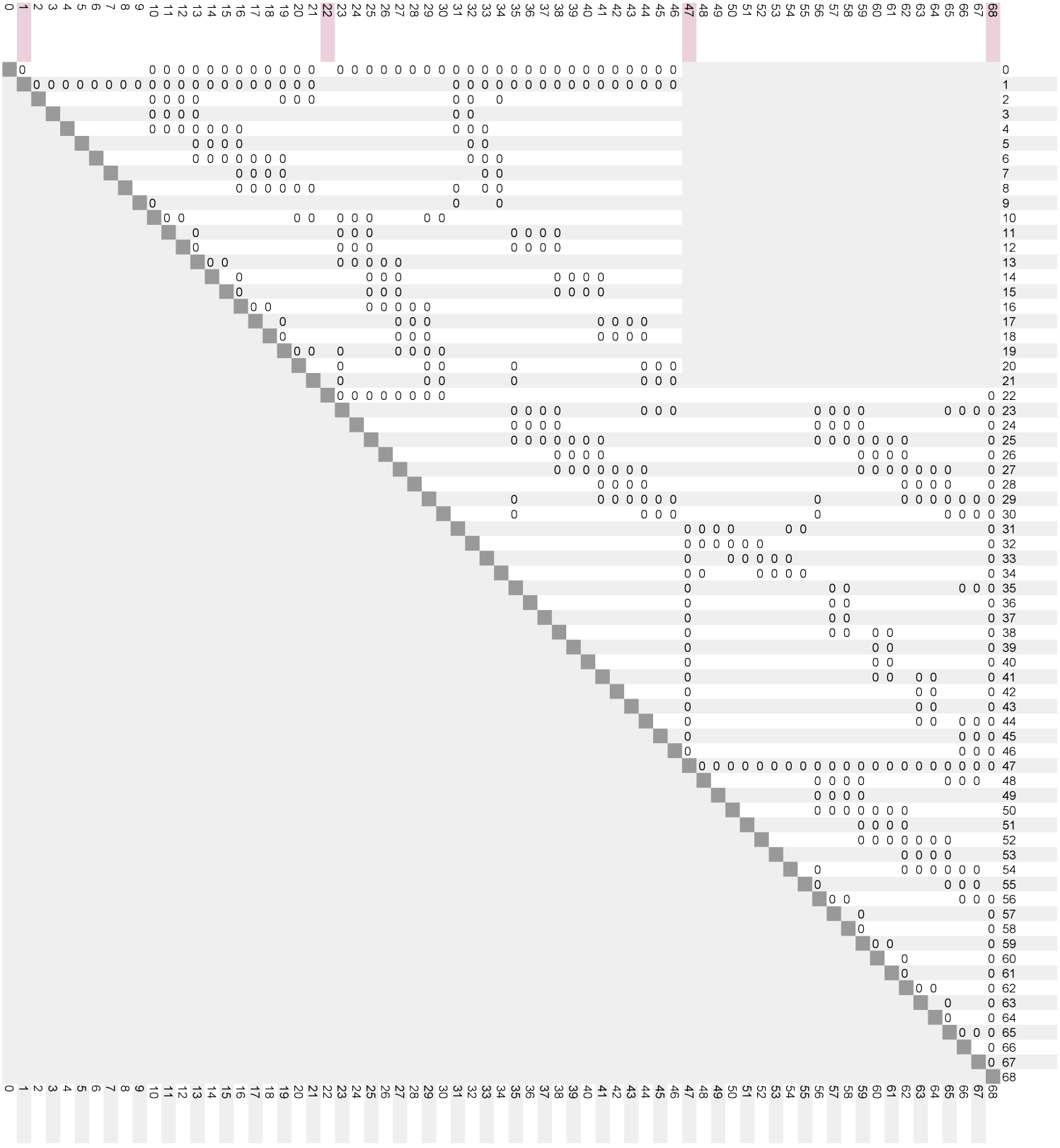


Figure A.1: The comparison table used by the comparison_dictionary algorithm to determine all to-be-compared neighbour pairs

A.5. History Check

A.5.1. History Check

```python
# History Check
# BAP Q1 2022-2023 Subgroup 2
# 2022/11/30
# Version 3.0

# Calculates whether ot not it is possible that a resident can be at a given location based on the previous location.
# This is accomplished by stating at the previous location and filling the hallway with the distance from the start
# point (starting at 0, descending) until it reaches its destination. It then checks this distance against the
# maximum achievable distance based on velocity and difference in time

import numpy as np
import pandas as pd

# Shows which coordinates are the direct neighbours of the current point (2D)
def lookaround(x, y, dimensions):
    neighbours = [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]
    # Prevents going out of bounds
    for n in neighbours:
        if not 0 <= n[0] <= dimensions[0] - 1 or not 0 <= n[1] <= dimensions[1] - 1:
            neighbours.remove(n)
    return neighbours


# Determines the shortest distance between two points in a two-dimensional map by filling the hallway squares with
# distances from the start point until it encounters the end point, after which it returns its current distance counter.
# Inputs:   Nursingmap      = the two-dimensional matrix map of the target floor of the nursing home
#           CurLoc, PrevLoc = the two-dimensional coordinates of the start and end points
def distancecalc(Nursingmap, CurLoc, PrevLoc):
    disCounter = 0  # The distance from the start point (starts at 0, descending)
    dimensions = Nursingmap.shape  # Extract the dimensions of the map matrix
    completed = 0  # Indicates whether the algorithm has finished running and exits the loop
    TimeoutCounter = 0  # tracks the number od runs to detect infinite loops
    checkList = list()  # List to be filled with hallway coordinates
    # Skip entire calculation if start and end points are the same and return distance = 0
    if CurLoc == PrevLoc:
        pass
    else:
        # Go through the entire matrix
        for i in range(dimensions[0]):
            for j in range(dimensions[1]):
                # Determine start location based on previously measured point
                if (i, j) == (PrevLoc):
                    startCord = (i, j)
                # And save the locations of the hallways
```

```
45                  elif Nursingmap[i, j] == 0:
46                      extension = (i, j)
47                      checkList.append(extension)
48
49          # Start the first iteration from the start point
50          firstit = lookaround(startCord[0], startCord[1], dimensions)
51          for s in firstit:
52              if Nursingmap[s] == 0:
53                  Nursingmap[s] = disCounter - 1
54          disCounter -= 1
55
56          # Walk through every point along the hallway (0) and keep track of how many
             ↪ steps were taken
57          while completed == 0:
58              # Go through checklist with hall points
59              for k in checkList:
60                  # If they are at the edge of the mapped field
61                  if Nursingmap[k] == disCounter:
62                      # Look around
63                      itpoints = lookaround(k[0], k[1], dimensions)
64                      for i in itpoints:
65                          # Check if the target room is encountered
66                          if i == CurLoc:
67                              # If true, exit the loop
68                              completed = 1
69                              break
70                          # Check if point is a hallway
71                          elif Nursingmap[i] == 0:
72                              # Assign a new distance
73                              Nursingmap[i] = disCounter - 1
74              disCounter -= 1
75              TimeoutCounter += 1
76
77              # Return 0 if timed out
78              if TimeoutCounter >= 1000:
79                  disCounter = 10000000
80                  break
81      return abs(disCounter)
82
83
84  # Main algorithm When start and end are on the same floor, it simply finds the
     ↪ shortest route using the distancecalc
85  # function. If start and end points are on different floors, we first find the
     ↪ closest staircase that leads in the
86  # right direction and enter the next floor. Then we will go to the next floor and
     ↪ repeat until we are on the same floor.
87  # From there we walk from the staircase to the desired end point.
88  # Inputs:   deltaT          = time since last measurement
89  #           threeDmap       = the three-dimensional matrix map of the nursing home
90  #           CurLoc, PrevLoc = the room numbers of the current and the previous
     ↪ locations
91  # It outputs whether it is possible to travel the calculated route in the given time
     ↪ (1) or not (0)
92  def threeDhistory(deltaT, threeDmap, CurLoc, PrevLoc):
93      velocity = .05  # Velocity in squares per second
94      Nursingmap = threeDmap.copy()  # Store a copy of the inputted map
```

```
95      dimensions = Nursingmap.shape  # Extract the dimensions of the map matrix
96      maxDistance = 3 + (velocity * deltaT)  # Evaluates the maxed distance traveled in
        ↪ Delta T
97      stairList = list()  # List to be filled with stairway coordinates
98      liftlist = list()  # List to be filled with lift coordinates
99      distance = 0  # the total distance between two points
100
101     data = "C:\\Users\\borin\\Documents\\BAP\\Hist_Dictionary.csv"
102     df_map = pd.read_csv(data, header=None)  # load map from csv
103     dictionary = df_map.to_numpy()
104
105     # If the location has not changed, it is always possible
106     if float(CurLoc) == float(PrevLoc):
107         Checkresult = 1
108     else:
109         # Go through the entire matrix
110         for h in range(dimensions[0]):
111             for i in range(dimensions[1]):
112                 for j in range(dimensions[2]):
113                     # Determine start location based on previously measured point
114                     if Nursingmap[h, i, j] == float(PrevLoc):
115                         startCord = (h, i, j)
116                     # Determine end location based on currently measured point
117                     elif Nursingmap[h, i, j] == float(CurLoc):
118                         endCord = (h, i, j)
119                     # Save the locations of the stairways
120                     elif Nursingmap[h, i, j] == -0.5:
121                         extension = (h, i, j)
122                         stairList.append(extension)
123                     # Save the locations of the lifts
124                     elif Nursingmap[h, i, j] == -1.5:
125                         extension = (h, i, j)
126                         liftlist.append(extension)
127
128         # Transform start and end coordinates to 2D, also save the start floor map
            ↪ and the floor number
129         begin = (startCord[1], startCord[2])
130         finish = (endCord[1], endCord[2])
131         floor = startCord[0]
132         floormap = Nursingmap[floor, :, :]
133
134         # If start and end points are on the same floor
135         if startCord[0] == endCord[0]:
136             # Perform 2D distancecalc
137             distance = distancecalc(floormap, begin, finish)
138         else:
139             # Set current location to begin
140             currentlocation = begin
141             while floor != endCord[0]:
142                 # Look for the lifts and stairs on the same floor which goes in the
                    ↪ desired direction
143                 # Create empty lists to be filled with (2D) lifts and staircases on
                    ↪ the same floor
144                 lifts = list()
145                 staircases = list()
146                 # Find all lifts that go in the right direction
```

```
147                 for lift in liftlist:
148                     if lift[0] == floor:
149                         # convert coordinates to 2D
150                         samefloorlift = (lift[1], lift[2])
151                         # If target is above current floor
152                         if endCord[0] < floor:
153                             # And lift below exists
154                             liftexitsts = liftlist.count((floor - 1, lift[1],
                                ↪ lift[2]))
155                             if liftexitsts > 0:
156                                 # Append to lifts list (2D)
157                                 lifts.append(samefloorlift)
158                             newfloor = floor - 1
159                         # If target is below current floor
160                         else:
161                             # And lift above exists
162                             liftexitsts = liftlist.count((floor + 1, lift[1],
                                ↪ lift[2]))
163                             if liftexitsts > 0:
164                                 # Append to lifts list (2D)
165                                 lifts.append(samefloorlift)
166                             newfloor = floor + 1
167 
168                 # Find all stairs that go in the right direction
169                 for stair in stairList:
170                     if stair[0] == floor:
171                         # convert coordinates to 2D
172                         samefloorstair = (stair[1], stair[2])
173                         # If target is above current floor
174                         if endCord[0] < floor:
175                             # And stair below exists
176                             stexitsts = stairList.count((floor - 1, stair[1],
                                ↪ stair[2]))
177                             if stexitsts > 0:
178                                 # Append to staircases list (2D)
179                                 staircases.append(samefloorstair)
180                             newfloor = floor - 1
181                         # If target is below current floor
182                         else:
183                             # And stair above exists
184                             stexitsts = stairList.count((floor + 1, stair[1],
                                ↪ stair[2]))
185                             if stexitsts > 0:
186                                 # Append to staircases list (2D)
187                                 staircases.append(samefloorstair)
188                             newfloor = floor + 1
189 
190                 # Now we have created a list of lifts and stairs on the current floor
191                 # Search for the closest viable lift and take it if it exists
192                 closestobjectdistance = 10000  # Distance to the closest staircase
                    ↪ (initially set far away)
193                 # Search for the closest viable staircase to the current position
194                 for elevator in lifts:
195 
196                     tdist = distancecalc(floormap, currentlocation, elevator)
197                     if tdist <= closestobjectdistance:
```

```
                        closestobjectdistance = tdist
                        # Remember the 2D coordinates of this stair
                        closestobject = elevator
                # If no (close) lifts were encountered, find the closest stair
                if closestobjectdistance == 10000:
                    # Search for the closest viable staircase to the current position
                    for stcase in staircases:
                        tdist = distancecalc(floormap, currentlocation, stcase)
                        if tdist <= closestobjectdistance:
                            closestobjectdistance = tdist
                            # Remember the 2D coordinates of this stair
                            closestobject = stcase
                # Change location parameters the closest object (lift or stair)
                floor = newfloor                        # Move to the new floor
                currentlocation = closestobject         # Set location to the
                 ↪  location of the object
                distance += closestobjectdistance + 1   # Increase the distance
                 ↪  accordingly
                floormap = Nursingmap[floor, :, :]      # Load floormap of new floor

            # Walk over the current floor to the end point
            distance += distancecalc(floormap, currentlocation, finish)

        # If somewhere something timer out, return -1
        if distance >= 10000000:
            Checkresult = -1
    # Check if measured distance is realistic
    # if distance <= maxDistance:
    #     Checkresult = 1
        else:
            # print("Measured distance is", distance, " and Max distance is",
             ↪  maxDistance)
            Checkresult = dictionary[int(round(maxDistance, 0))][distance]
    return Checkresult

```

A.5.2. History Probability

```
# History Probability
# BAP Q1 2022-2023 Subgroup 2
# 2022/12/06
# Version 1.1
#
# Takes the values for measured distance and max distance from the History Check
 ↪  file. Then it plots a normal
# distribution around the measured distance to create a probability density function
 ↪  of this measured distance. This is
# then multiplied it with a filter function based on the calculated maximum distance
 ↪  and integrates it to obtain an
# overall likelihood

import math
from scipy.stats import norm
from scipy.integrate import quad


```

```
# The main function, functionality described above
def hist_prob(measureddistance, max_distance):
    # The variance of the normal distribution
    var = 1.3 + 0.09 * measureddistance
    # The integration range
    r = measureddistance + 100
    # The main function
    p_x = quad(lambda x: cdf_maxdist(x, max_distance) * norm.pdf(x,
     ↪  loc=measureddistance, scale=var), -r, r)
    return p_x[0]


# Creates a maximum distance filter based on the actual statistical properties
 ↪  associated with it
def cdf_maxdist(p, maximumdistance):
    # The variance of the PDF
    vartwo = 0.069 * (maximumdistance**2) + 0.09 * measureddistance
    # Translates PDF into CDF
    p_x = quad(lambda m: norm.pdf(m, loc=maximumdistance, scale=vartwo), -100, p)
    return 1 - p_x[0]
```

A.5.3. History Dictionary

```
import numpy as np
import Hist_probability_1_1 as prob

# Creates a dictionary of precomputed hist_prob values in order to significantly
 ↪  speed up real time calculations
def hist_lib(d_meas_it, d_max_it):
    d_library = np.empty((d_max_it, d_meas_it))
    for d_max in range(d_max_it):
        for d_meas in range(d_meas_it):
            p = prob.hist_prob(d_meas, d_max)
            print(p)
            d_library[d_max][d_meas] = p
    return d_library
```

A.6. Ratings on Map

A.6.1. Map Rating

```
# BAP Q1 2022-2023 Subgroup 2
# Ratings RSSI on floor plan
# Create .csv with all rooms that received the signal replaced by the index
 ↪  indicating the order in which they received
# Version 1.0
# 2022/12/04 12:30 Corné

import numpy as np
import pandas as pd
from datetime import datetime

from Coupling_bed_bedsense_2 import Couplings
from Data_2 import Data
from Map_FloorPlan_1 import map_floorplan
```

```
14
15
16  def map_rating(data_couplings, data_measurements, data_map):
17      # Get some data
18      coupling = Couplings(data_couplings)  # get coupling data
19      measurements, bedsenses = Data(data_measurements)  # get measurement data
20      floor_plan = map_floorplan(data_map)  # get floor plan data
21
22      time = measurements[:, 0]  # time column
23      data = measurements[:, 1:]  # RSSI columns
24
25      ratings_on_map = [[], []]  # final list of time with RSSI rated map
26
27      for i in range(len(data)):
28          floor_plan_copy = floor_plan.copy()  # copy of floor_plan to fill ratings for
            ↪ current measurement
29          row = np.nan_to_num(data[i].astype(float), nan=-200)  # set all NaN values to
            ↪ very low RSSI value
30
31          row_isorted = np.flip(np.argsort(row))  # create index array of sorted (big
            ↪ -> small) current-measurement-row
32
33          for j in range(len(row_isorted)):  # for each measurement (big -> small)
34              if row[row_isorted[j]] > -200:  # check if value was measured at all
                ↪ (note, -200 was NaN)
35                  index_couplings = int(np.where(coupling[:, 1] ==
                    ↪ bedsenses[row_isorted[j]])[0])  # find bedsense in couplings...
36                  room = coupling[index_couplings, 0]  # and find belonging room number
37
38                  for k in range(len(floor_plan)):  # find room number in floor_plan
                    ↪ (np.where() not working here...)
39                      for l in range(len(floor_plan[0])):
40                          for m in range(len(floor_plan[0, 0])):
41                              if floor_plan[k, l, m] == room:
42                                  floor_plan_copy[k, l, m] = j + 1  # put strength of
                                    ↪ measurement in floor_plan_copy
43
44          timestamp = datetime.fromtimestamp(time[i]/1000)  # set timestamp to
            ↪ date-and-time
45
46          ratings_on_map[0].append(timestamp)  # save time of measurement
47          ratings_on_map[1].append(floor_plan_copy)  # save rating of measurement
48
49      return ratings_on_map
50
51
52  np.set_printoptions(suppress=True)  # get rid of scientific E-notation
53  print("processing data")
54
55
56  # ADJUST THIS ...
57  data_measurements = "/Users/corneploumen/Desktop/BAP/Data/Floor 6 - grouping 5s (max)
    ↪ - 16.50.19-17.05.16.csv"
58  output_filename = "Floor 6 - grouping 5s (max) - 16.50.19-17.05.16"
59
60
```

```
# DO NOT ADJUST ##################
data_map = "/Users/corneploumen/Desktop/BAP/Data/textual_map_laurens_blijdorp - Floor 5-10.csv"
data_couplings = "/Users/corneploumen/Desktop/BAP/Data/koppelingen_laurens (w.o. 416B and 719rechts).csv"

ratings_on_map = map_rating(data_couplings, data_measurements, data_map)  # Calling the function

print("exporting")
df = pd.DataFrame(ratings_on_map)
df.to_csv("Rating on map - %s.csv" % (output_filename))
```

A.6.2. Couplings (adjusted)

```
# BAP Q1 2022-2023 Subgroup 2
# Coupling bed bedsense
# Return np.array with room number in column 0 and bedsense number in column 1
# Version 2.0
# 2022/12/04 12:30 Corné

import pandas as pd


def Couplings(directory):
    data = open(directory)
    df = pd.read_csv(data)
    coupling = df.to_numpy()[:, :2]

    return coupling
```

A.6.3. Data (adjusted)

```
# BAP Q1 2022-2023 Subgroup 2
# Data
# Extract bedsenses and RSSI values in two separate arrays
# Version 2.0
# 2022/12/04 12:30 Corné

import pandas as pd
import numpy as np
import re


def Data(directory):
    # To load the data
    data = open(directory)  # This directory if different for everyone so change it when running the code
    df = pd.read_csv(data)
    df = df.apply(pd.to_numeric, errors='coerce')  # To convert all unknowns to NaN values

    # Creating the bedsense names
    column_names = list(df.columns)  # Extracting headers from dataframe
    bedsenses = []  # Create empty list for bedsenses
```

```
    for name in column_names:
        new_name = re.sub('\D', '', name)  # Delete all non-numeric characters from list of headers
        bedsenses.append(new_name)  # Add new headernames to bedsenses list

    bedsenses[:] = [bed for bed in bedsenses if bed]  # Delete all empty headers
    bedsenses = [eval(bed) for bed in bedsenses]
    bedsenses = np.array(bedsenses)  # Transform bedsenses list into numpy array

    df = df.to_numpy()  # Converting dataframe to a numpy array

    # Setting all values below -90 to zero, as it is no longer useful to work with
    df[df < -90] = np.nan

    return df, bedsenses
```

A.6.4. Map Floorplan (adjusted version of Map Matrix)

```
# BAP Q1 2022-2023 Subgroup 2
# Map to floor plan
# create floor_plan 3-D array
# Version 1.0
# 2022/12/04 12:30 Corné

import numpy as np
import pandas as pd


def map_floorplan(data):
    data = open(data)
    df_blueprint = pd.read_csv(data, header=None)  # load building's "blueprint" from csv into dataframe
    blueprint = df_blueprint.to_numpy()  # transform dataframe to np.array

    floor_height = (np.shape(blueprint)[0] - 1) / blueprint[0, 0]  # height of each individual map
    floor_plan = []
    for i in range(int(blueprint[0, 0])):
        floor_plan.append(blueprint[int(i * floor_height + 1):int((i + 1) * floor_height + 1), :])  # stack floor maps
    floor_plan = np.array(floor_plan)  # 3-D array of entire floor plan (stacked 2-D floor maps of each floor)

    return floor_plan
```

A.7. Multilateration

A.7.1. Multilateration

```
# BAP Q1 2022-2023 Subgroup 2
# Localization Algorithm Iteration
# Using trilateration, find button's location using every set of 3 bedsenses and taking the mean of all resulting locations
# Version 3.0
# 2022/11/06 17:00 Corné
```

```
6
7  import numpy as np
8  from datetime import datetime
9
10 import Coupling_bed_bedsense_1_1 as coup
11 import Data_1_0 as sort_data
12 import Trilateration_1_0 as trilat
13 import Inverse_Square_Law_1_0 as isl
14
15
16 def loc_1(loc, directory):
17     # Calling the function Couplings to obtain the beds coupled to the bedsenses
18     coupling, pos = coup.couplings(loc)
19
20     # Calling the function Data to obtain a sorted by 10 sec dataframe and an array
       ↪ with bedsense numbers
21     df, bedsenses = sort_data.data(directory)
22
23     time = df[:, 0]  # Time column
24     data = df[:, 1:]  # Data columns for all bedsenses
25
26     t = 0
27     j = 0
28     avg_location = np.array([0., 0.])
29     for row in data:
30         mean_location = np.array([0., 0.])
31         i = 0
32         nan = not(np.isnan(row.astype(float)))  # Highlight non NaN values in row
33
34         if np.count_nonzero(nan) > 2:  # Check if row has more than 2 non NaN values
35             for m in range(len(row)-2):  # Itterate over all sets of three bedsenses
                   ↪ measurements
36                 for n in range(m+1, len(row)-1):
37                     for o in range(n+1, len(row)):
38                         if m != n and m != o and n != o:
39                             tri_dis = np.array([[row[m]], [row[n]], [row[o]]])  #
                               ↪ Distances to button for triangle of bedsenses k, k+1,
                               ↪ and l
40
41                             if not(np.any(np.isnan(tri_dis))):  # Check if all
                               ↪ distances are non NaN
42                                 pos1 =
                                   ↪ pos[pos['Bedsense'].isin([bedsenses[m]])].to_numpy()[0,
                                   ↪ 1:]  # Read coordinates of three bedsenses
43                                 pos2 =
                                   ↪ pos[pos['Bedsense'].isin([bedsenses[n]])].to_numpy()[0,
                                   ↪ 1:]
44                                 pos3 =
                                   ↪ pos[pos['Bedsense'].isin([bedsenses[o]])].to_numpy()[0,
                                   ↪ 1:]
45                                 tri_dis = np.array([np.append(pos1,
                                   ↪ isl.inverse_square(tri_dis[0])), np.append(pos2,
                                   ↪ isl.inverse_square(tri_dis[1])), np.append(pos3,
                                   ↪ isl.inverse_square(tri_dis[2]))])  # Create array
                                   ↪ with coordinates + distances
46
```

```
                                location = trilat.trilaterate(tri_dis)  # Trilaterate button with three bedsenses
                                if not(np.any(np.isnan(location))):  # If trilateration does not result NaN values
                                    mean_location += location  # Add location to
                                    i += 1

            if i > 0:
                mean_location /= i  # Average the locations found by all trilaterations
                avg_location += mean_location
                j += 1

                timestamp = datetime.fromtimestamp(time[t]/1000)  # Translate time from milisec to date & time
                t += 1

                print("De drukknop is op ", mean_location, " op tijd ", timestamp)

        else:  # If row contains less than three non NaN values
            data = np.delete(data, t)  # Delete row
            time = np.delete(time, t)  # Delete corresponding time

        if j > 15:
            avg_location /= j
            j = 0

            timestamp = datetime.fromtimestamp(time[t] / 1000)  # Translate time from milisec to date & time
            t += 1

            print("De drukknop is geschat op ", avg_location, " op tijd ", timestamp)
            avg_location = np.array([0., 0.])
    return
```

A.7.2. Trilateration

```
# BAP Q1 2022-2023 Subgroup 2
# Trilateration algorithm
# Trilaterate, aka. locate a point in 2D using a set of three ankers
# Version 1.0
# 2022/11/06 17:00 Corné

import numpy as np


# Optional alternative (not yet investigated): https://pypi.org/project/easy-trilateration/
def trilaterate(distances):
    p1 = np.array(distances[0, :2])  # Coordinates of each bedsense
    p2 = np.array(distances[1, :2])
    p3 = np.array(distances[2, :2])

    r1 = distances[0, -1]  # Distance between button and each bedsense
    r2 = distances[1, -1]
    r3 = distances[2, -1]
```

```

    # This block applies the basic trilateration math from wikipedia_1 to any case_2
    # 1: https://en.wikipedia.org/wiki/True-range_multilateration
    # 2: https://stackoverflow.com/questions/9747227/2d-trilateration
    v1 = p2-p1
    v2 = p3-p1
    e_x = v1/np.linalg.norm(v1)
    i = np.dot(e_x, v2)
    e_y = (v2-(i*e_x))/(np.linalg.norm(v2-(i*e_x)))
    d = np.linalg.norm(v1)
    j = np.dot(e_y, v2)

    x = ((r1**2)-(r2**2)+(d**2))/(2*d)
    y = (((r1**2)-(r3**2)+(i**2)+(j**2))/(2*j)) - ((i/j)*x)
    position = p1+(x*e_x)+(y*e_y)

    return position
```

A.7.3. Path Loss Model

```
# BAP Q1 2022-2023 Subgroup 2
# Path Loss Model
# Translate RSSI in dBm to distance (aka the radius of the sphere intersecting the
 ↪  button and bedsense as its centre)
# Version 1.0
# 2022/11/07 15:00

import math as m


def inverse_square(P_dBm):
    G = 0.000002  # empirically determined
    P_tx = 1

    distance = m.sqrt( 1/(4*m.pi) * G * P_tx * 10**(2-P_dBm/10) )

    return distance
```

B

Antenna Datasheet

Innovative Service Around the Globe YAGEO

DATA SHEET

WIRELESS COMPONENTS

PCB type antenna

ANTX150P116B08683

860~880MHz

YAGEO Phycomp

Product Specification – January 25, 2018 V.0

FEATURES & BENEFITS

- The smallest PCB antenna in the market
- Miniature design allows users to save required space
- Double-side adhesive tape makes it easy to install in device
- Ranges of types of connector and cable provide a flexible design options
- Halogen free and RoHS compliant

APPLICATIONS

- Tablet / Desktop PC
- Internet TV / STB / Game console / Camera
- WiFi network devices (IEEE 802.11b/g/n)
- Bluetooth / ZigBee devices
- Car Infotainment
- Smart meter
- Lighting control
- POS terminal
- Wireless Industrial Control

ORDERING INFORMATION-GLOBAL PART NUMBER, PHYCOMP CTC & 12NC

All part numbers are identified by the series, packing type, material, size, antenna type, working frequency and packing quantity.

YAGEO BRAND ordering code

GLOBAL PART NUMBER (PREFERRED)

ANT	X150	P	116	B	0868	3
(1)	(2)	(3)	(4)	(5)	(6)	(7)

(1) FAMILY

ANT = Antenna products

(2) CONNECTOR & CABLE LENGTH (MM)

X = I-PEX
150 = 150mm

(3) ANTENNA TYPE

P=PCB

(4) SERIAL NUMBER

116 = SERIAL NUMBER 116

(5) PACKAGE TYPE

B = Bulk

(6) WORKING FREQUENCY

868 = 860~880MHz

(7) CABLE TYPE

3 = 1.13mm diameter Mini-Coaxial Cable

SPECIFICATIONS

Table 1

DESCRIPTION	VALUE
Working Frequency	860~880MHz
VSWR	2.5:1 max
Peak Gain	2dBi
Polarization	Linear
Radiation Pattern	Omni-directional
Impedance	50 Ω Nominal
Operating Temperature	–40 °C to 85 °C
Maximum Power	1 W
Dimension	34mm x 7mm x 0.95mm
Radio Connector	I-PEX (20278-112R-13)
Cable Diameter / Length / Color	1.13mm / 150mm / Black
Mounting	Adhesive Tape (HF-DS)

DIMENSIONS

Table 2 Mechanical

DIMENSION	VALUE
L (mm)	150+-3.00
W (mm)	34+-0.30
H (mm)	7+-0.30
T (mm)	0.95+-0.15
A (mm)	2.30 Max

OUTLINES

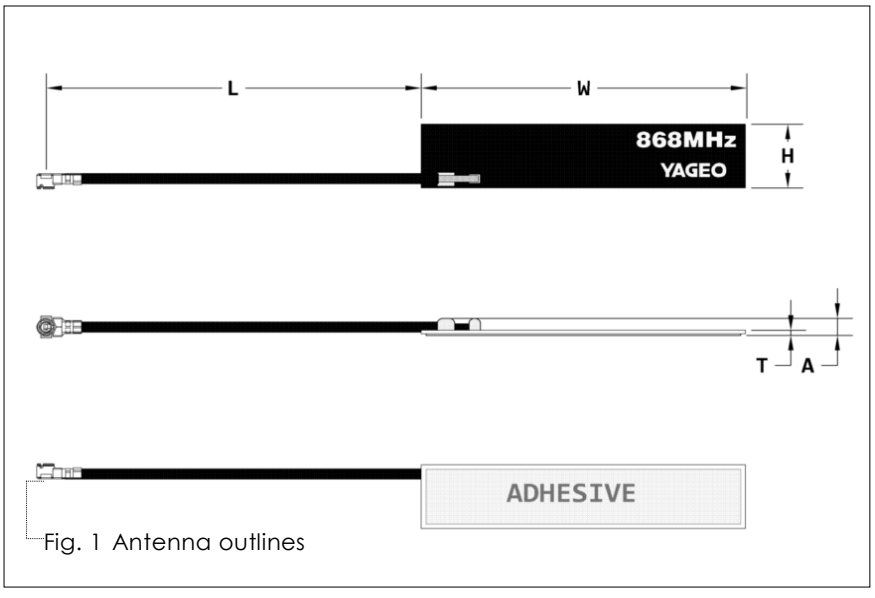
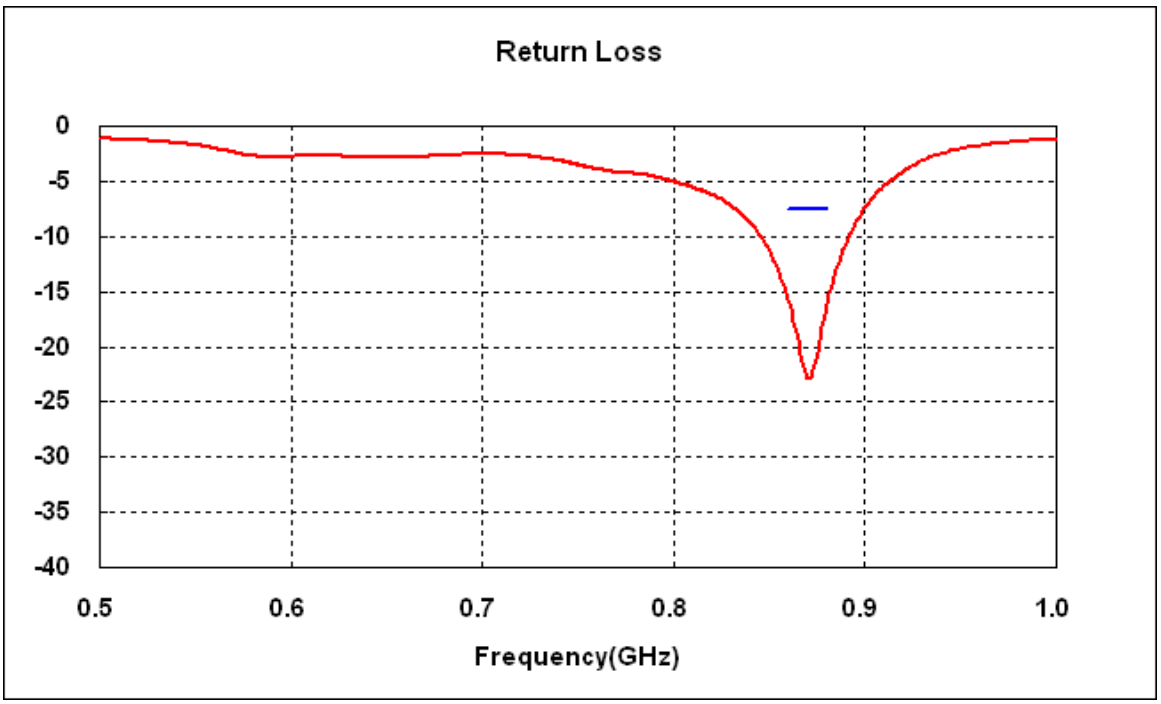


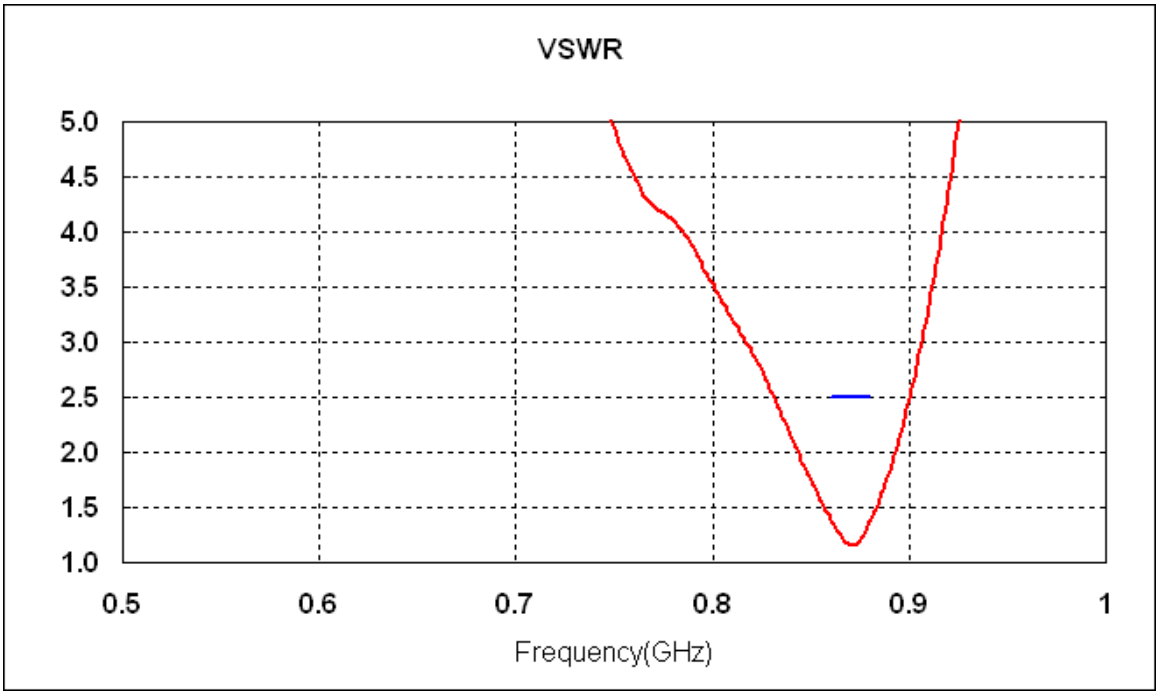
Fig. 1 Antenna outlines

RETURN LOSS & VSWR

Return Loss (dB)



VSWR



Frequency (GHz)

Fig. 2 Return loss & VSWR Measurement

ANTENNA GAIN & EFFICIENCY

Table 3

FREQUENCY (GHz)	AVERAGE GAIN (dBi)	EFFICIENCY (%)	PEAK GAIN (dBi)
860	-2.2	60.4	2.0
868	-2.0	62.6	2.0
875	-2.4	57.6	1.6
880	-2.6	55.6	1.4

ANTENNA RADIATION PATTERNS

Scale: 5 dBi / div Max : 5 dBi Min : -20 dBi

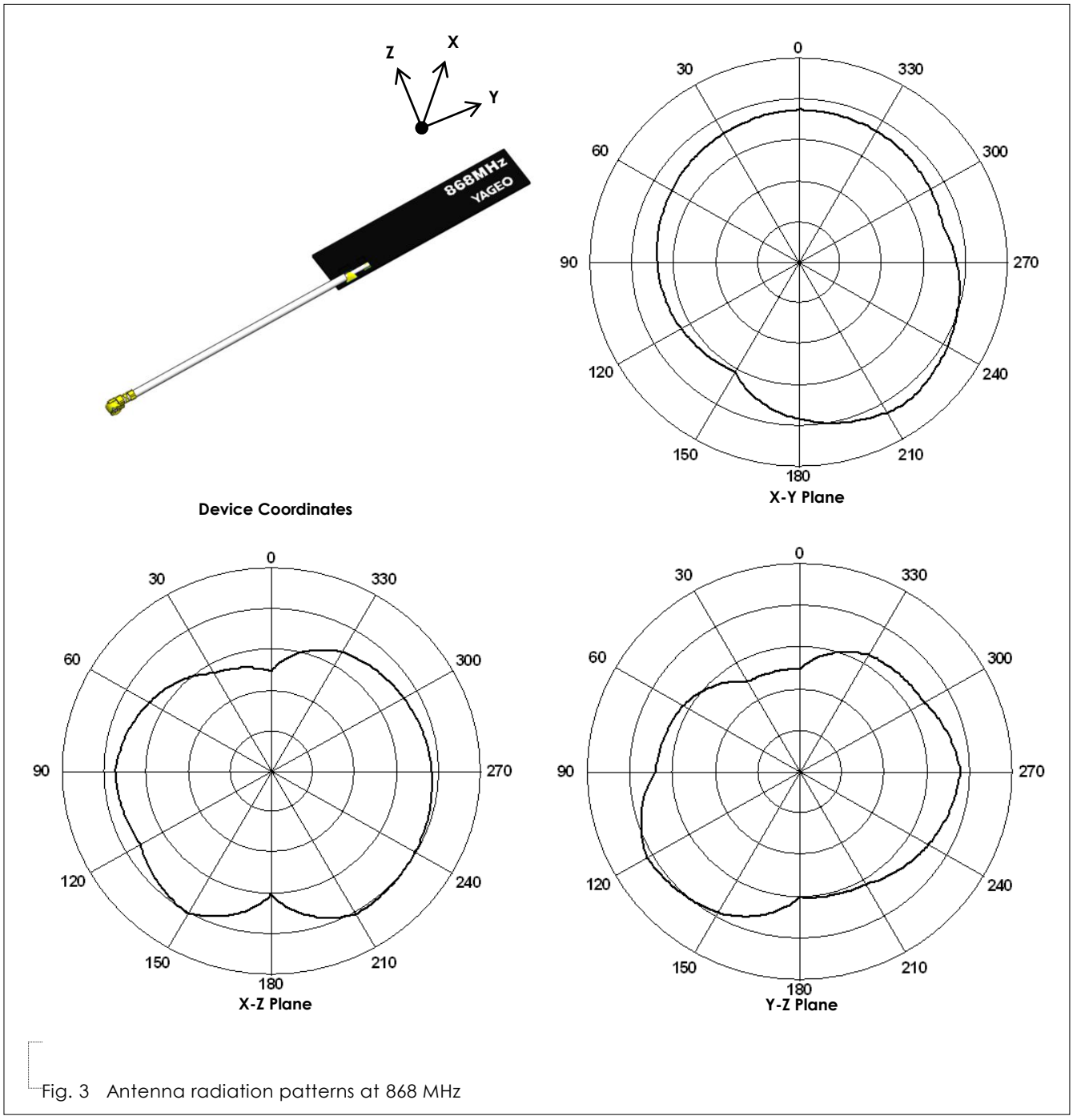


Fig. 3 Antenna radiation patterns at 868 MHz

REVISION HISTORY

REVISION	DATE	CHANGE NOTIFICATION	DESCRIPTION
Version 0	Jan. 15, 2018	-	- New data sheet for PCB type antenna, 860~880MHz application