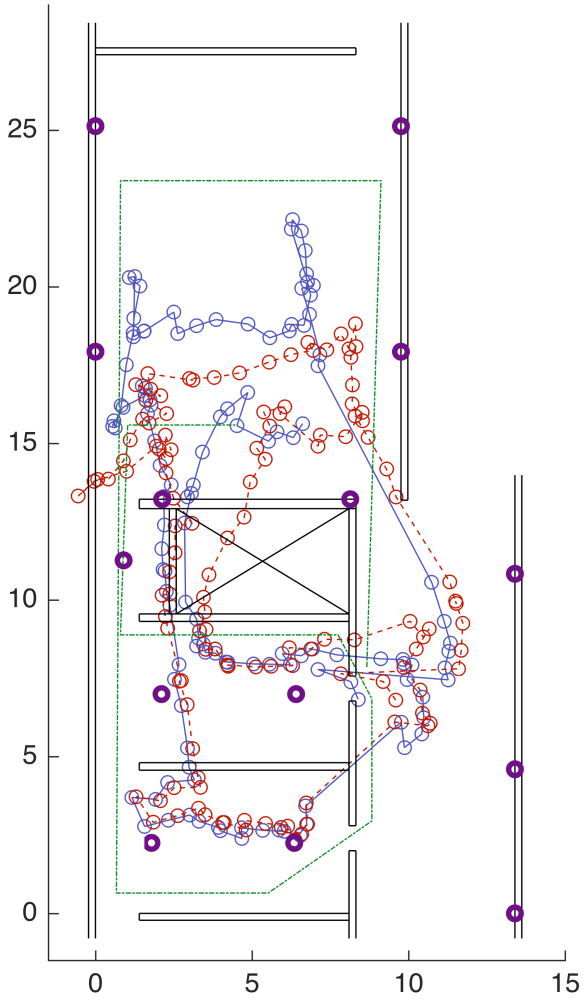
CHALMERS
UNIVERSITY OF TECHNOLOGY



Sensor Fusion based Indoor Positioning with iBeacons

GUSTAV EHRENBORG
HERMAN FRANSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

Master's thesis 2015:10

Sensor Fusion based Indoor Positioning with iBeacons

GUSTAV EHRENBORG
HERMAN FRANSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Sensor Fusion based Indoor Positioning with iBeacons
GUSTAV EHRENBORG
HERMAN FRANSSON

Supervisor: Thomas Petig, Department of Computer Science and Engineering
Examiner: Olaf Landsiedel, Department of Computer Science and Engineering

Master's Thesis 2015:10
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The real path compared to the estimated paths, with map and without.

Department of Computer Science and Engineering
Gothenburg, Sweden 2015

Sensor Fusion based Indoor Positioning with iBeacons

GUSTAV EHRENBORG
HERMAN FRANSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Regarding outdoor positioning, GPS has become de facto standard, however, there is no equivalent system in the indoor scenario. The literature considers promising solutions in the domain of indoor positioning, however, it does not use widely available hardware. This thesis considers a new map-based approach on indoor positioning that assumes the availability of an indoor map and affordable on-board sensors that are available on modern, off-the-shelf smartphones. By implementing a map matching algorithm, it is possible to reduce the uncertainty, arising from the use of affordable sensors, and improve the accuracy of the indoor positioning system.

A pilot of the design has been implemented and the results from the validation showed an average improvement of 17.8 % in accuracy and also an average improvement of 3.33 % in room correctness compared to the same design without including the indoor map. However, developers who choose to implement the map-based approach should be aware of the increased costs in computational demand and power consumption of the design when developing applications.

Keywords: iBeacon, sensor fusion, indoor positioning, Bluetooth 4.0, BLE, smartphone, Particle filter, Bluetooth smart, recursive Bayesian estimation, inertial navigation

This page has intentionally been left blank

Acknowledgements

We would like to express appreciation to our supervisor Thomas Petig for all implementation specific aid and encouragement. We would also like to thank Elad Michael Schiller for supporting us and especially for all the time spent on improving this report. Martin Tannerfors, and the other people at Trueflow, also deserves a lot of gratitude for having us and letting us accompany them on the beginning of their journey. Many thanks also to everyone else involved, both those who were involved in the project and those who supported us in stressful times.

Gustav Ehrenborg and Herman Fransson
Gothenburg, October 2015

This page has intentionally been left blank

Contents

List of Figures

List of Tables

1
Introduction

Positioning is a topic that is gaining in popularity. Multiple concepts of computing, such as ubiquitous computing and location- and content-aware systems, are dependent on positioning. Regarding outdoor positioning, GPS has become de facto standard, however, there is no equivalent system in the indoor scenario. The literature considers promising solutions in the domain of indoor positioning, however, it does not use widely available hardware. By making indoor positioning feasible with widely available hardware, the range of application can be broaden. This work offers an attractive solution for indoor positioning using only affordable sensors that are available on modern, off-the-shelf smartphones.

This work considers a new approach on indoor positioning that assumes the availability of an indoor map and affordable on-board sensors. By implementing a map matching algorithm, it is possible to reduce the uncertainty, arising from the use of affordable sensors, and improve the accuracy of the indoor positioning system. The implementation uses Bayes filter to swift out unwanted noise from the sensors and also to combine data from different sources, such as external beacons and inertial navigation.

1.1 The Challenge of Indoor Positioning

The problem considers estimating the position of a mobile device, where the device is moving on a plane and follows some unknown path. A path is defined as a step, which is defined as a motion on a short straight line in constant speed, between two points with a duration of a time unit t. A walk is defined as a number of subsequent steps.

The device receives rangings from beacons with well-known locations. A ranging is defined as $r = \{(x, y), d\}$, where $(x,\ y)$ is the position of the beacon and d is the distance to the beacon from the mobile device. The distance is derived from the signal strength of the received package from the beacons. During each time unit t, multiple rangings, $\{(x_i, y_i), d_i\}_{i \in I}$, are received. The device has to receive at least α rangings from unique beacons, $|I| \geq \alpha$.

The task is to, at the end of each step, estimate the current position, $(x,\ y)$, of the device. The challenge is to reduce the error, $(\hat{x},\ \hat{y})$, defined as the difference between the real and estimated position of the device.

1.1.1 Existing Solutions

Reducing the error is a challenge because the sensors are imprecise and subject to noise. The beacon rangings suffer from noise and non-deterministic delays, they are most likely also disturbed by reflections and interference. To filter out inaccuracies, Bayes filters can be used, for example a Kalman filter [1] or a particle filter [2]. The filters are able to filter out inaccuracies by estimating the unknown state with the help from the current measurements and the previous known states according to the hidden Markov model. A position is referred to as a state in this context. These filters also have the ability to fuse data from independent sources, a technique called sensor fusion [3]. This can be used to reduce the state space by removing unlikely states, which improves the estimation of the unknown state.

To utilise sensor fusion, it is assumed that the device has access to on-board sensors. It is also assumed that the on-board sensors consist of a gyroscope, an accelerometer and a magnetometer. These sensors also suffers from noise and inaccuracies as the rangings do [4]. However, by using sensor fusion, it is possible to reduce the impact from the noise [3].

1.1.2 Opportunity

Everywhere where an indoor positioning system is used, an application map must be present. An example is in exhibitions, where the visitors gets a map of the premises to make it easier for them to find what they are looking for. This map needs to contain information about the premises, such as the walls and doors, in other words, all possible locations for a visitor. In current indoor positioning system, this information is just used for relating a position into a location, but not in the position estimation process itself.

1.1.3 Our Approach

To improve the accuracy and the performance of the indoor positioning system, a model containing map information is included in the position estimation. The information from the map can be used to disqualify impossible states in the hidden Markov model. Impossible states are those that imply an impossible movement, such as a wall crossing. Because of the reduced state space, it is possible to improve the accuracy and performance since fewer, and only feasible, states has to be considered.

1.1.4 Evaluation Criteria

The result of this thesis is a prototype of an indoor positioning system. To validate its performance, two different criteria are evaluated. The first criterion is the *accuracy* which is the distance between the estimated position and the real position. The accuracy can be expressed in different ways, for example in a single time unit

or an average over time.

The accuracy does not, unfortunately, tell the whole truth. The accuracy can be good, but the position estimation can be in a adjacent room. To account for these scenarios, a second criterion, called *correct room estimation*, is introduced. The criterion is a boolean, true or false, telling whether the position estimation is in the correct room in the current time unit. Over time, the criterion can be expressed as a percentage. This criterion is useful in real life scenarios, where, for example, a shopping mall could offer indoor positioning and stores could use the information to target special offers to people depending on their location.

The correct room estimation and accuracy testing is carried out in a static and controllable environment. The tests are done with both the map, map-based, and without, map-less approach. The tests are repeatable and performed a predefined number of times to reduce the impact caused by arbitrary faults. When evaluating the possible performance gain of the map-based approach, the results from the map-less approach are used as a reference.

1.1.5 Our Contribution

This thesis studies the potential benefit of solving the indoor positioning problem while considering a map-based approach. Based on the potential improvement, we design a prototype and implement a pilot in MATLAB. The pilot is tested to validate the approach and the results show that it is worthwhile considering a map-based approach in future indoor positioning systems.

The prototype utilises Bluetooth low energy iBeacons and all sensor data has been collected on an iPhone 5S. The prototype uses a Bayes filter to accommodate for noise and inaccuracies. To estimate the position, a particle filter is used in combination with a mobility model which estimates the walk pattern of a human.

1.2 Related Work

The existing literature on indoor positioning relays on advanced hardware that is not likely to be available soon on off-the-self smartphones. This includes [5] and [6], which propose solutions while assuming the availability of specialised hardware, such as infrared cameras and precise on-board sensors. The goal of this thesis is to make indoor positioning available on modern smartphones, using widely available hardware. This thesis therefore proposes an alternative solution that utilises available sensors on a modern, off-the-shelf smartphone.

One way to solve the indoor positioning problem is by considering the use of internal sensors, so called inertial navigation. A requirement of inertial navigation is a highly accurate inertial measurement unit (IMU). The authors of [6] utilises inertial navigation in the form of pedestrian dead reckoning (PDR). The PDR algorithm is fed

with data from an IMU, worn on the foot, that detects steps and the heading of the user. The experiments in [6] shows a 5 % drift of the total travelled distance when using a highly accurate IMU. Also, the authors of [7] observes drift in their PDR implementation, using a torso-mounted IMU. Since the IMU in a modern smartphone is not accurate enough, inertial navigation is not feasible on its own without using accessory sensors.

Specialised hardware is commonly used to obtain accurate distance measurements. The authors of [8] and [9] are both using different kinds of sensors, RFID and ultrasonic beacons respectively, that is not affordable or is difficult to deploy. The solution proposed by [9] uses both angle of arrival (AoA) and time of flight (ToF) to position the device. These two kinds of measurements cannot be made when using a smartphone without the use of additional hardware and are therefore not considered options.

The literature explores the concept of using a map of the premises to aid in an indoor positioning system. An example is the authors of [10]. They saw an improvement in accuracy when including a map in the position estimation. Compared to [10], that used a particle filter, the authors of [11] implemented a deterministic map matching algorithm. They also noticed an improvement in the accuracy when a map was included in the position estimation.

To the best of our knowledge, no scientific publications have analysed a system where both iBeacons, inertial navigation and map matching has been used together on a modern, off-the-shelf smartphone.

1.3 Outline

The report is structured as follows. A Background chapter introduces the basic knowledge about positioning. It describes positioning techniques that are used in the implementation, the basics of Bayes filters and it explains how a particle filter works. The Hardware chapter contains all the information about the hardware that is used; the sensors in the smartphone, the beacons and some theory about Bluetooth. The next chapter is the Testing chapter which is about how the positioning system is validated. The results of the testing are presented in the subsequent Result chapter. The Result chapter also holds the information about the implementation of the positioning system. The report is finished with Discussion, Conclusions and Bibliography chapters.

2
Background

The indoor positioning system is based on the same principles as trilateration, that is, distances are used to estimate a position. Since the distance calculations are noisy and inaccurate, inertial navigation, using the accelerometer and magnetometer to estimate movement, is combined with the distance measurements. A particle filter is responsible for the fusing of the data sources, and it has access to multiple models to interpret the data.

2.1 Positioning Techniques

Several different techniques exists to calculate a position, and these can be sorted into two different types depending on the kind of data that is available. External positioning techniques use external information, for example distance values, angles or light seen through the camera, to estimate the position. Internal positioning techniques instead use internal information, such as the magnetometer, gyroscope or accelerometer to estimate the current position.

Some common ways to use external sensors are trilateration, triangulation and fingerprinting. Trilateration uses distance measurements to calculate the current position and triangulation uses angles to calculate the position. Fingerprinting is a bit different compared to the previous mentioned techniques. It utilises an offline radio heat map of the premises, which is then used to relate online information into a position estimation [12]. A technique using internal sensors is dead reckoning. It recursively calculates the new position based on the last position and the sensed movement. Dead reckoning comes in multiple variations depending on how the movement is calculated [6].

2.1.1 Trilateration

Trilateration uses distance measurements from multiple known external reference points, beacons, to determine the current position of the device. By knowing the distance to a single beacon in a two-dimensional world, it can be concluded that the user is located at the edge of a circle with the radius equal to the known distance from the beacon. By adding two more beacons, it is possible to get a position estimation where all the circles intersects. Trilateration also works in the three-dimensional world, where spheres intersects instead of circles. The problem with trilateration is when the measurements are inaccurate. In such cases, multiple or

none intersections can occur [13].

The distance measurements can be derived in various ways. In the case of iBeacons, the distance is approximated by using the received signal strength (RSS) values of the signals. See Chapter 3 for more information on the beacons. Another way is to calculate the time of flight, and derive the distance from the other known characteristics [12].

Figure 2.1 illustrates an example of trilateration in a two-dimensional plane. The positions of the reference points, $b_{1..3}$, are known, and so are the distances, $d_{1..3}$. With this information, the position of the intersection can be calculated.

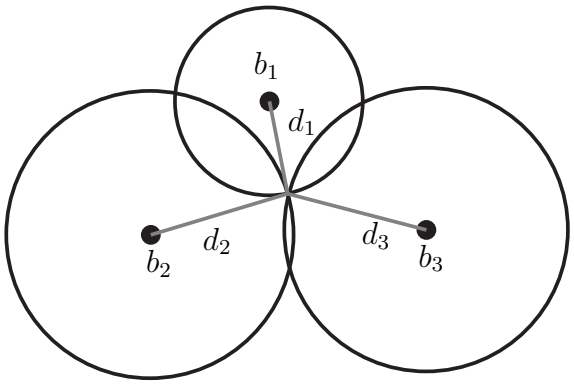


Figure 2.1: Illustration of trilateration

2.1.2 Inertial Navigation

Inertial navigation is the concept of only utilising internal motion sensors, such as gyroscopes, magnetometers and accelerometers, to calculate the position. The most common approach is dead reckoning where the movement is calculated and continuously added to the previous position, to calculate the new position [14].

There are different ways to calculate the movement. A common way is to integrate the acceleration, from an accelerometer, twice. This will result in the movement in three dimensions. Another way is to use a step detection algorithm on the accelerometer data. In combination with the heading from the magnetometer, every step will result in a movement in two dimensions. In this approach, the step length must also be known or estimated [14].

Drawbacks of inertial navigation are that drift is very common and that the start position must be known. Since the new position is recursively calculated from the last position, a small error might cause a greater error in the end [14].

2.2 Bayes Filters

Filters are mathematical methods for estimating unknown variables. The estimation is aided by known, observable variables like sensor input. In the localisation

case, the unknown variable is the location. The strength of filters is the ability to merge and combine multiple known variables from different sensors. This section introduces the basics of filters with a focus on positioning.

The Bayes filter, or recursive Bayesian estimation, is a probabilistic approach to estimate the location by using probability density functions. Since the position only can be estimated, not calculated with full certainty, the notion of belief is introduced. The belief is just an estimation of the state, and should not be confused with the real position. The belief is denoted as in Equation 2.1. The filter is given measurement data ($z_{1:t}$) and control data ($u_{1:t}$), applies this to different models and returns an estimation the new state of the system, a belief, at a time t [3].

$$bel(x_t) = p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.1}$$

The Bayes filter is usually implemented in two steps, where in the first step only incorporates the latest control data, and not the measurements update. This is denoted as in Equation 2.2. The different steps are called the prediction step and the update step.

$$\overline{bel}(x_t) = p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.2}$$

In the prediction step, unwanted noise is introduced from the inaccurate sensors, and the accuracy of the probability density function is decreased. The update step then increases the accuracy by applying the latest measurement data. Code snippet 2.1 lists a pseudocode implementation of the Bayes [3].

Listing 2.1: Pseudocode of the Bayes filter

```
1 bayes_filter(bel(x_{t-1}, u_t, z_t)):
2 for all x_t do
3       bel-bar(x_t) = ∫ p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}
4       bel(x_t) = η p(z_t|x_t) bel-bar(x_t)
5 endfor
6 return bel(x_t)
```

2.2.1 Particle Filter

The particle filter, PF, is a non-parametric implementation of the Bayes filter. Non-parametric filters use a finite number of values to approximate the posterior, where a parametric filter use functions. An advantage of non-parametric filters over parametric filters is the ability to represent multimodal beliefs, where the parametric filters only can handle the unimodal case [3]. This means that a PF can have multiple guesses, where parametric filters only can have one.

In the PF, a number of samples, called particles, are randomly drawn from the state space. These particles represent the distribution and each particle represents a possible position. Every particle has a weight that denotes it's possibility to represent

the correct state. The weight is used when resampling the particles.

Listing 2.2: Pseudocode of a particle filter, from [3]

```
1  particle_filter(X_{t-1}, u_t, z_t):
2  X̄_t = X_t = ∅
3  for m = 1 to M do
4      sample x_t^[m] ~ p(x_t|u_t, x_{t-1}^[m])
5      w_t^[m] = p(z_t|x_t^[m])
6      X̄_t = X̄_t + (x_t^[m], w_t^[m])
7  endfor
8  for m = 1 to M do
9      draw i with probability ∝ w_t^[i]
10     add x_t^[i] to X_t
11 endfor
12 return X_t
```

Listing 2.2 illustrates a particle filter. The input is, since the filter is recursive, the last state X_{t-1} and the control and the measurement updates, u_t and z_t. The initial state, X_0, can be initialised in numerous ways.

The filter starts by drawing M samples from the conditional density distribution $p(x_t|u_t,\ x_{t-1}^{[m]})$ on Line 4 and constructs the temporal belief $\overline{bel}(x_t)$. The distribution, from which the samples are drawn from, is based on the previous state x_{t-1} and the control u_t. On Line 5, the weight of the particles, which represents the temporal belief $\overline{bel}(x_t)$, is calculated based on the conditional probability $p(z_t|x_t^{[m]})$ where z_t is the measurement update. The particles, now representing the new belief $bel(x_t)$, is then put into a temporary set, $\overline{X_t}$, on Line 6.

The next step is the resampling of the particles. How the resampling is made can vary in different implementations. In this step, the M particles that should be in the real set are chosen from the temporary set $\overline{X_t}$. A particle can be chosen multiple times or none, and it depends on its weight if it is chosen or not. That means that a particle can be copied and be a part of the particle set on the same spot, with the same weight. The chosen particles are added to the definitive set of particles, X_t [3].

2.3 Models

There are several different models on how to draw samples from the conditional density distribution $p(x_t|u_t,\ x_{t-1}^{[m]})$ and how to calculate the weight of a particle based on the conditional probability $p(z_t|x_t^{[m]})$. When dealing with localisation, the models which extract samples from $p(x_t|u_t,\ x_{t-1}^{[m]})$ are often referred to as motion

models and those who calculate the weights are referred to as ranging models. Each of these models are however highly implementation dependent on how they extract a sample, but they are based on the same principals.

2.3.1 The motion model

A motion model is modelled by a conditional density distribution $p(x_t|u_t,\ x_{t-1})$, where x_t is the new state given previous state x_{t-1} and the control u_t. The control u_t consists of information about the transition between the previous state and the new state. The difference between different motion models is how they make use of and what the control exists of. When working with robots, the control could be estimated from the odometry sensors like wheel encoders and steering angle, but it could also be estimated from the given commands to the engines on the robot. The models which rely on odometry information for the control are called odometry models and the models which relies on given commands are called velocity models. In the localisation case with humans, is it only possible to use an odometry model. This is because it is not possible to give commands about how to walk, it is only possible to retrieve information about how a human have walked [3].

A possible drawback of the odometry model is that it relies on posterior data and that implies that a movement has to be detected before it can be used in the model. However, it is often a more accurate model since it is easier to estimate how much the user or the robot has moved rather than how much it is going to move.

2.3.2 The ranging model

The measurement model, when dealing with range sensors, is described as a conditional probability $p(z_t|x_t,\ m)$ where z_t is the measurement done at time t, x_t is the current state and the variable m is the map of the current location. The map is needed to be able to relate the measurements z_t to the current state x_t. This can be illustrated by a robot trying to locate itself in a building with the help from sonar sensors. To locate itself based on the results from the sonar, the robot needs to have an understanding of the current environment and various distances to different objects in the premises to be able to extract useful information from the measurement. When dealing with other types of sensors than range sensors, the current state x_t and the measurement z_t is all that is needed. Therefore, the conditional probability can be written as $p(z_t|x_t)$ were the map m, has been removed [3].

3
System

An iPhone 5S, made by Apple Inc., will be used during the validation. It was released in 2013 and was at that time a high-end smartphone. It ranks currently as the 172 best smartphone in Futuremarks Ice Storm Unlimited CPU and GPU benchmark[1]. The iPhone 5S is therefore a good example of a commonly used and commonly available, off-the-shelf, smartphone.

3.1 Sensors

The iPhone 5S has a lot of different sensors, some of which can be used in position estimation. These are for instance the gyroscope, the accelerometer and the magnetometer that is needed when designing an inertial navigation system. Another useful sensor that is found on an iPhone 5S is the Bluetooth low energy chip, which is needed when designing a navigation system that is based on external references.

3.1.1 Gyroscope

The gyroscope gives the user the possibility to measure at what speed (angular velocity) the device has been rotated with, around three different axis. These axis are orthogonal to each other and are often referred to as roll, pitch and yaw. As the gyroscope does not have an external frame of reference, it is only possible to measure the relative motion in these three axis. The output from the gyroscope is given in radians per second and often referred to as θ/s [2].

Due to the proprietary algorithms used by Apple Inc., it is unknown if the gyroscope data has been filtered before it becomes available for the user. However, it is most likely that it has been filtered in some way. The reason why this is suspected, is because when working directly with a gyroscope, the user normally needs to take the bias and drift of the sensor into account. This is not needed when working with the gyroscope from an iPhone[3].

[1] Futuremark. *Best Smartphones and Tablets.* `http://www.futuremark.com/hardware/mobileice_storm_unlimited/filter/androidioswinrtwindowsshowall`. Accessed: 2015-07-27. June 2015.

[2] Apple Inc. *Core Motion Framework Reference.* `https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/index.html`. Accessed: 2015-04-01. Sept. 2013.

[3] Apple Inc. *CMGyroData.* `https://developer.apple.com/library/ios/documentation/`

3.1.2 Accelerometer

The accelerometer works similar to how the gyroscope does, but instead of measuring the angular velocity, it measures the acceleration in three different direction which are orthogonal to each other. The accelerometer can refer to the external references, because of the presence of gravity. By knowing this, it is possible to fix the orientation of the accelerometer in the real world frame if the accelerometer is stationary. The output from the accelerometer is given in G:s ($G = 9.81m/s^2$) [4].

As the case with the gyroscope, it is not possible to know if the sensor data from the accelerometer has been filtered. It most likely has been filtered before the data is made available to the user.

3.1.3 Magnetometer

The magnetometer enables the user to measure the magnetic field around the sensor in three different orthogonal directions (x, y, z). By knowing the strength of the magnetic field in the different directions and comparing it to the magnetic field of the Earth, it is possible to calculate in which direction the sensor is pointing at. The output from the magnetometer is given in Tesla accompanied with a suitable prefix.

The output of this sensor might, just like the other sensors, be filtered by the smartphones operating system before it is delivered to the user. This is however not a problem as the operating system offers another convenient way to work with the magnetometer. The most common goal with the magnetometer is to determine the geographical north which implies some problems that needs to be solved. The first problem is to interpret the readings from the magnetometer, to extract the heading to the magnetic north. The second problem is that the magnetic north is not the same as the geographical north. The operating system solves these two problems and is able to directly output the true geographical north to the user[5].

3.1.4 iBeacon

The beacons used are from Estimote Inc. and they are certified under the Apple Inc. *iBeacon* specification. The certification includes specifications on, for example, identification and wireless technology of the beacon[6]. The beacons have a 32-bit

`CoreMotion / Reference / CMGyroData _ Class / index . html # / / apple _ ref / occ / instp / CMGyroData/rotationRate`. Accessed: 2015-08-15. May 2010.

[4] Apple Inc. *Core Motion Framework Reference*. `https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/index.html`. Accessed: 2015-04-01. Sept. 2013.

[5] Apple Inc. *Core Location Framework Reference*. `https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/index.html`. Accessed: 2015-07-01. Sept. 2013.

[6] Apple Inc. *Getting Started with iBeacon*. `https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf`. Accessed: 2015-03-31. June 2014.

ARM Cortex M0 CPU and supports Bluetooth low energy[7]. More technical specifications are listed in Table 3.1.

The beacons are set up to send advertisements at 1 Hz frequency, at their highest possible signal strength, +4 dBm. An update interval of 1 Hz gives a good balance between performance and battery life. The signal of the beacons are emitted in every direction and and the range at the highest signal strength is about 70 m.

CPU	ARM Cortex M0
Memory	256 kB
Bluetooth	2.4 GHz Bluetooth 4.0 BLE
Max range	70 m
Advertising frequency	$1 - 10$ Hz
Broadcasting power	-30 dBm to $+4$ dBm

Table 3.1: Estimote iBeacon technical specifications

3.2 Bluetooth Low Energy

Bluetooth is a wireless standard for connecting different types of devices and the main use is to transmit data over short distances. Today, Bluetooth is built into billions of products, mobile phones, computers, cars, etc. The technology operates on multiple different channels in the 2.4 GHz band, 2.4 to 2.485 GHz.[8].

Bluetooth low energy (BLE) is an extension to the original Bluetooth that was introduced in the Bluetooth 4.0 specification. The main purpose of the extension is to provide low-power communication of smaller amounts of data. The low-power characteristics enables BLE products to run for months on battery[9].

3.2.1 Signal Characteristics

A beacon advertises its UUID (universally unique identifier) and its major and minor values (configurable identifier values) at the selected advertising frequency. The UUID and major and minor values are used to identify the beacon. The identification will enable localisation of the beacon on a map.

The receiving smartphone will receive the received signal strength (RSS) value of the advertisement broadcast. The RSS value is used to calculate the distance between the sender (beacon) and receiver (smartphone). Many beacon manufacturers supply an API which calculates the distance, otherwise it can be calculated with

[7]Estimote Inc. *API Documentation*. `http://estimote.com/api/`. Accessed: 2015-03-24.

[8]Bluetooth SIG, Inc. *Bluetooth Fast Facts*. `http://www.bluetooth.com/Pages/Fast-Facts.aspx`. Accessed: 2015-05-05.

[9]Bluetooth SIG, Inc. *Bluetooth Fast Facts*. `http://www.bluetooth.com/Pages/Fast-Facts.aspx`. Accessed: 2015-05-05.

the following formulas. Equation 3.1 is the equation of the RSS, where n is the propagation path loss exponent, d is the distance, d_0 is a reference distance and X is the noise [22][23].

$$RSS(d) = RSS(d_0) + 10\,n\,log_{10}(\frac{d}{d_0}) \tag{3.1}$$

The propagation path loss exponent n is unknown, but can be derived from Equation 3.1, resulting in Equation 3.2.

$$n = \frac{RSS(d_0) - RSS(d)}{10\,log_{10}(\frac{d}{d_0})} \tag{3.2}$$

Equation 3.1 can be changed into Equation 3.3 to calculate the distance. $RSS(d_0)$ is a setting on the beacons, and it is $-60\ dBm$. d_0 is equal to $1\ m$.

$$d = d_0 * 10^{\frac{RSS(d_0)-RSS(d)}{10*n}} \tag{3.3}$$

The accuracy of the RSS values was examined by [24] and it was concluded that it is not suitable on its own to use for positioning purposes. This is since the RSS values fluctuate, and therefore the calculated distance will also fluctuate. Estimote, the manufacturer of the beacons used, reports deviations listed in Table 3.2[10].

The RSS values will fluctuate more or less depending on the environment. Obstructing line-of-sight has a great impact on the RSS values. People moving around, holding the smartphone at different angles, other signals, such as WiFi, and walls are typical things that also worsens the RSS fluctuations and also are common.

distance	deviation
20 cm	$5 - 6$ cm
1 m	15 cm
> 10 m	$2 - 3$ m

Table 3.2: Reported deviation from Estimote Inc.

[10]Estimote Inc. *What are the characteristics of Beacons' signal?* `https://community.estimote.com/hc/en-us/articles/201029223-What-are-the-characteristics-of-Beacons-signal-`. Accessed: 2015-05-06.

4
Testing

To validate the positioning system, a set of use cases is considered. Based on these use cases, a set of test cases and evaluation criteria are proposed to examine the performance of the positioning system. The testing itself is carried out in a public building consisting of several adjacent rooms and a hallway connecting them. The building is mostly used for conferences and meetings and is therefore spacious furnished.

4.1 Use cases

A set of use cases is identified to be able to do a proper evaluation of the system. With the help of these, a set of test cases and the associated evaluation criteria are proposed. The following two use cases are identified and are considered during the testing.

Routing
: Routing is a good example of a service that could benefit from a positioning system. Without having to rely on the user to supply the starting position, which could be erroneous, the positioning system supplies an estimation of the starting position instead. However, this puts great demands on the accuracy of the positioning system. If the position estimation is wrong with a few meters, it could lead to an incorrect routing which can affect both user experience and also the safety of the user in some cases.

Proximity marketing
: Location dependent services are something that is gaining in popularity. An example is proximity marketing where the user can get store dependent information, like sales or information about specific products. Another example could even be that competing stores offers coupons to customers who are in the competitors store. To be able to offer the right service, at the right time, it is important that the positioning system is able to correctly detect which location the user is located in.

4.2 Evaluation Criteria

To evaluate the use cases (Section 4.1), a set of evaluation criteria is defined. The first evaluation criterion is accuracy, which tests how far the estimated position is

from the real position. The second criterion is correct room estimation, a boolean telling if the estimated position is in the same room as the real position. This criterion is a complement to accuracy because the error of the estimation could be really small but still be in a adjacent room.

4.2.1 Accuracy

Accuracy is how correct the position estimation, $(x',\ y')$, is compared to the real position, $(x,\ y)$. The error, $(\hat{x},\ \hat{y})$, can be calculated, being the difference between the estimated and real position as shown in equations 4.1 and 4.2.

$$\hat{x} = |x - x'| \tag{4.1}$$

$$\hat{y} = |y - y'| \tag{4.2}$$

When a path has been traversed, an average error can be derived using Equation 4.3 where N is the total number of estimation points. The accuracy is the primary aspect of the evaluation criteria since the purpose of a positioning system is to give an as correct position estimation as possible.

$$error_{avg} = \sum_{i=0}^{N} \frac{\sqrt{\hat{x_i}^2 + \hat{y_i}^2}}{N} \tag{4.3}$$

4.2.2 Correct room estimation

The accuracy tells only half of the story about the error. The accuracy can be good, but the system may still estimate a position in another room than the real position. This is an important criterion since an indoor positioning system application may be based on the room estimation rather than the estimated position of the user. The correct room estimation criterion is used to measure this kind of error by assuming the value zero when the estimated position is in the wrong room and assuming the value one when in the correct room. From these values, a percentage can be calculated, like Equation 4.4 shows. In the equation, N is the total number of estimation points.

$$\frac{|\{(x_i', y_i') : r(x_i', y_i') = r(x_i, y_i)\}|}{N} \tag{4.4}$$

$$r(x, y) = room\ of\ coordinate\ (x, y) \tag{4.5}$$

4.3 Test Environment

The tests are carried out in a public building consisting of several adjacent rooms and a hallway connecting them. The floor plan and the placement of the beacons are visible in Figure 4.1. In the forthcoming figures, the black lines represent walls, the circles are the beacons and the dotted lines are the path walked, that is, the real position.

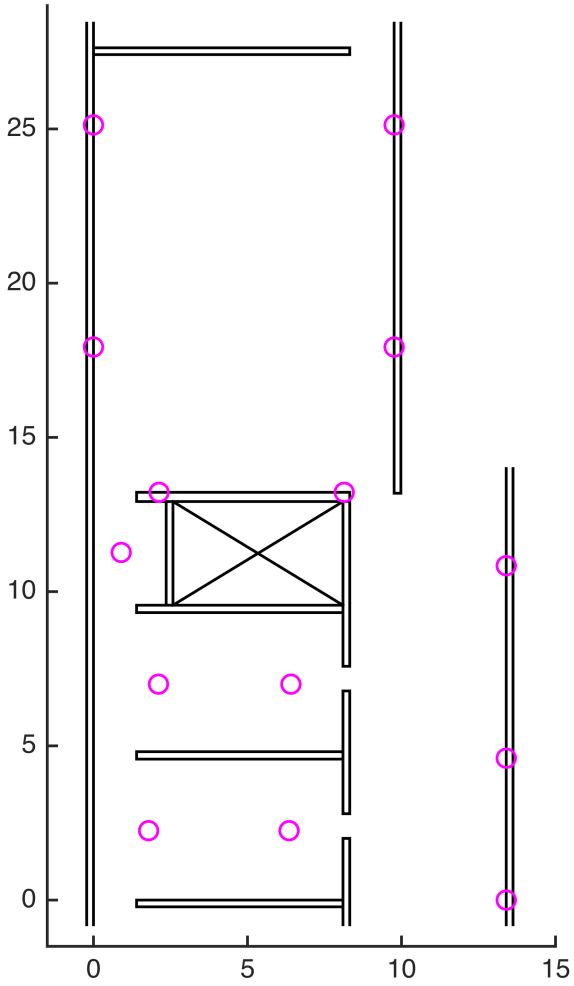


Figure 4.1: The testing environment, located in Chalmers student union building, on the second floor.

The two smaller rooms have a ceiling height of about 2.5 m whilst the greater room is a larger auditorium with high ceiling. The environment consists of mostly open space, housing some tables and chairs. The space was not sealed off from disturbing signals from other radio sources such as WiFi and other Bluetooth units. The environment is thereby a good example of a real world scenario where an indoor positioning system could have been operating in.

The path loss exponent in the testing environment was derived to approximately 2.1525 when using Equation 4.6, which also can be found in Chapter 3. The $RSS(d_0)$ is a setting on the beacons, and it is $-60\ dBm$. d_0 is equal to 1 m. The path loss exponent is used in the distance estimation, see Equation 3.3.

$$n = \frac{RSS(d_0) - RSS(d)}{10\, log_{10}(\frac{d}{d_0})} \tag{4.6}$$

4.4 Tests

A set of test cases are proposed according to the use cases (Section 4.1). Each test case examines different scenarios were the prototype could experience difficulties, such as door crossings or narrow passages. The tests are designed to be repeatable and are carried out in a static environment and is done for both the map-based approach and the map-less approach, to see the potential performance gain of the map.

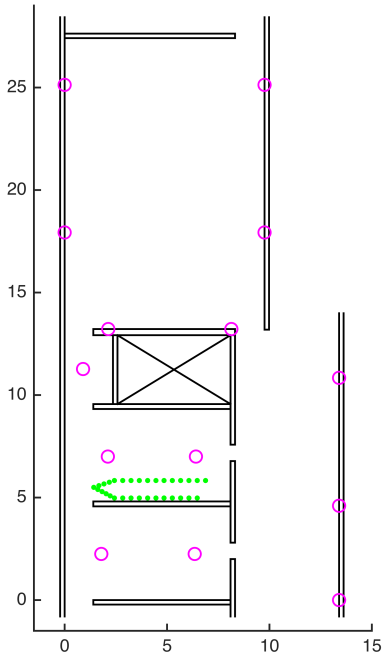


Figure 4.2: Wall Crossing test

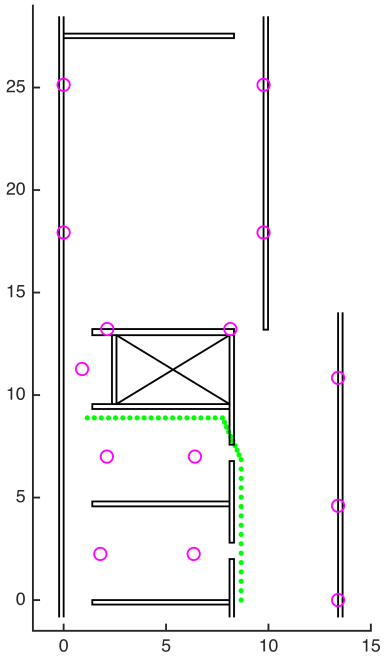


Figure 4.3: Walk through door test

4.4.1 Wall crossing

In this test, the user walks by a wall. The goal of the test is to see if the map model manages to prevent wall crossings and take advantage of the presence of the wall to increase the accuracy. The path walked is illustrated in Figure 4.2.

4.4.2 Walk through door

Since the map-based approach is supposed to prevent wall crossings, this test examines if the model can handle that a door is walked through. The door is walked through by coming up on the side of it and walking diagonally through it. This is considered a more difficult test case than just walking straight through it. Paths straight through doors are available in other test cases though. Figure 4.3 illustrates the path.

4.4.3 Unbounded path

This test examines a straight path, in the middle of a large room, to check if the map matching feature makes any difference. The aim is to investigate if the map matching features could improve the estimation even in large rooms without any nearby walls. The path is illustrated in Figure 4.4.

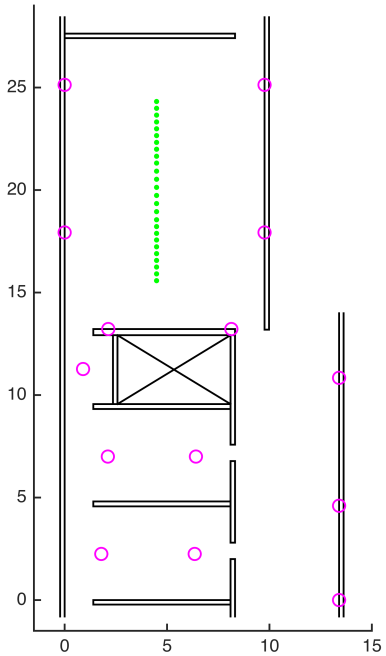


Figure 4.4: Path unbounded by walls

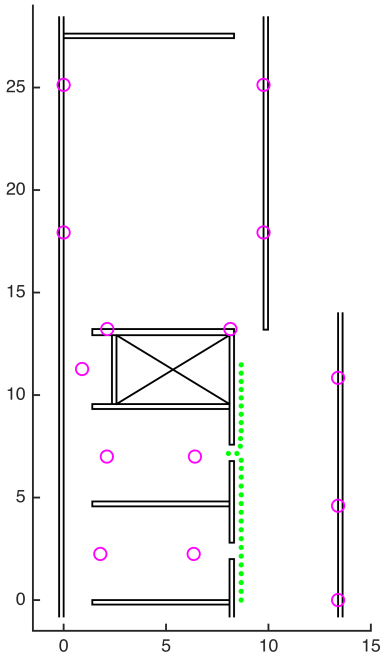


Figure 4.5: Nearly enter room test path

4.4.4 Nearly enter room

In an attempt to fool the system, a straight path is walked by a door. When passing the door, a step is taken into the adjacent room, and then a step back is taken and the previously path is continued. This is made to fool the system and can be considered a worst-case scenario, since the system might believe that the other room was entered. Figure 4.5 illustrates the path walked.

4.4.5 Longer path

A longer path is walked to see how well the indoor positioning system can handle more realistic scenarios where the user navigates through several adjacent rooms. This is the type of behaviour that is expected from a typical user. The test also investigates how the prototype handles any accumulating of potential errors. Figure 4.6 illustrates the walked path.

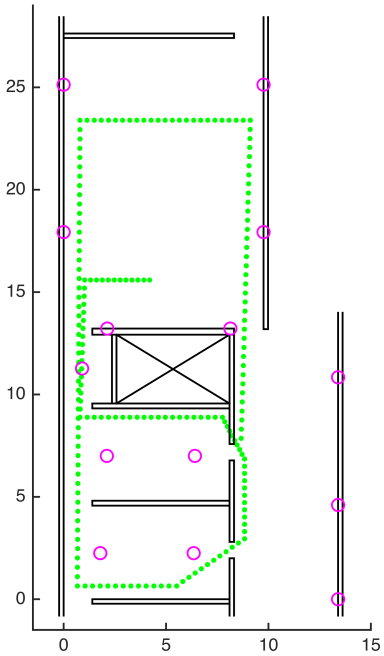


Figure 4.6: Longer path

5
Results

The results of the map-based approach showed an average improvement of 17.8 % in accuracy and also an average improvement of 3.33 % in room correctness compared to the map-less approach. An improvement of 3.33 % in room correctness might seem small but in most test cases was the room correctness already around 90 % to 95 % which means that the error of the room estimation has been reduced between 33 % and 66 %. The minimum and maximum error was also reduced with an average of 26.2 % and 20.4 % which is a significant improvement. The standard deviation showed an average improvement of 4 % but the improvement varied considerably between different test cases.

5.1 The Design of the Prototype

The design uses data from sensors found on a modern, off-the-shelf smartphone. It also uses a map of the premises and calculates the position by using a particle filter. The processing of the sensor information also includes accounting for sensor noise and step detection. The used sensors are the accelerometer, the magnetometer and the Bluetooth low energy chip.

5.1.1 System Architecture

The **accelerometer** delivers acceleration data in the three axes in G, ($\mathrm{G} = 9.81\ m/s^2$) to the **Step_detection** component (Section 5.1.2). The value of each of the axes are added together to get the magnitude of the acceleration and from this value are the steps detected. When the **Step_detection** component detects a step, the current heading from the **Heading_estimation** is sent to the **Prediction_model**. The **Heading_estimation** component itself, gets the current heading by interpreting the magnetic field received from the **magnetometer**.

The **Prediction_model** (Section 5.1.3), is responsible for moving the belief of the particle filter in the direction and with the length of the step. It is this movement that can be restricted by the **Map_matching** component. When the belief has been moved, the **Map_matching** component corrects parts of the belief that is impossible. After the new belief has been created, it is sent to the **Update_model** (Section 5.1.4) who incorporates the distance measurements from the beacons.

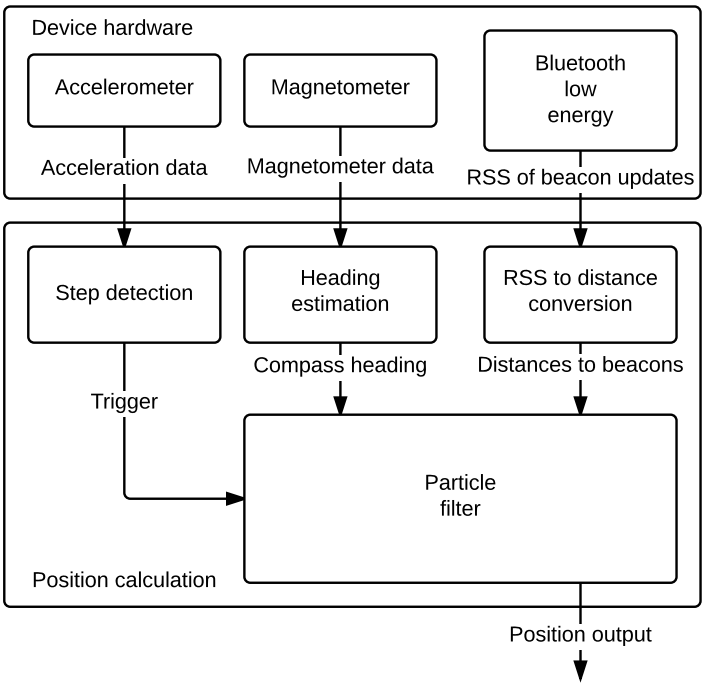


Figure 5.1: The system setup **(Accelerometer, Magnetometer, Bluetooth_low_energy, Step_detection, Heading_estimation, RSS_to_distance, Particle_filter):** the accelerometer samples the acceleration with a frequency of 40 Hz and sends it to the Step_detection component (**Accelerometer**), the magnetometer samples the magnetic field with a frequency of 10 Hz and sends it to the Heading_estimation component (**Magnetometer**), the Bluetooth chip ranges the iBeacons with a frequency of 1 Hz and sends the RSS values to the RSS_to_distance component (**Bluetooth_low_energy**), the Step_detection (Section 5.1.2) component receives acceleration data and trigger the Prediction_model if a step is detected (**Step_detection**), the Heading_estimation component receives magnetic fields readings from the Magnetometer and converts it into a heading estimation. This component is provided by the iOS platform (**Heading_estimation**), the RSS_to_distance component receives the RSS values from the iBeacons rangings and converts them into distances according to Equation 3.3 and the given path loss exponent (**RSS_to_distance**), the particle filter processes the input from the Step_detection, Heading_estimation and RSS_to_distance components and outputs a position estimation. A more thoroughly description is presented in figure 5.2 (**Particle_filter**)

The **Bluetooth_low_energy** chip delivers RSS values combined with the UUIDs from all beacons in range. The RSS values are converted to distance measurement by the **RSS_to_distance** component. The distance measurements are then used to update and weight the belief in the **Update_model** (Section 5.1.4).

After the weight has been updated, the belief is sent the to the **Resample** component where a new set of particles, which represents the belief, is created. The new set is created by, based on the weight of the particles, randomly drawn particles from the old set into the new one. This results in that only the most likely particles survives to the next iteration of the particle filter. To achieve a final position

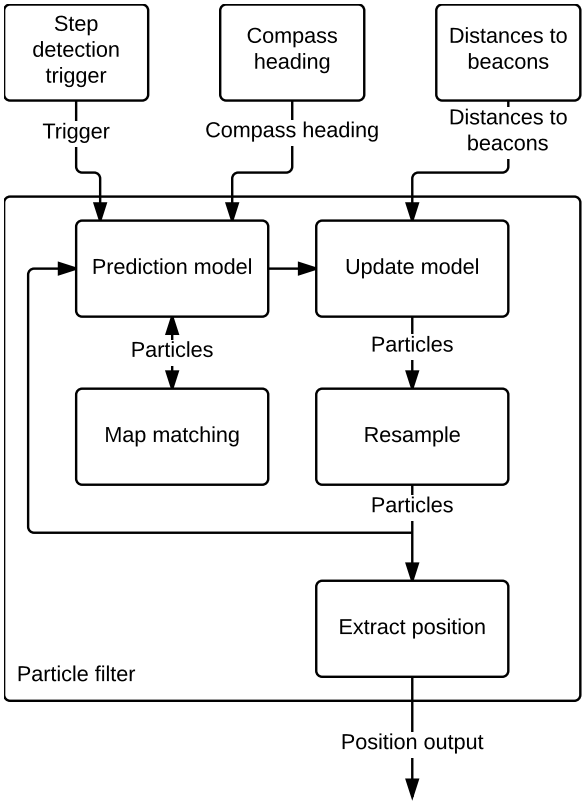


Figure 5.2: The particle filter **(Prediction_model, Update_model, Map_matching, Resample, Extract_position)**: the Prediction_model (Section 5.1.3) moves the belief according the the estimated heading and step length (**Prediction_model**), the Update_model (Section 5.1.4) updates the belief according to the reported distances (**Update_model**), the Map_matching (Section 5.1.3) component corrects the new belief according to the premises (**Map_matching**), the Resample component is responsible for doing the resample step in the particle filter. The Low-variance algorithm provided by [3] is used to realise this component (**Resample**), the Extract_position (Section 5.1.5) component extracts the estimated position from the belief (**Extract_position**)

estimation to be handed to the user, a weighted average is calculated from the new belief by the **Extract_position** component which receives the new belief from the **Resample** component. The average is based on the weight and position of the particles.

5.1.2 Step Detection

The step detection algorithm is used to trigger the prediction model when a step has been detected. When detected, a reading from the magnetometer is also made and sent to the prediction model. However, the detection functionality already exists in the iPhone[1] but tests show that the data is delayed to a degree there it is not possible to use it to trigger the prediction model. Instead, a step detection algorithm

[1]Apple Inc. *CMPedometer*. `https://developer.apple.com/library/prerelease/ios/documentation/CoreMotion/Reference/CMPedometer_class/index.html`. Accessed: 2015-08-07. Aug. 2015.

was implemented from scratch. It uses data from the accelerometer to detect when a step has occurred. As the graph in Figure 5.3 shows, the acceleration caused by the steps are clearly visible. The step detection algorithm's task is just to detect steps, not detect if the user is walking, going by car, running or some other activity.

The algorithm works by receiving data from the accelerometer, which is then low-pass filtered. The algorithm uses two thresholds to keep track of the acceleration. If the acceleration has been above a certain number of G and then drops below another number of G, a step is registered. The thresholds are selected such as even light steps will trigger the step detection but normal use of the phone will not. A pseudocode implementation is illustrated in Algorithm 1. The second plot in Figure 5.3 illustrates the resulting low pass filtered acceleration data and the vertical bars representing where steps have been detected.

Algorithm 1 Step detection algorithm
Input: Readings from the accelerometer at time t
Output: *true* if a step is detected at time t

The step detection algorithm is fed with data from the gyroscope at each time unit t and returns *true* if a step was detected.

Global variables: *has_been_above*
Constants: $threshold_{higher}$, $threshold_{lower}$

```
1:  procedure STEP_DETECTION(acceleration)
2:      if (acceleration >= threshold_higher) then
3:          has_been_above ← 1
4:      end if
5:      if (acceleration < threshold_lower) then
6:          if has_been_above = 1 then
7:              has_been_above ← 0
8:              return true
9:          end if
10:     end if
11:     return false
12: end procedure
```

5.1.2.1 Description of the step detection algorithm

The algorithm is constructed with two if-statements. The first one, Line 2 to 4 changes the global variable *has_been_above* to indicate that the acceleration has been higher than $threshold_{higher}$. The next if statement, Line 5 to 10, checks if acceleration has been below $threshold_{lower}$. If it has, and also has been above $threshold_{higher}$, the characteristics of a step has been achieved and the step is registered.

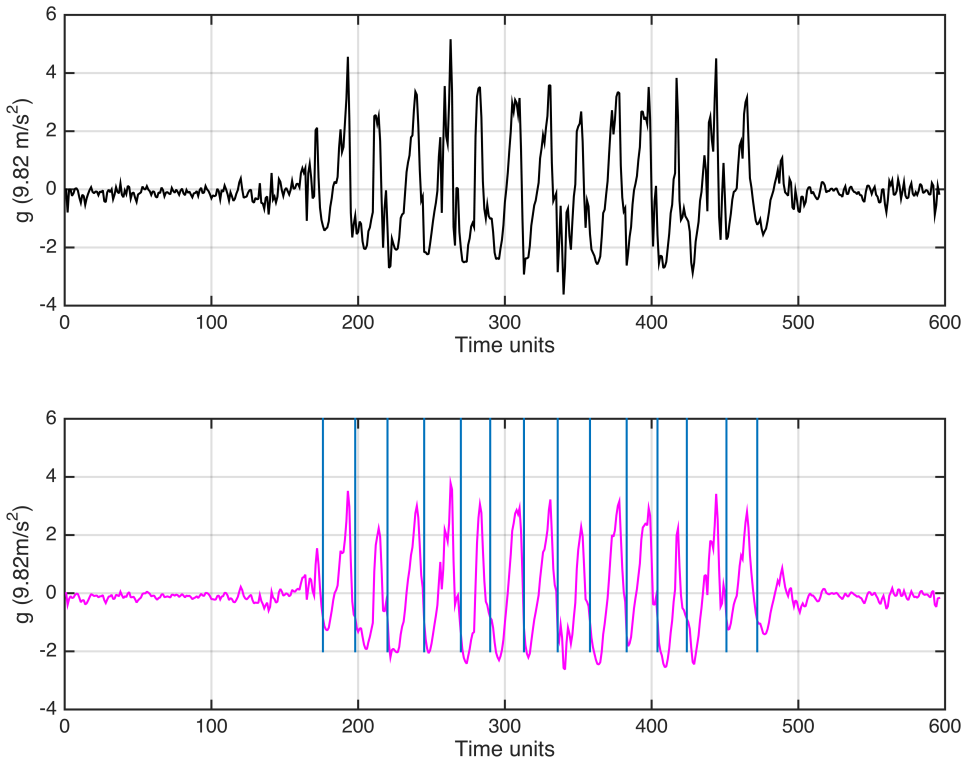


Figure 5.3: The step detection visualised. The vertical lines represent detected steps.

5.1.3 Prediction Model

To observe the potential benefit of introducing a map into the position estimation, two different versions of the prediction model were developed. As mentioned in Section 2.3.1, the motion model is responsible for moving the particles in a particle filter, i.e. the prediction step. Since each particle represents a possible location, it was determined that the map should be incorporated in the prediction model instead of in the update model.

Both models are designed in a similar way, as they both returns a new belief based on the given control and the previous state. To construct the new belief, the models takes the given control and moves the particles, which symbolises the old state, accordingly. Before the movement of each particle, a random noise is added to the control. The noise is added to account for the noise of the sensor and results in every particle moving slightly differently compared to each other. To be able to account for false-positive step detection, both models are given a 20 % chance of not applying the control at all. When all particles have been evaluated, a new belief has been produced.

Pseudocode of the map-less prediction model is presented in Algorithm 2.

The potential improvement of the map-based approach over the map-less approach is that the map-based approach also examines if the particles new positions are possible or not. In other words, the model checks if any particles has crossed any walls. If that is the case, the model slightly changes the noise that was added to the control and tries once again to move the particle according the noisy control. If the particle still crosses any walls after a predefined number retries, the particle is placed somewhere arbitrary and gets an initial weight of zero. As a result of this, the prediction model can recover even if particles gets stuck in a corner which is possible when limiting the movement of the particles with a map. Pseudocode of the map-based prediction model is presented in Algorithm 3.

Algorithm 2 Prediction model, map-less approach
Input: a set of particles, the control(consisting of step length and heading)
Output: a set of updated particles

The motion model takes all the particles and moves them in the direction of the step the user takes.

Global variables: –
Constants: *length_deviation*, *heading_deviation*, $false\_positive\_ratio$

```
1:  procedure PREDICTION_MODEL(particles, u_t)
2:      for each particle in particles do
3:          length ← u_t.length + add_noise(length_deviation)
4:          heading ← u_t.heading + add_noise(heading_deviation)
5:
6:          r ← rand()                    ▷ Returns random number between 0 and 1
7:          if r > false_positive_ratio then
8:              move(particle, length, heading)
9:          end if
10:     end for
11:
12:     return particles
13: end procedure
```

5.1.3.1 Description of the map-less prediction model algorithm

Line 3 decides what length the specific particle is going to move according to. The length is an elongated average step length, since some particles are not moved, see Line 6 and 7 below. The function **add_noise** adds noise to the step length by drawing a random sample from a normal distribution with mean 0 and standard deviation *length_deviation*. In Line 4, the heading, from the magnetometer, undergoes a similar procedure.

Line 6 and 7 has the functionality to skip updating $false\_positive\_ratio$ particles. This is since some steps detected may be false positives, and this thereby leaves some

particles to cover up if a false positive detection occurs. The rest of the particles are moved by the function **move**, a function that moves the particle according to *length* and *heading*.

5.1.3.2 Description of the map-based prediction model algorithm

The code in Algorithm 3 is equal to Algorithm 2 up to line 7. Algorithm 3 also uses the **move** function, but checks if the new location of the particle is valid. The check is made with the function **possible_location** which checks if the particle is placed in a possible location. If it is not, the movement is undoed in Line 13. Line 14 to 20 holds an if-statement which tries to move the particle into a possible position *count* number of times. If a possible position is not found during *count* iterations, the particle is not moved. Since every iteration takes time, the algorithm is slowed down if a too high *count* is chosen.

5.1.4 Update Model

The update model is responsible for determining how correct the positions of the particles are. Similar to trilateration, the update model uses distances to accomplish this task. The main task of the model is account for the deviation and noise that the rangings suffers from. As stated in Table 3.2, the deviation rather quickly increases from a few centimeters to several meters when the distance increases.

To partly solve the problem with the deviation, the model only relies on the four closest beacons. This results in the model only uses rangings with the lowest possible deviation and therefore decreases the impact of the high deviation. The number beacons the model relies on depends on the environment and how close beacons are placed to each other.

The purpose of the update model is to determine how correct a particle is compared to the real world. The comparison is also referred to as weighing of the particles, where the weight symbolise how correct a particle is. The weighing is divided into two separate stages where the first stage calculates the theoretical distance d_t to the nearby beacons from each particle. The second stage compares the theoretical distance to the reported distance d_r from the beacons by taking $d_t - d_r$. The result from the subtraction is then used as an index in a probability density function operating on a normal distribution with a mean equal to zero. This results in, the smaller the difference is between the theoretical and the reported distance, a higher value is returned from the probability density function. The returned value from the probability density function is then used as weight and the total weight of a particle is the sum of all weights divided by the number of used beacons.

5.1.4.1 Description of update model algorithm

Line 2 sorts the received distance measurements in ascending order. This is made because only the beacons most nearby will be used. In Line 4, all the particles are

Algorithm 3 Prediction model, map-based approach
Input: a list of particles, the control(consisting of step length and heading), a map of the premises
Output: updated particles

The motion model takes all the particles and moves them in the direction of the step of the user. The movement of the particles is controlled by the map matching, which also is hosted in the motion model.

Global variables: –
Constants: *length_deviation*, *heading_deviation*, $false\_positive\_ratio$, *retries*

```
1: procedure PREDICTION_MODEL(particles, u_t, map)
2:     for each particle in particles do
3:         length ← u_t.length + add_noise(length_deviation)
4:         heading ← u_t.heading + add_noise(heading_deviation)
5:
6:         r ← rand()                    ▷ Returns random number between 0 and 1
7:         if r > false_positive_ratio then
8:             count ← 0
9:             while true do
10:                move(particle, length, heading)
11:                p ← possible_location(particle, map)
12:                if p = false then
13:                    undo_movement(particle, length, heading)
14:                    if count = retries then
15:                        break                    ▷ Do not move particle
16:                    else
17:                        length ← u_t.length + add_noise(length_deviation)
18:                        heading ← u_t.heading + add_noise(heading_deviation)
19:                        count ← count + 1
20:                    end if
21:                else
22:                    break
23:                end if
24:            end while
25:        end if
26:    end for
27:
28:    return particles
29: end procedure
```

Algorithm 4 Update model
Input: a list of particles, RSS values from beacons in range
Output: modified particles

The sensor model updates the particles according to the measurements received from the beacons.

Global variables: –
Constants: *number_of_beacons*

```
1:  procedure UPDATE_MODEL(particles, z_t)
2:      sort(z_t, 'ascending distance')
3:
4:      for each particle in particles do
5:          weight ← 0
6:          for i ← 1 to number_of_beacons do
7:              position ← get_beacon_position(z_t^i.id)
8:              distance ← dist2(position, particle)
9:              variance ← (p_0^2 · distance + p_1 · distance + p_2)
10:             weight ← weight + prob2(z_t^i.real_dist − distance, variance)
11:         end for
12:         particle.weight ← weight / number_of_beacons
13:     end for
14:
15:     return particles
16: end procedure
```

looped through. Each particle is having its weight reset and in the next for loop, on Line 6, the particle is having its weight set again.

The new weight is calculated based on its position in relation to the position of the *number_of_beacons* closest beacons. Throughout the testing, 4 beacons have proven to supply the best balance between performance and speed. The function **get_beacon_position** looks up and returns the position of the beacon being inputted to it. The function **dist2** calculates and returns the distance between the beacon and the particle. The distance is used to derive the variance(line 9) which is one of the two inputs to **prob2**. The other input is the difference between the real and the calculated distance. **prob2** returns a value from a probability density function, operating on a normal distribution with a mean of zero and the supplied variance. The second parameter is used as an index and the return value represent the weight, given by that beacon. A particles total weight is then calculated on line 12 as the sum of all weights divided by the number of used beacons.

5.1.5 Extract position

To get any useful data form a particle filter, there needs to be some sort of mechanism that can extracts the information from the belief. As the belief consists of a finite number of particles, where each particles holds a fraction of the wanted information, each particle needs to be included in the extraction.

There are two methods of extracting the information from the belief, where one way is to use the position of each particle and derive an average of all positions to get an estimation. The other method is to also include the weight of each particle when deriving the average. This gives a better estimation because only the position of a particle does not contain any information about how good the estimation is.

5.1.5.1 Description of Extract position algorithm

Line 3 loops trough every particle in the given set as every particle needs to be evaluated. On the lines 4-5, the current particle's weight is multiplied with its position and saved during each iteration of the loop. Line 6 sums up the total weight of all particles. Line 8 and 9 calculates the weighted average for both the x and y coordinate.

Algorithm 5 Extract position
Input: a set of particles
Output: a position estimation (x,y)

Takes a set of particles and extracts the position estimation from it.

Global variables: –
Constants: –

```
1: procedure EXTRACT_POSITION(particles)
2:     weight_x, weight_y, weight ← 0
3:     for each particle in particles do
4:         weight_x ← weight_x + (particle.x · particle.weight)
5:         weight_y ← weight_y + (particle.y · particle.weight)
6:         weight ← weight + particle.weight
7:     end for
8:     x ← weight_x / weight
9:     y ← weight_y / weight
10:    return (x, y)
11: end procedure
```

5.2 Validation of the Indoor Positioning System

Due to the probabilistic nature of the algorithms that is used, it is difficult to present a result that reflects the true performance of the system. Even if the same data is fed to the system, the results will vary. A way to minimise this problem is to feed the system with the same data a multiple number of times and derive an average of the results. The average will then give a good representation of the performance. The presented results is based on an average calculated from ten different runs. The improvement factor, defined as the map-less value divided by the map-based value, is also presented. The greater the improvement is, the greater the map-based approach performed, compared to the map-less approach. If the improvement factor is below 1, the map-less approach performed better.

A set of test cases is defined in Chapter 4 which each one evaluates different aspects of the system. The longer path test case represents, to a greater extent than the others, a realistic use of the system. Therefore, a greater focus will be on the results from that test case where the others are mentioned briefly. A complete presentation of the results can be found in Appendix A.

5.2.1 Wall crossing test results

The map-based approach shows an improvement over the map-less approach when moving close to walls. The most evident improvement is the error in room correctness, which decreases from 9 % to 3 % in this test, i.e. a 67 % decrease in error. The other values are very similar as can be seen in Table 5.1. When looking at Figure A.2, it is visible how the map-less approach allows the position to be estimated in the adjacent room more often compared to the map-based approach. Both version do however estimate in the wrong room when being close to the door. The map-based approach allows this since the positioning estimation must be allowed to enter through doors.

Table 5.1: Results of wall crossing test

	Map-less	**Map-based**	**Improvement factor**
Average error [m]	1.54	1.50	1.03
Min error [m]	0.24	0.27	0.89
Max error [m]	3.27	3.26	1.00
Standard deviation [m]	0.82	0.82	1.00
Correct room estimation	91%	97%	

The distribution of the error for both the map-based and map-less approach is shown in Figure 5.4.

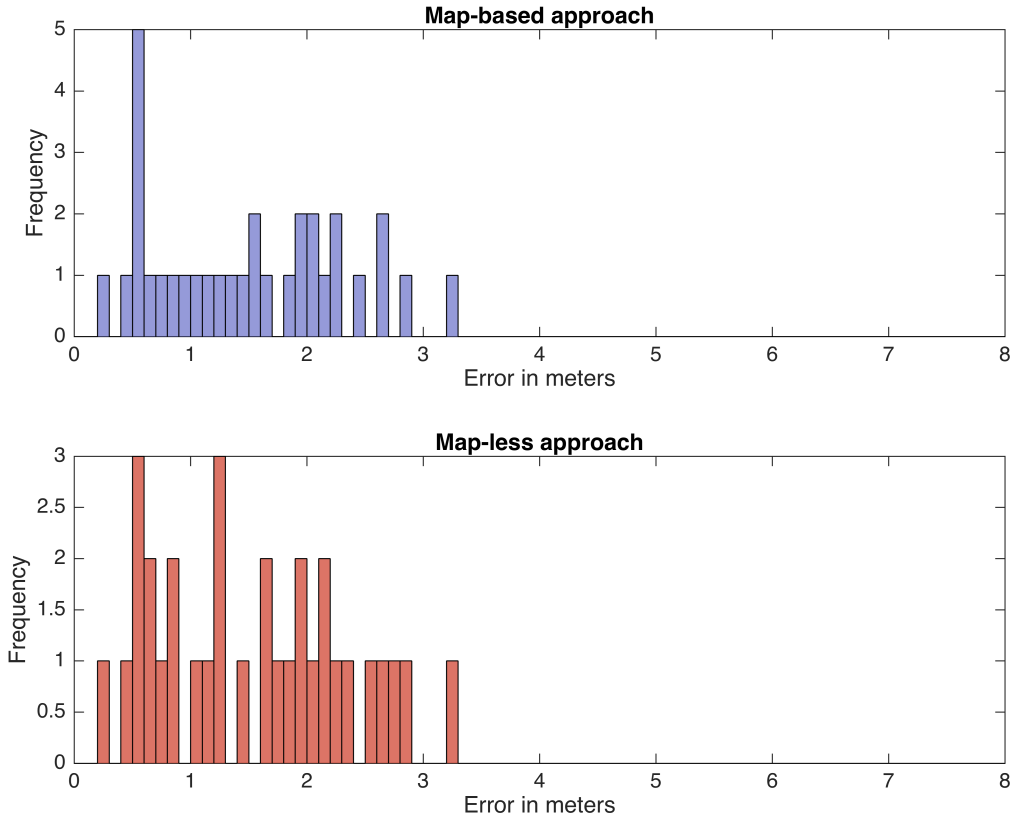


Figure 5.4: The distribution of errors in the wall crossing test

5.2.2 Walk through door test results

In the walk through door test, the map-based approach performs better than the map-less approach. Its standard deviation is lower, which means it delivers a more consistent result. This is clearly visible in the maximum value, where the map model lowers the most erroneous estimation with a meter. The correct room estimation is the same for both versions. Figure A.4 is indicating that the transition to one room to the other occurred at the same time for both versions, which means that the map model does not prevent that a user moves through a door. The result is listed in Table 5.2.

Table 5.2: Results of walk through door test

	Map-less	**Map-based**	**Improvement factor**
Average error [m]	2.50	2.42	1.03
Min error [m]	1.35	1.23	1.10
Max error [m]	5.51	4.40	1.25
Standard deviation [m]	0.75	0.58	1.29
Correct room estimation	90%	90%	

The distribution of the error for both the map-based approach and map-less approach is shown in Figure 5.5.

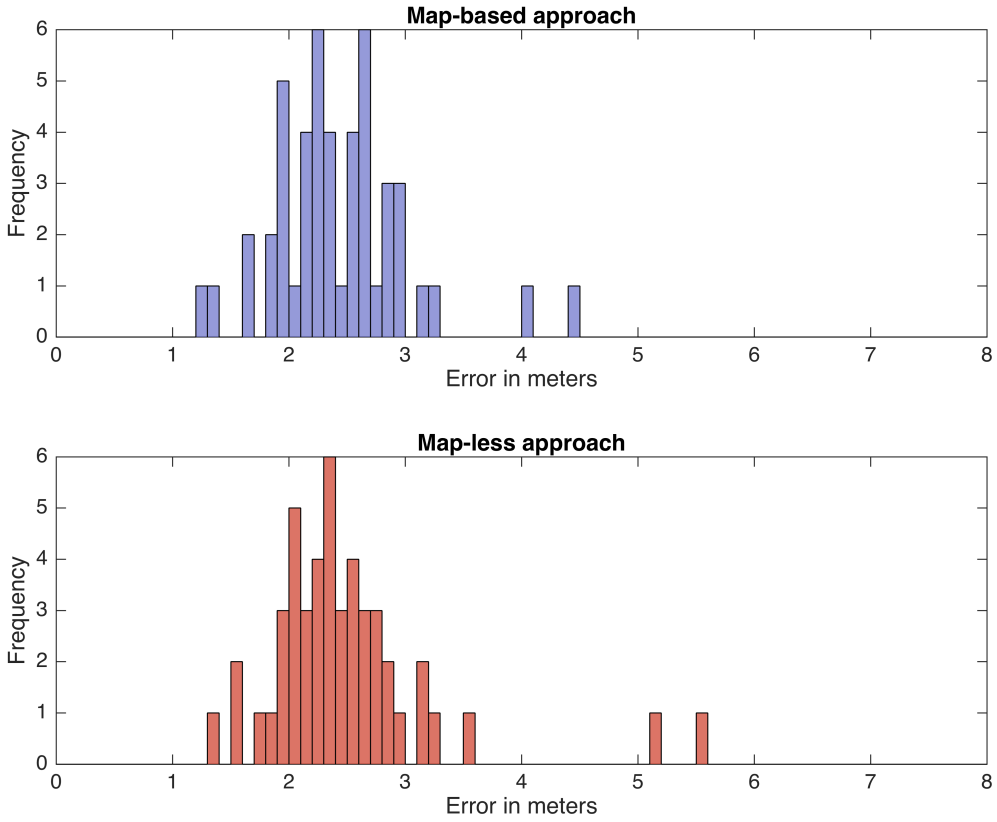


Figure 5.5: The distribution of errors in the walk through door test

5.2.3 Unbounded path test result

This test produced an interesting result. The map-less approach has a lower standard deviation than the map-based approach, however, when comparing the average, it is visible that the map-based approach has a far better accuracy. Figure 5.6 and Figure A.6 also clearly illustrates the outcome. The map model reduces the average error with over a meter, a 41 % change. The correct room estimation is at 100 % for both versions, signifying that no estimated position was in the wrong room. The goal of the test is to make sure that the map-based approach also works when being far away from walls. Table 5.3 presents the numbers.

Table 5.3: Results of unbounded path test

	Map-less	Map-based	Improvement factor
Average error [m]	2.66	1.56	1.70
Min error [m]	2.03	1.00	2.03
Max error [m]	3.07	2.50	1.23
Standard deviation [m]	0.24	0.50	0.50
Correct room estimation	100%	100%	

The distribution of the error for both the map-based approach and map-less approach is shown in Figure 5.6.

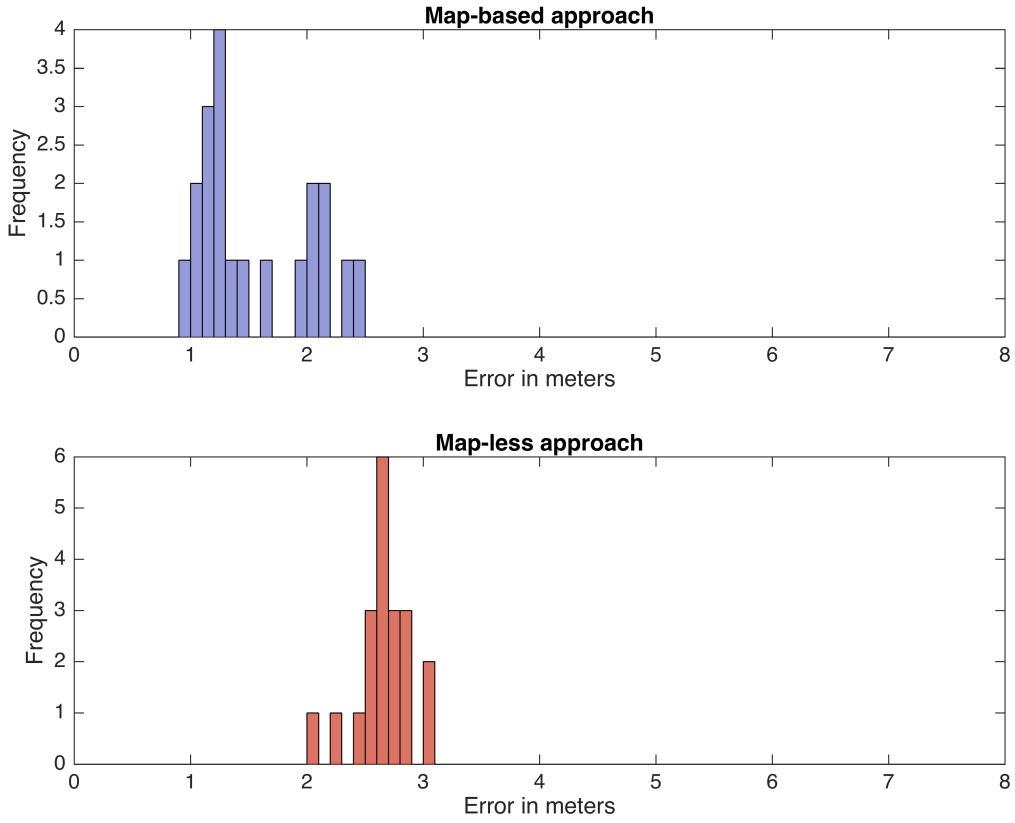


Figure 5.6: The distribution of errors in the unbounded path test

5.2.4 Nearly enter room test result

The main goal of this test is to exploit the underlying functionality of the map-based approach. When entering a room, particles will be spread inside that room. During this test, just a small step was taken into the room, enough to spread and trap particles inside, and then continue the path outside the room. Both versions handled the scenario well since the position estimation was quite far from the adjacent room, resulting in not many particles spreading in the room. The accuracy and room correctness values are listed in Table 5.4.

Table 5.4: Results of nearly enter room test

	Map-less	**Map-based**	**Improvement factor**
Average error [m]	2.01	2.05	0.98
Min error [m]	0.86	0.85	1.01
Max error [m]	4.52	3.99	1.13
Standard deviation [m]	0.80	0.87	0.91
Correct room estimation	100%	100%	

The distribution of the error for both the map-based approach and map-less approach is shown in Figure 5.7.

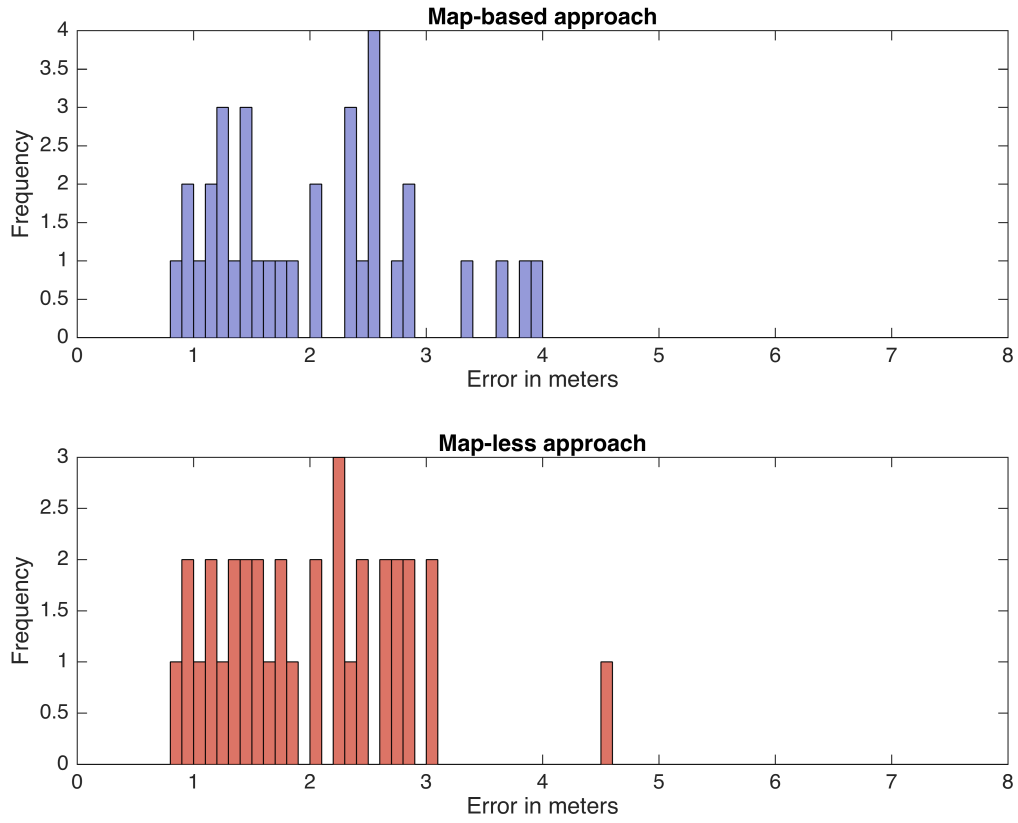


Figure 5.7: The distribution of errors in the nearly enter room test

5.2.5 Longer path test result

While walking a longer, more realistic, path, the map-based approach shows an improvement over the map-less approach. The average error is reduced by nearly half a meter, 13 %. The maximum error is seeing a decrease of over two meters. The standard deviation is also lower, meaning that the map model produces a more consistent result, supporting the other values. The improvement is also visible in the upper part of Figure 5.8 where the absolute error is shown for each time unit. Figure 5.10 is a graphical representation on a map which compares the different versions. Most notable difference in Figure 5.10 is the map-less estimation which takes a shortcut over the non-accessible area in the middle of the figure.

Table 5.5: Results of longer path test

	Map-less	**Map-based**	**Improvement factor**
Average error [m]	3.37	2.94	1.15
Min error [m]	1.03	0.78	1.32
Max error [m]	7.79	5.51	1.41
Standard deviation [m]	1.63	1.12	1.46
Correct room estimation	65%	68%	

The distribution of the error for both the map-based and map-less approach is shown in Figure 5.9. The estimated paths and the real path are visible in Figure 5.10.

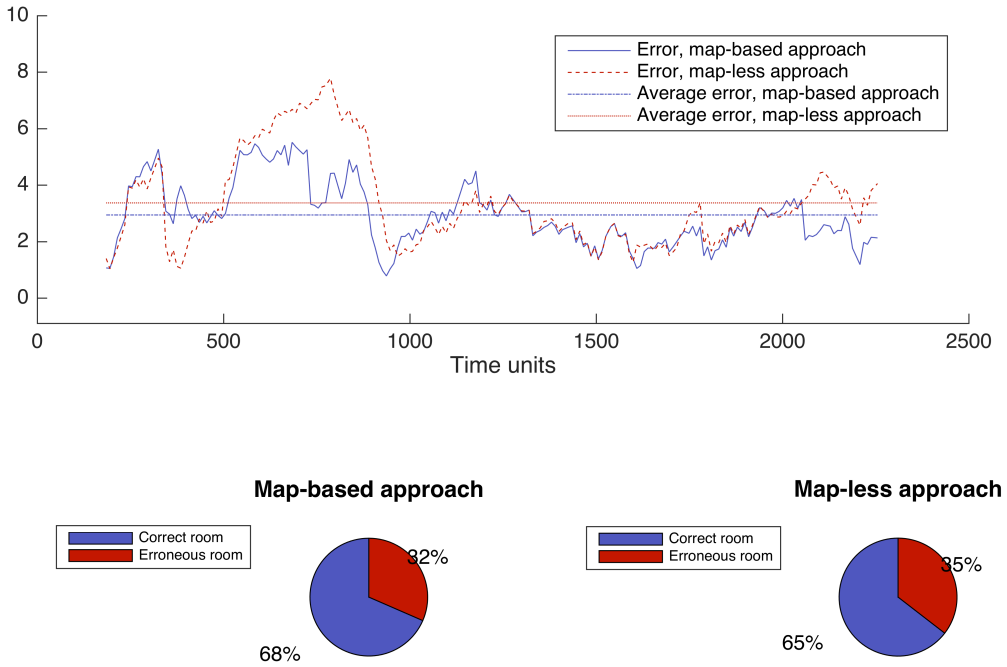


Figure 5.8: Error and correct room estimation of longer path test

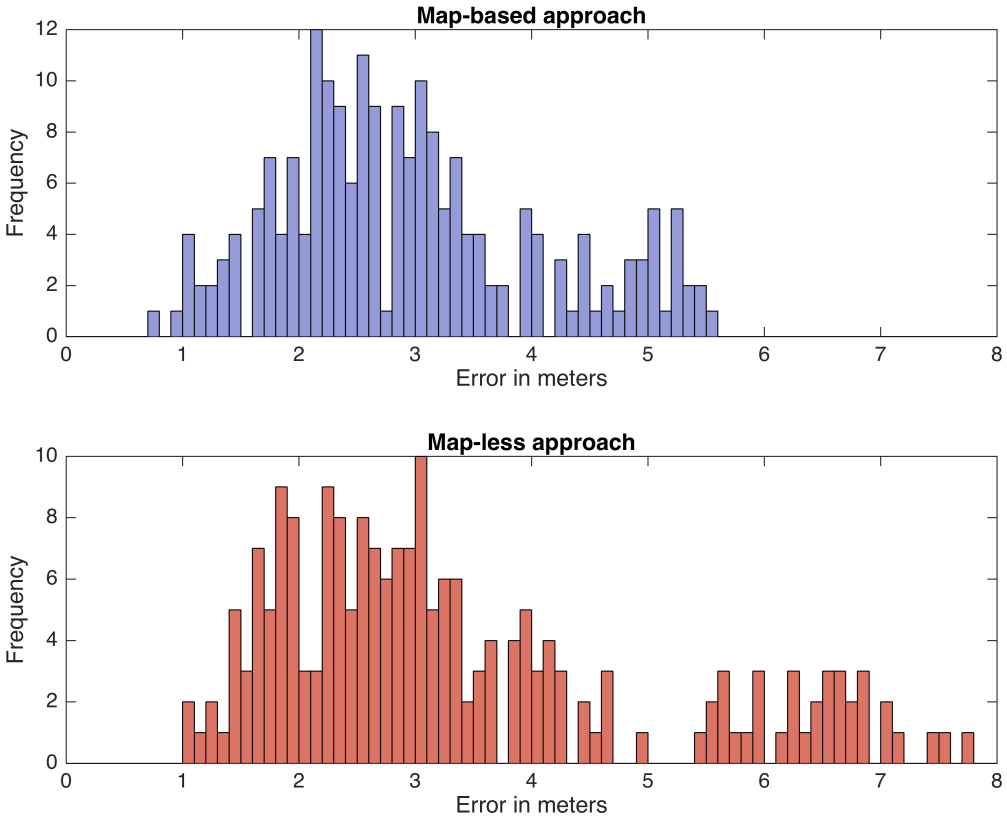


Figure 5.9: The distribution of errors in the longer path test

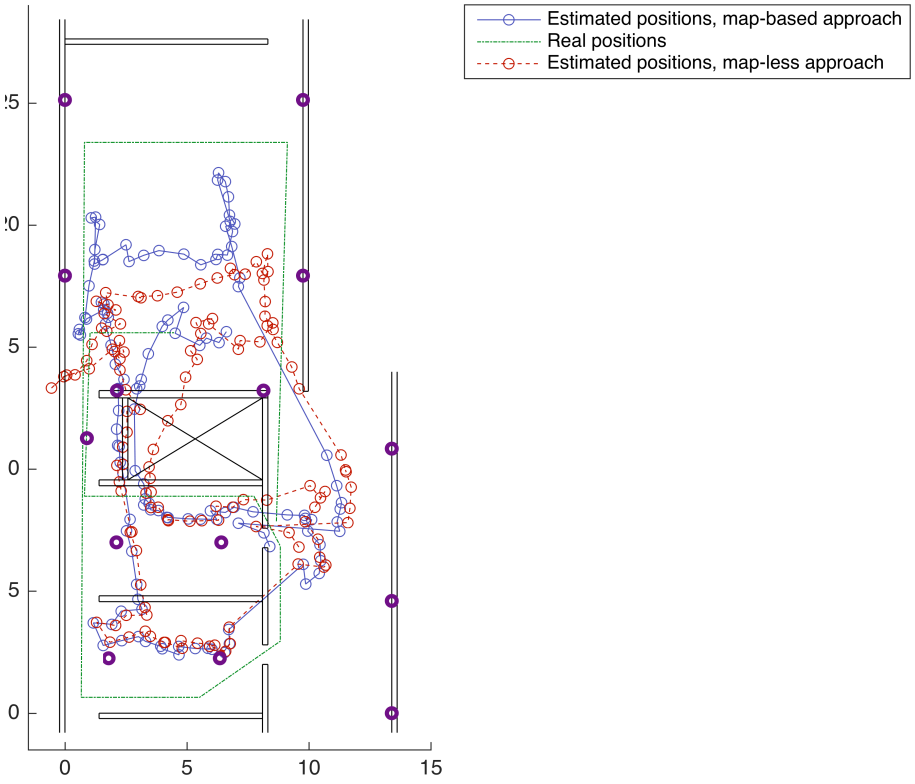


Figure 5.10: Graphical representation of the estimated positions

6
Discussion

A design of an indoor positioning system is presented that includes a step detection model as well as a particle filter that base the position estimation with a map-based approach. The design utilises sensors found on a modern, off-the-shelf, smartphone and knowledge of the current premises. A pilot of the design has been implemented in MATLAB and the validation of the pilot shows an improvement compared to the map-less approach.

6.1 Implementation

The data gathering was made on an iPhone 5S, a platform having some limitations in the ability to collect data from the different sensors. For example, the frequency of collecting beacon updates was limited and the different functions for retrieving sensor updates from the accelerometer and the magnetometer did not come with any in-depth documentation on how the data was processed. The general perception is that the data is filtered and processed in different ways before reaching the client. If the data is filtered, this may degrade the performance of the algorithm compared to when designing all the filtering by oneself. Filtering the data twice is also a potential waste of CPU cycles.

A design for each of the models was selected and implemented, leaving alternative designs out. In the case of the map-based prediction model, a design using map matching that did not allow particles to move to impossible positions was selected. Another design that was considered was by constructing the most common routes through the different rooms and lock the user to them. This idea is similar to how a GPS works in a car, where the car is locked to the closest road because a car can not be driven outside a road. However, this design has some obvious flaws like not being able to position the user outside of the routes, but, while traversing a room in the common way, there is a possibility that the accuracy had increased. The outcome could have been completely different compared to the selected design but it would also introduce a new problem, the construction of the routes.

A design improvement that was tested was the ability to calculate which beacons that was behind the user and therefore not being in line-of-sight of the smartphone. The values of those beacons were supposed to be trusted less due to the RSS values being decreased by the obstructing body of the user. However, the heading estima-

tion was not accurate enough to ensure the correctness of such calculations and was therefore not included in the proposed design. If it was possible to achieve a more accurate heading estimation, this approach might have improved the performance of the proposed design.

6.2 Validation

The validation of the pilot showed a clear improvement in accuracy in the map-based approach compared to the map-less approach in most test cases. However, in the nearly enter room test case, Section 5.2.4, there was a degradation of 2 % in accuracy. This degradation is however negligible since it is just a matter of a few centimetres compared to the map-less approach. The degradation could possibly be due to that a particle filter uses probabilistic models where the output could differ between different runs. The probabilistic models is at the same time also the strength of the design, because without it, it would not be possible to account for the noise that is experienced by the sensors.

In the longer path, Section 5.2.5, the room correctness is much worse than what was initially expected, at least in the map-based approach. This might however depend on that the prediction model, Section 5.1.3, uses an odometry model that is based on a control that has happened and not what is going to happen. This might be the reason why the position estimation is a bit behind while moving which is likely to affect the room correctness in a negative way.

A notable observation can be made from the results from the unbounded path test case, Section 5.2.3, where the biggest improvement of accuracy can be seen amongst all test cases. This was the test case where it was most likely to see a small improvement or non at all due to the great distances to any adjacent walls. It appears that the map-based approach not only improves the performance in smaller, closed space as initially thought but also in bigger, more open spaces too. This could be due to the fact that the available sensors on the smartphone are so inaccurate, that in even bigger rooms there is a need of restraining the movement of the belief in the prediction step.

6.3 Test cases

The tests were conducted in a static environment to see the potential increase in performance without any interference affecting the results. This means that no dynamic environment were evaluated and it is therefore not possible to conclude how well the proposed design would perform when there, for example, is other people present. The reasons why this was not considered during the testing was that it is difficult to control the environment and also make the testing repeatable. Though, by looking at the results from the test that were conducted, it seems that it is correct to assume that a dynamic environment would not degrade the performance of the

map-based approach relatively the map-less approach. However, actual testing needs to be conducted in a dynamic environment to solidify this assumption.

7
Conclusion

The map-based approach showed an improvement in both accuracy and room correctness compared to the map-less approach. It also showed a clear improvement in the maximum and minimum estimation error. However, the developers who choose to implement the map-based approach should be aware of the increased costs in computational demand and power consumption of the design when developing applications. This work facilitate the use of a map-based approach in future smartphone platforms that are not restricted by computational or energy constraints.

The extension of this work include further testing of the map-based approach and comparing it to the map-less in future smartphone.

7.1 Further extension

Some thoughts that arose during the work and was not further investigated, was if the placement of the beacons affects the performance of the positioning system. If the placement affects the performance, will the performance be the same for both the map-based and map-less approach. It seems that it is worth investigating if it is better to focus on the placement of the beacons instead of trying to improve the design of the systems to gain performance.

Another possible approach to improve the performance that is worth investigating is if it is better to lock the user to predefined routes as mention in the discussion, Section 6.1. However, this approach raises some other problems that needs to be addressed. Some of these are how to create optimal routes throughout different rooms and how to handle temporary changes of the environment. A possible solution to these problems is to utilise big data and do an analysis of the moving pattern from the users and constantly update the routes according to the observed pattern.

Bibliography

[1] R. E. Kalman. "A new approach to linear filtering and prediction problems". In: *Journal of Fluids Engineering* 82.1 (1960), pp. 35–45.

[2] N. Gordon, D. Salmond, and A. Smith. "Novel approach to nonlinear/non-Gaussian Bayesian state estimation". In: *Radar and Signal Processing, IEE Proceedings F* (1993).

[3] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[4] C. K. Schindhelm, F. Gschwandtner, and M. Banholzer. "Usability of apple iphones for inertial navigation systems". In: *Personal Indoor and Mobile Radio Communications (PIMRC), 2011 IEEE 22nd International Symposium on*. IEEE. 2011, pp. 1254–1258.

[5] S.-H. Won, W. W. Melek, and F. Golnaraghi. "A Kalman/particle filter-based position and orientation estimation method using a position sensor/inertial measurement unit hybrid system". In: *Industrial Electronics, IEEE Transactions on* 57.5 (2010), pp. 1787–1798.

[6] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara. "A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU". In: *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*. IEEE. 2009, pp. 37–42.

[7] A. Perttula, H. Leppakoski, M. Kirkko-Jaakkola, P. Davidson, J. Collin, and J. Takala. "Distributed indoor positioning system with inertial measurements and map matching". In: *Instrumentation and Measurement, IEEE Transactions on* 63.11 (2014), pp. 2682–2695.

[8] S. S. Saab and Z. S. Nakad. "A standalone RFID indoor positioning system using passive tags". In: *Industrial Electronics, IEEE Transactions on* 58.5 (2011), pp. 1961–1970.

[9] M. M. Saad, C. J. Bleakley, T. Ballal, and S. Dobson. "High-accuracy reference-free ultrasonic location estimation". In: *Instrumentation and Measurement, IEEE Transactions on* 61.6 (2012), pp. 1561–1570.

[10] F. Zampella, A. R. Jimenez Ruiz, and F. Seco Granja. "Indoor Positioning Using Efficient Map Matching, RSS Measurements, and an Improved Motion Model". In: *Vehicular Technology, IEEE Transactions on* 64.4 (2015), pp. 1304–1317.

[11] H. Bao and W.-C. Wong. “An indoor dead-reckoning algorithm with map matching”. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*. IEEE. 2013, pp. 1534–1539.

[12] H. Liu, H. Darabi, P. Banerjee, and J. Liu. “Survey of wireless indoor positioning techniques and systems”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 37.6 (2007), pp. 1067–1080.

[13] F. Thomas and L. Ros. “Revisiting trilateration for robot localization”. In: *Robotics, IEEE Transactions on* 21.1 (2005), pp. 93–101.

[14] R. Jirawimut, P. Ptasinski, V. Garaj, F. Cecelja, and W. Balachandran. “A method for dead reckoning parameter correction in pedestrian navigation system”. In: *Instrumentation and Measurement, IEEE Transactions on* 52.1 (2003), pp. 209–215.

[15] Futuremark. *Best Smartphones and Tablets*. `http://www.futuremark.com/hardware/mobileice_storm_unlimited/filter/androidioswinrtwindowsshowall`. Accessed: 2015-07-27. June 2015.

[16] Apple Inc. *Core Motion Framework Reference*. `https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CoreMotion_Reference/index.html`. Accessed: 2015-04-01. Sept. 2013.

[17] Apple Inc. *CMGyroData*. `https://developer.apple.com/library/ios/documentation/CoreMotion/Reference/CMGyroData_Class/index.html#//apple_ref/occ/instp/CMGyroData/rotationRate`. Accessed: 2015-08-15. May 2010.

[18] Apple Inc. *Core Location Framework Reference*. `https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework/index.html`. Accessed: 2015-07-01. Sept. 2013.

[19] Apple Inc. *Getting Started with iBeacon*. `https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf`. Accessed: 2015-03-31. June 2014.

[20] Estimote Inc. *API Documentation*. `http://estimote.com/api/`. Accessed: 2015-03-24.

[21] Bluetooth SIG, Inc. *Bluetooth Fast Facts*. `http://www.bluetooth.com/Pages/Fast-Facts.aspx`. Accessed: 2015-05-05.

[22] J. Rodas, C. J. Escudero, and D. I. Iglesia. “Bayesian filtering for a bluetooth positioning system”. In: *Wireless Communication Systems. 2008. ISWCS'08. IEEE International Symposium on*. IEEE. 2008, pp. 618–622.

[23] L. Wang, C. Hu, J. Wang, L. Tian, and M. Q.-H. Meng. “Dual-modal indoor mobile localization system based on prediction algorithm”. In: *Information and Automation, 2009. ICIA'09. International Conference on*. IEEE. 2009, pp. 236–241.

[24] E. Dahlgren and H. Mahmood. “Evaluation of indoor positioning based on Bluetooth Smart technology”. MA thesis. Chalmers University of Technology.

[25] Estimote Inc. *What are the characteristics of Beacons' signal?* https://community.estimote.com/hc/en-us/articles/201029223-What-are-the-characteristics-of-Beacons-signal-. Accessed: 2015-05-06.

[26] Apple Inc. *CMPedometer*. https://developer.apple.com/library/prerelease/ios/documentation/CoreMotion/Reference/CMPedometer_class/index.html. Accessed: 2015-08-07. Aug. 2015.

A

Appendix 1

Omitted Result Figures

This appendix presents the result figures and plots.

Wall crossing test result

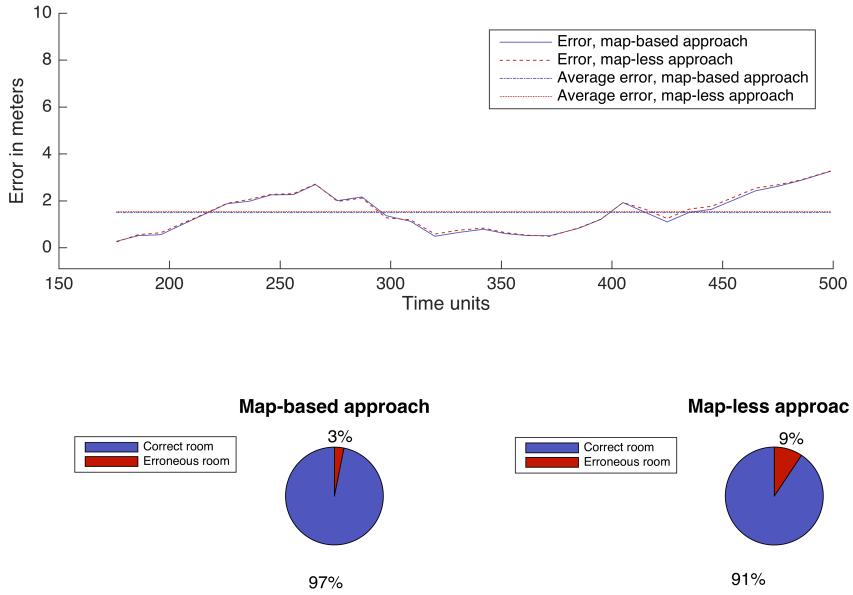


Figure A.1: Average error and correct room estimation plots

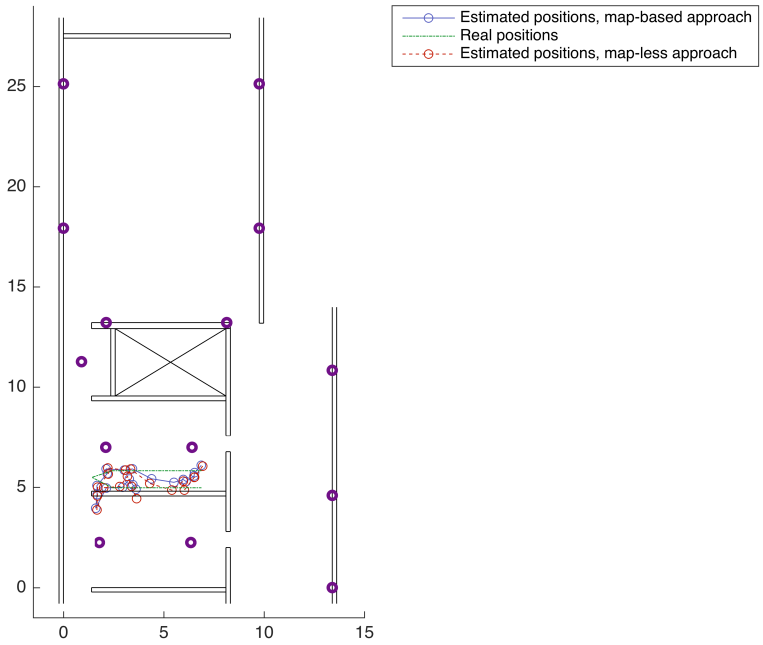


Figure A.2: Real and estimated paths

Walk through door test result

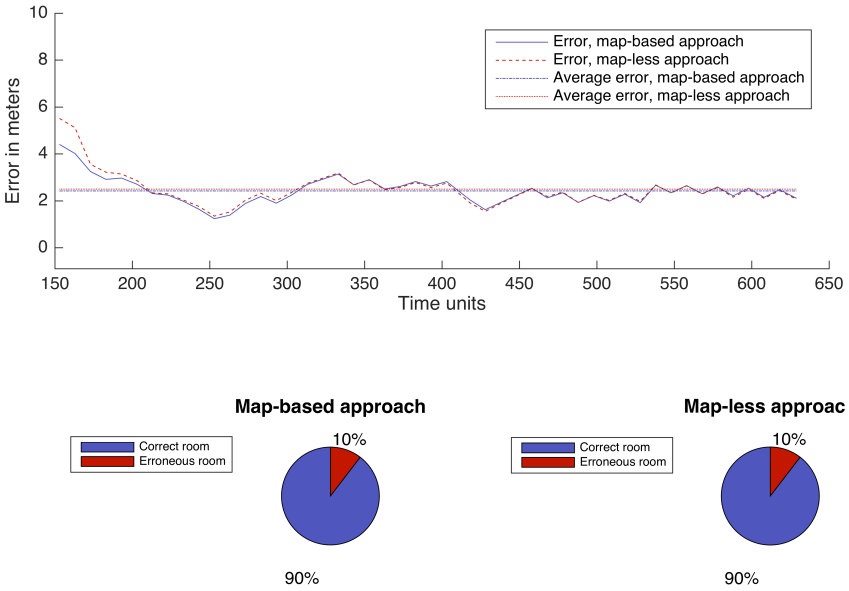


Figure A.3: Average error and correct room estimation plots

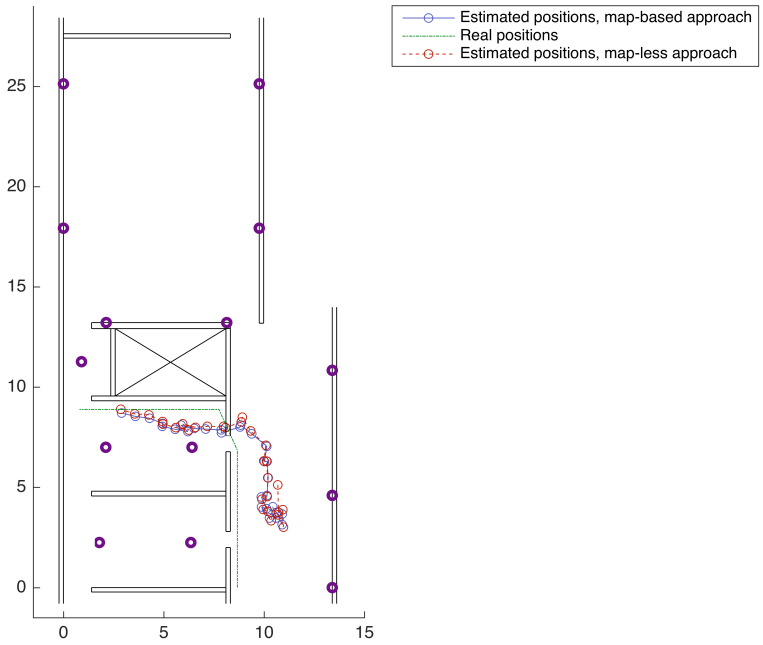


Figure A.4: Real and estimated paths

Unbounded path test result

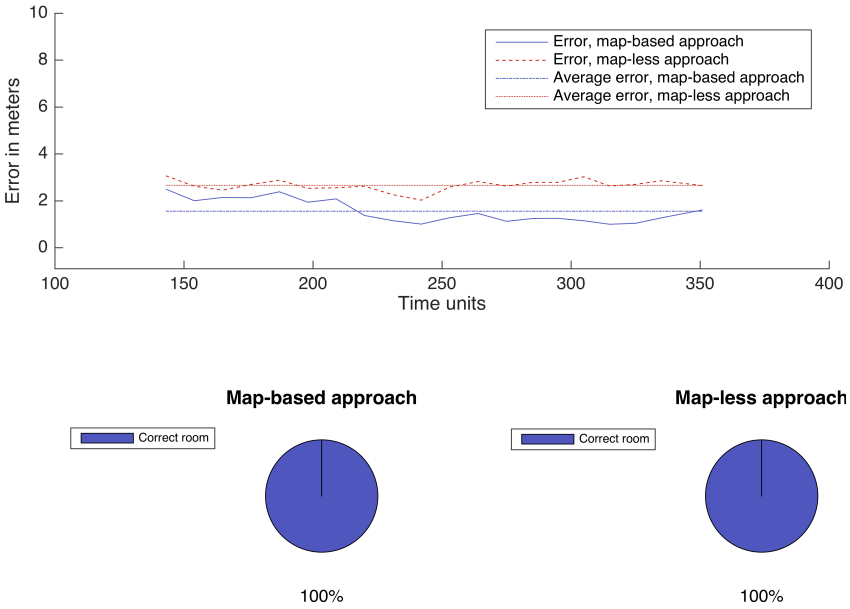


Figure A.5: Average error and correct room estimation plots

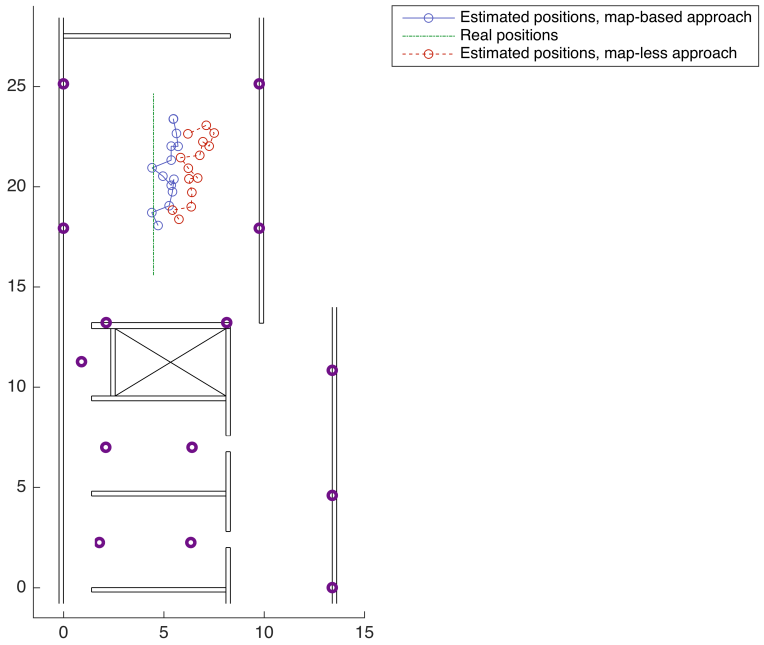


Figure A.6: Real and estimated paths

Nearly enter room test result

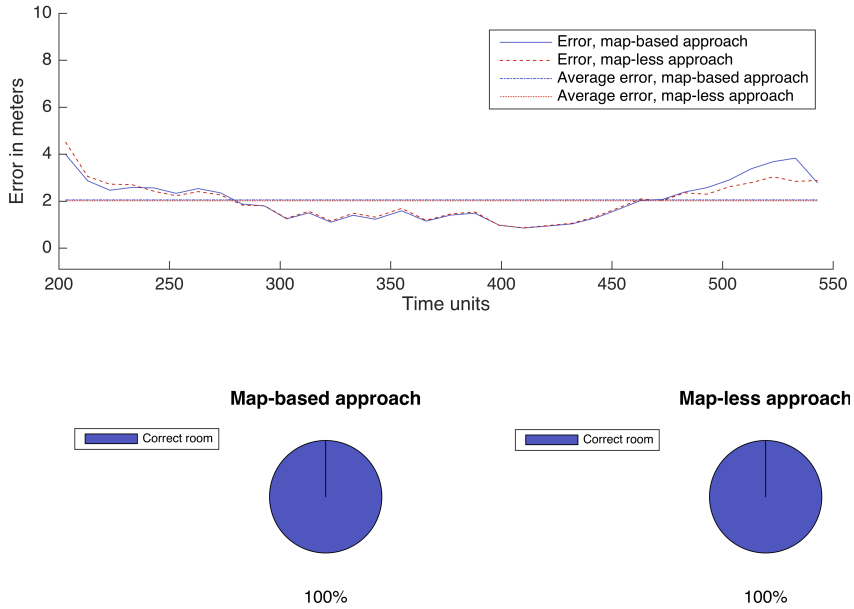


Figure A.7: Average error and correct room estimation plots

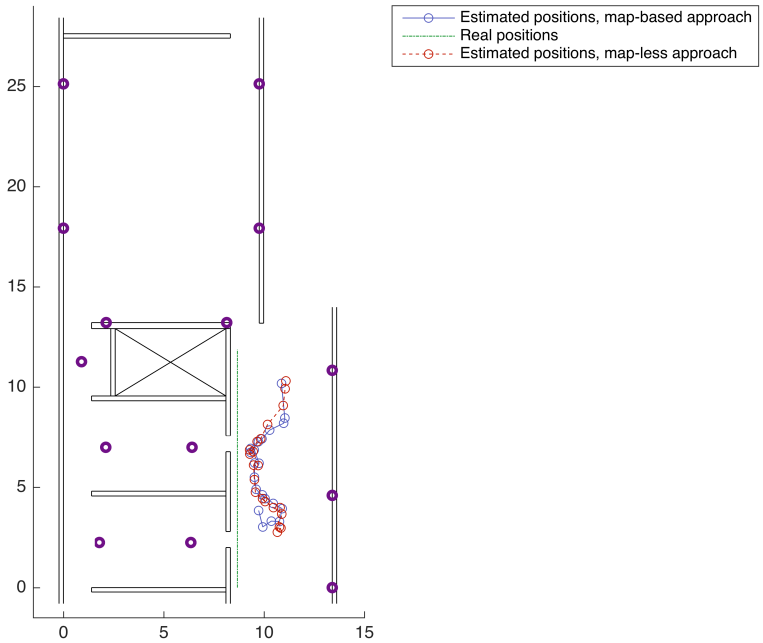


Figure A.8: Real and estimated paths