HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

Indoor Self-Localization with Smartphones Using iBeacons

Thomas Bopst

Konstanz, 28.04.2015

MASTERARBEIT

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

an der

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Fakultät Informatik
Studiengang Master of Science Informatik

Thema: **Indoor Self-Localization with Smartphones Using iBeacons**

Masterkandidat: Thomas Bopst, Albert-Gebhardt-Str. 5, 79761 Waldshut-Tiengen

1. Prüfer: Prof. Dr. Oliver Bittel
2. Prüfer: M. Sc. Michael Blaich

Ausgabedatum: 01.11.2014
Abgabedatum: 28.04.2015

Zusammenfassung (Abstract)

Thema:	Indoor Self-Localization with Smartphones Using iBeacons
Masterkandidat:	Thomas Bopst
Hochschule:	HTWG-Konstanz
Betreuer:	Prof. Dr. Oliver Bittel M. Sc. Michael Blaich
Abgabedatum:	28.04.2015
Schlagworte:	Indoor Localization, Indoor Positioning, iBeacon, Smartphone, Particle Filter, Monte Carlo Localization

Location-based applications are nowadays taken for granted in our life, but until now the available indoor localization systems have not been able to establish themselves. At their WWDC 2013 Apple introduced a new technology called iBeacons. This relies on small and cheap Bluetooth Low Energy (BLE) chips which enables new location awareness features (Bruins, 2013).

In this thesis a new indoor self-localization solution based on Monte Carlo Localization is presented. A Particle Filter is used to combine RSS-based distance estimations to deployed beacons, with the user's motion, as determined by the smartphone's built-in sensors. In addition map information, such as free and occupied space, is used.

The solution's complexity is very low; besides, the smartphones and beacons no additional hardware or infrastructure is required. This reduces the initial and maintenance costs. The whole localization takes place on the users' smartphones. Compared to other approaches, the location estimation is very robust. During the evaluation, a user's stationary location with an accuracy of 2.29 m was achieved. Furthermore, a user's path in the test environment could be tracked. In addition, the solution's algorithm detects wrong position estimations to recover.

Ehrenwörtliche Erklärung

Hiermit erkläre ich *Thomas Bopst, geboren am 31.01.1990 in München*, dass ich

(1) meine Masterarbeit mit dem Titel

Indoor Self-Localization with Smartphones Using iBeacons

an der HTWG-Konstanz unter Anleitung von Prof. Dr. Oliver Bittel selbständig und ohne fremde Hilfe angefertigt und keine anderen als die angeführten Hilfen benutzt habe;

(2) die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 28.04.2015

(Unterschrift)

Abbreviations

AOA Angle of Arrival

API Application Programming Interface

BLE Bluetooth Low Energy

BR Basic Rate

BT Bluetooth

CDF Comulative Distribution Function

CL CoreLocation

CM CoreMotion

CPU Central Processing Unit

EDR Enhanced Data Rate

EKF Extended Kalman Filter

GPS Global Positioning System

KF Kalman Filter

LE Low Energy

MAC Media Access Control

MCL Monte Carlo Localization

MEMS Microelectromechanical Systems

OS Operating System

PDF Probability Density Function

PF Particle Filter

UUID Universal Unique Identifier

LOS Line-of-sight

RSS Received Signal Strength

RSSI Received Signal Strength Indicator

RTOF Roundtrip Time of Flight

RX Receiver

TDOA Time Difference of Arrival

TOA Time of Arrival

TX Transmitter

WWDC Worldwide Developers Conference

Contents

Chapter 1

Introduction

Would it not be great if at the airport we could be guided from the arrival to the departure gate? Would it not be fantastic, if at a huge conference or trade fair your smartphone guides you from one interesting stand to the next one? Would it not be awesome, to compose a shopping list in your favorite grocery store's app, and then being guided on the shortest and fastest path through the store?

Location-based applications are by many people taken for granted and seen as natural aid in their life. About two decades ago, the probably most popular location-based applications, Global Positioning System (GPS) based navigation systems, drastically improved our possibilities, by providing an alternative to the old paper map. Today, GPS based applications are an essential feature of smartphones. We use them on a daily basis to determine our position in new cities and on trails, or to track our sports activities. Importantly, all of them only work outdoors, because GPS requires Line-of-sight (LOS), from the smartphone to the satellites for accurate location estimation.

In the area of mobile indoor robotics, indoor self-localization is a well known problem. Robots are equipped with sensors, such as odometers, laser scanners, or ultrasonic sensors, to determine their position. Humans are typically not equipped with sensors, but smartphones, our permanent companions, contain lots of them, to perceive our environment.

Actually, approaches for indoor self-localization with smartphones, based on WiFi or Bluetooth, do exist, but until now, they did not manage to get established in our, respectively the everyday life. From my point of view it is a classical chicken-and-egg problem. Most people do not know that technically indoor localization is possible, and thus the owners of buildings do not see a reason to provide the necessary infrastructure. An additional confounding factor is that the needed infrastructure is relatively expensive. Consequently, once the infrastructure has become cheaper and the technical feasibility is more widely known, applications of this technology will emerge.

Apple contributed to the solution of both problems by introducing a new technology called *iBeacon*[1]. This was presented at their yearly Worldwide Developers Conference (WWDC) in June 2013 (Bruins, 2013). iBeacons are small hardware chips equipped with a BLE module. Adequately equipped smartphones can pick up the BLE signals and estimate the proximity to a beacon. Apple Inc. (2014b) promotes the technology on their iBeacon developer website with the following use-cases: "From welcoming people as they arrive at a sporting event to providing information about a nearby museum exhibit, iBeacon opens a new world of possibilities for location awareness, and countless opportunities for interactivity between iOS devices and iBeacon hardware." Finally, the press also added the above mentioned use-case *indoor navigation*, such as navigation from arrival hall to the departure gate at an airport, or the navigation from one interesting booth to the next at a conference or trade fair.

[1] *iBeacon* is a registered trade mark of Apple Inc.

1.1 Requirements

From my point of view an indoor self-localization system should fulfill the following requirements. The system should have the possibility to be used with any of the most common devices. Therefore, it needs to be cross-platform, i.e. across operating systems. Additionally, the system should be of low cost for both, the relevant building or business owner as well as the user. Consequently, the system's infrastructure should be as simple as possible. The system should also be easy-to-deploy and cause little maintenance costs. Besides that, the users' device and the application's algorithm should be as independent as possible from the remaining infrastructure. Thus, localization should take place on the user's device.

1.2 Idea

The idea is to design, implement and evaluate an indoor self-localization system with smartphones using iBeacons, and which fulfills the above mentioned requirements.

To that purpose, commonly available smartphones equipped with a BLE module are used. The solution is based on Monte Carlo Localization (MCL). The smartphones' built-in sensors are used to measure the user's motion. The algorithm requires also a map of its environment. Additionally, the proximity to known beacons is integrated into the position estimation.

The designed solution can run on every platform. iBeacons are relatively cheap and require no additional infrastructure, such as a power supply or network connection. They can be powered with a small battery or a coin cell over months. The deployment is very simple, as well. The beacons just need to be distributed over the area, where the localization should take place. Additionally, an initial calibration step needs to be performed. If the beacons' environment does not change, no maintenance is necessary. The user's device is completely independent. The whole position estimation takes place on the user's device. No connection between smartphone and the remaining infrastructure is being established.

1.3 Structure

The document is divided into seven chapters. First, an overview about localization in general is given. Chapter 2 covers localization algorithms and related work. Then, Chapter 3 gives a detailed insight into iBeacon technology, including Bluetooth Low Energy, configuration parameters, the Application Programming Interface (API) provided by Apples iOS 8.1 operating system, and the signal's quality. Afterwards, a close look on the smartphones' built-in sensors, the provided APIs and the provided data is taken in Chapter 4, including an evaluation at the chapter's end. After providing the basis, the solution's implementation is presented in Chapter 5. This proof of concept is evaluated in Chapter 6. Finally, research is concluded in Chapter 7, including ideas, how the project could be developed further.

Chapter 2

Fundamentals

In this chapter, first the basics of localization are introduced. Afterwards, the Particle Filter (PF) and Monte Carlo Localization (MCL) algorithm are explained. The detailed reasons for choosing MCL for the solution are pointed out in Section 5.1. At the chapter's end, an overview of further localization approaches used in related work, is given.

2.1 Fundamental Aspects of Localization

Localization, i.e. positioning, is the process of estimating the position of an object in its environment. Usually the object's environment is defined by using a coordinate system. One of the most popular coordinate systems is the *World Geodetic System 1984 (WGS84)*, which describes a position on earth, by longitude and latitude. Sometimes, it is sufficient to use a local coordinate system; for instance, a two or three dimensional cartesian coordinate system. Indoor localization is one example, where it is sufficient to use a local coordinate system, because the object's position is limited to a defined area, e.g. within a building.

2.1.1 Location Types

There are four types of locations. A *physical location* is described by coordinates on a map. *Symbolic locations* such as "in the living room" or "on the small table in the kitchen", are often used by humans. If all objects share a frame of reference, like chess pieces on a chess board, the type of location is called an *absolute location*. When an object has its own reference frame, as for instance when saying "I am two meters away from the door", which often includes a proximity value, it is called a *relative location* (Liu et al., 2007).

Typically indoor localization is based on absolute locations, because all objects, such as landmarks, obstacles (i.e. occupied space), and corridors (i.e. free space) share the same frame of reference, i.e. the building's map. It is also possible to use physical locations for indoor positioning by either transforming the objects' absolute positions into physical positions or by knowing them. In the following a location is usually an absolute location.

2.1.2 System Topology

Usually, a pure wireless localization system consists of at least one transmitter that emits some sort of signal, and one or more receivers, i.e. measuring units. The system's topology can be *remote positioning*, which means that the transmitter's position is estimated by using the measurements from multiple measuring units, that are deployed at fixed locations. The position estimation takes place in a component separate from the transmitter. This component uses the measurements as input. If the measurement unit is mobile and capable to estimate its own position by collecting the signals from fixed transmitters, the system's topology is called *self-positioning*. *Indirect remote positioning* is basically the same as self-positioning, but the collected measurements are sent via some data connection to an external service, which performs the position estimation. If the system's topology is basically remote

positioning, but the measurements are transferred via a wireless data link to the mobile side, the topology is called *indirect self-positioning* (Liu et al., 2007).

Besides a pure wireless localization systems, which just uses wireless signals for the positioning, other systems, often used in the area of mobile robotics, do exist. Robots are often equipped with sensors, such as laser scanners or ultrasonic sensors, to measure distances to obstacles. Wheel- and chain-based robots are usually also equipped with odometers, others have accelerometers and rotation rate sensors to estimate the traveled distance. The used localization algorithms are able to combine different sensors to provide a more accurate estimation (Thrun et al., 2005). Mobile robots, especially if they operate autonomously, are usually self-positioning, or indirect remote positioning systems, which often depends on their system's hardware capabilities. Remote positioning as well as indirect self-positioning systems are rarely the case in the field of mobile robots.

The presented solution is a self-positioning system. It measures the distances to known beacons, but it not only relies on the measured wireless signals, it uses also the user's motion like a robot. Thus, the user's smartphone is the mobile measuring unit, which additionally does the whole position estimation.

2.1.3 Uncertainty

In order to derive an accurate position, localization systems need to be able to accommodate uncertainty. According to Thrun et al. (2005), uncertainty in robotic systems has several reasons.

One of these reasons is the *environment*. If a robot works together with humans, the environment is very dynamic and unpredictable, because the robot does not know for; instance, where and when a human moves to. Of course, robots are equipped with *sensors* to recognize obstacles like humans, but sensors may also cause uncertainty. On one hand, they have limitations, e.g. a camera has a specific resolution, and on the other hand, noise may unpredictably influences their measurements. Typically, robots have *actuators*; for instance, motors to move them within their environment. Actuators have precision limitations. Additionally, their successive physical components; for instance, a robot's wheels on different surfaces, may cause unpredictable effects. To allow robots to operate in a physical environment, models of it are used. *Models* are abstractions, which approximate a certain object or behavior. Thus, information is lost, and additionally every model contains some inaccuracy. Robotic systems work in real-time environments. Due to computational reasons, their *algorithms* cannot really operate in real-time. In reality, a physical property changes continuously, but a robot's measurements can only be processed in certain time steps, which causes inaccuracy.

All mentioned factors together might cause large uncertainty. Thus, the system needs to be able to accommodate or somehow compensate for it. This is the reason, why the algorithm, introduced in Chapter 5, not only relies on the distance measurements to the beacons. Due to the uncertaintys' high importance, the used sensors' uncertainty is discussed at the end of Chapter 3 and 4.

2.1.4 Localization Problems

Localization is not always the same. There are different localization problems depending on the environment and the use-case of the application. According to Thrun et al. (2005) the following problems exist:

- **Local vs. Global Localization:** If the initial position is known when the localization starts, a so-called position tracking only is necessary, to know future positions. This is called *local localization*. In the case where the initial position is unknown, and this position can by anywhere on the map, this is called *global localization*. These types of algorithms are more difficult than position tracking. Global localization solves also the *kidnapping* problem, which means that e.g. a robot is being placed on a different position during the position estimation. Of course, that is not very typically, but by solving this problem the algorithm is also able to recover if it is stuck in a state where it never will be able to approximate the true position.

- **Static vs. Dynamic Environments:** In a *static environment* the localizing object, e.g. a robot, is the only object that changes its position over time. All obstacles and other objects maintain their fixed position. In a *dynamic environment* other objects besides the localizing object can change their position. Typical objects in a dynamic environment are humans, furnitures, like chairs or doors that sometimes are moved, closed or opened.

- **Passive vs. Active Approaches:** If the localization algorithm is able to control the localizing object's motion, this is called an *active approach*. Through the ability to influence a motion, the algorithm can induce movement of the object, e.g. the robot, if the localization is stuck and needs more information; for example, from another point within a room, to decide what its position is. *Passive* algorithms just observe the motion, but cannot influence it. Usually, active approaches are built upon passive ones.

- **Single-Robot vs. Multi-Robot localization:** *Single-Robot* localization means, that the robot only needs to localize itself. *Multi-Robot* localization requires of course multiple robots in the same area. It builds upon single-robot localization; thus, each robot can estimate its own position. New opportunities arise if the robots can recognize, each other and are able to communicate, to exchange their estimated positions.

The solution presented in this thesis, solves the global localization and kidnapping problem in a dynamic environment; whereas, the in Chapter 6 presented results where measured in a static environment. The algorithms implementation is a passive approach. Furthermore, the solution only solves the Single-"Robot" localization problem, but Chapter 7 presents an idea how Multi-"Robot" localization could be used to improve the presented solution.

2.1.5 Performance Metrics

To be able to compare different localization systems and algorithms the following criteria, recommended by Liu et al. (2007), is introduced. Often the system's accuracy is the only metric being used. But also from my opinion as well, the system's accuracy is not sufficient for evaluation purposes. In addition to the metrics recommended by Liu et al. (2007), the system's usability is in my opinion another important criteria.

- **Accuracy** is the error of the estimated position. It is the Euclidean distance between the estimated and the real position.

- **Precision** takes the consistency into account. Liu et al. (2007) defines it as the result of "the cumulative probability function[s] (CDF) of the distance error", for instance 90 % location precision within 2 m, i.e. $\mathrm{CDF}(2\,\mathrm{m}) = 0.9$.

- **Complexity** depends on several factors, such as the required hardware, the software complexity, the algorithms complexity, the necessary infrastructure, etc. However, Liu et al. (2007) focus on software, i.e. computing complexity only.
- **Robustness** expresses the ability of a system to deal with wrong sensor data, unexpected values, or even no sensor data within a certain time interval.
- **Scalability** describes the possibility of a system to be extended or reduced in space, or a change of the density of the transmitters or measurement units.
- **Cost** depends on several factors. The financial costs are a very important factor. This depends on the hardware, implementation and installation costs. The energy consumption, which in the end also reflects in financial cost, is also an important factor. The factor time is also not negligible. The amount of time it takes to deploy and maintain such a system can be enormous.
- **Usability** depends mainly on the system's user friendliness, i.e. is it difficult for the user to use the system, is the system self-explaining, etc. If a system's usability is very bad its usually not being accepted by the users.

In Chapter 6, the presented solution is evaluated against these criterias, with focus on the system's accuracy and precision.

2.2 Particle Filter Algorithm and Related Work

The solution presented in this thesis builds upon Monte Carlo Localization (MCL), which is based on Particle Filter (PF). This section introduces first the PF's basics and afterwards the fundamentals of MCL. Finally, it gives an overview of related work, based on PF and MCL.

2.2.1 Introduction to Particle Filter

The PF is a non-parametric filter, i.e. it does not have fixed parameters which describe its distribution's functional form. By contrast Kalman Filter (KF) is a parametric filter, which parameters describe its functional form, i.e. a Gaussian distribution. PF approximates the functional form by a finite number of values, often referred to as *particles* or *samples*. The approximation's quality depends on the particle set's size. Figure 2.1 illustrates the approximation of a posterior distribution by samples. It also shows, that PF is well-suited for multi-modal distributions (Thrun et al., 2005).

According to Thrun et al. (2005), the idea is the posterior's $bel(x_t)$ representation by a random set of samples, drawn from the posterior. Each sample x at time t is a hypothetical state in state space, i.e. in real world. The particle set $\chi_t = \left\{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}\right\}$ is the distribution's approximation, containing M particles. The PF's algorithm, depicted in Listing 2.1, transforms the posterior $bel(x_{t-1})$, represented by the particle set χ_{t-1}, by integrating the latest control u_t and measurement z_t into the current posterior $bel(x_t)$. Thus, the algorithm creates first an empty, temporary particle set $\bar{\chi}_t$. Then, the states from χ_{t-1} are updated by integrating the current control u_t. Formally, this is done by sampling the new state $x_t^{[m]}$

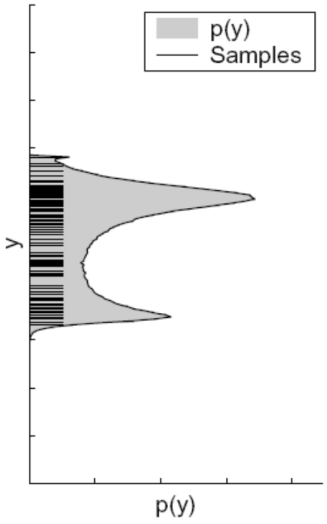


Figure 2.1: Approximation of a random variable Y, i.e. a posterior distribution by samples (Thrun et al., 2005, p.97).

```
1   ParticleFilter(χ_{t-1}, u_t, z_t) {
2
3     χ̄_t = ∅
4     for m = 1 to M {
5       sample x_t^[m] ~ p(x_t|u_t, x_{t-1}^[m])
6       w_t^[m] = p(z_t|x_t^[m])
7       add ⟨x_t^[m], w_t^[m]⟩ to χ̄_t
8     }
9
10
11    \\ resampling
12    χ_t = ∅
13    while size of χ_t less M {
14      draw i with probability ∝ w_t^[m]
15      add x_t^[i] to χ_t
16    }
17
18    return χ_t
19  }
```

Listing 2.1: The Particle Filter algorithm (Thrun et al., 2005).

from the *state transition probability* $p(x_t|u_t, x_{t-1}^{[m]})$ (Listing 2.1, Line 5). Afterwards, an *importance factor* $w_t^{[m]}$ for the new sample $x_t^{[m]}$ is calculated (Listing 2.1, Line 6). It is defined as $p(z_t|x_t^{[m]})$, which is the probability for the measurement z_t given the new state hypothesis, representing a value in real world. The new state hypothesis and its importance factor are stored together as tuple, in the temporary set $\bar{\chi}_t$ (Listing 2.1, Line 7). The weighted set approximates the posterior $bel(x_t)$.

Then, the resampling, i.e. importance sampling, transforms the weighted particle set, based on the particls' weights into a new set, which represents the new distribution. To do so, M particles are drawn according their weight and inserted into the new set χ_t (Listing 2.1, Line 14, 15). During this step some particles are duplicated, usually the ones with high weight, and others, the ones with low weight, are lost. Thrun et al. (2005) compares it with the Darwinian idea of *survival of the fittest*. According to them, "it refocuses the particle set to regions in state space with high posterior probability".

Instead of the drawing, it is also possible to have a weighted set χ_t. In every recursion, the weights of the existing particles are multiplied with their new weight. The method's disadvantage is, that a larger particle set is required to reach the same approximation quality, because the particles keep their position in state space for ever, i.e. the particle cloud does not move. For this method the particle weight $w_t^{[m]}$ of each particle needs to be initialized with 1.

PF can be adaptive; for instance, based on the available processing power. The particle set is then being increased or decreased to adapt to the available processing power (Thrun et al., 2005).

2.2.2 Monte Carlo Localization

MCL is a popular localization algorithm often used in the indoor mobile robotics field. The algorithm can deal with a broad range of the before discussed localization problems, especially with the local and global localization problem.

The basic algorithm, shown in Listing 2.2, is very similar to the PF, presented in Listing 2.1. The sampling from the *state transition probability* is substituted by a method called `sample_motion_model`, which samples the new hypothesis by integrating the control u_t with respect to the motion and its uncertainties during the execution (Listing 2.2, Line 5). Furthermore, the *importance factor* calculation is substituted by the `measurement_model` function, which depends on the used sensor (Listing 2.2, Line 6). It uses the sensor measurements and their uncertainty to calculate the particles' importance factors. It additionally takes the environment's map into account. Usually, the map is used to calculate the importance factor based on the measurement z_t; for instance, the distance to a landmark, where the landmark's position is stored in the map. But a map can also be used to detect particles at impossible positions, and thus to reduce their importance factor. A very detailed description of MCL and variations of the algorithm is provided by Thrun et al. (2005).

A visual example of, how a global localization using MCL works, is shown in Figure 2.2. The three images show the particles, i.e. the posterior $bel(x_k)$, over time. At the beginning uniform random particles are distributed over the whole state space, because the robot actually does not know where it is, and thus could be positioned anywhere. After moving and gathering sensor data, the particles concentrate in certain regions with higher probability for the true position. Thus, the posterior changes. In the end, the particles are concentrated on

```
1  mcl(χ_{t-1}, u_t, z_t, map) {
2
3    χ̄_t = ∅
4    for m = 1 to M {
5      x_t^[m] = sample_motion_model(u_t, x_{t-1}^[m])
6      w_t^[m] = measurement_model(z_t, x_t^[m], map)
7      add ⟨x_t^[m], w_t^[m]⟩ to χ̄_t
8    }
9
10
11   \\ resampling
12   χ_t = ∅
13   while size of χ_t less M {
14     draw i with probability ∝ w_t^[m]
15     add x_t^[i] to χ_t
16   }
17
18   return χ_t
19 }
```

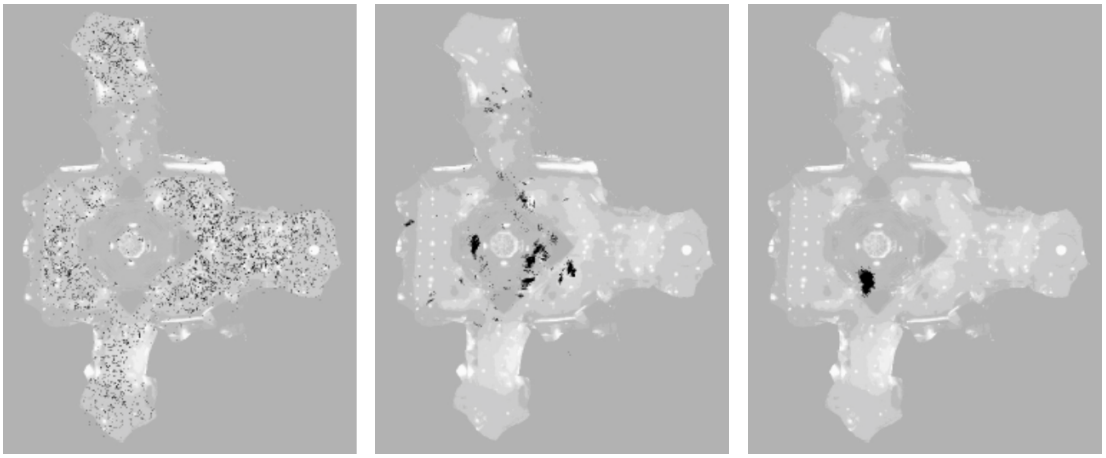
Listing 2.2: The basic Monte Carlo Localization algorithm (Thrun et al., 2005).

one spot; hence, the confidence of being at this position is very high (Thrun et al., 2005).

2.2.3 Related Work

Siddiqui and Khalid (2012) proposes a localization system that uses PF and RSS-based trilateration of WiFi signals (explained in Section 2.3.1), to track a mobile unit. The approach relies solely on sensing Received Signal Strength (RSS) values, no other sensors are used. First, they collect the RSS values in their environment. They are stored together with their location in a lookup table. For their MATLAB simulation they generate a random path with 100 points using the before collected values. The path is tracked during the simulation, just based on the RSS values. Their simulation simulates a notebook equipped with WiFi, which is tracked. Thus, their PF's implementation does not sample a new state by integrating the control u_t as mentioned before (Listing 2.1, Line 5). They use two different weighting functions during their experiment, firstly, a *Gaussian weighting function*, based on the Euclidian distance, and secondly, a *Triangular weighting function* (Listing 2.1, Line 6). According to Siddiqui and Khalid (2012), their solution achieves an accurate localization of $\approx 3\,\text{m}$ as a result of their simulation. During the simulation they assume a static environment with sparse activity and a smooth and continuous motion. They also assume that no large variations in the signals' strength occur over short distances. In addition, they remove the first and last 10 % of the recorded measurements, i.e. when the person placed the measuring unit and picked it up again, to have independent values of a human in proximity. Additionally, they neglect outliers.

Straub (2010) implementes an indoor pedestrian localization and tracking system based



(a) (b) (c)

Figure 2.2: A visual example of the Monte Carlo Localization algorithm. It shows a building's map and the particle distribution over time (Thrun et al., 2005).

on PF, which uses a system called *PiNav*, developed at the institute where Straub wrote his thesis. It extracts the person's motion in form of step frequency, step size, and heading. The person needs to be equipped with the hip-mounted PiNav-System, which includes an accelerometer, gyroscope and a compass. Additionally, Straub (2010) uses an accessibility map, which is based on a Neural Network, to learn which space on the map is accessible by pedestrians. By fusing the pedestrian's motion and the accessibility map in a PF, his solution achieves an average accuracy of 1.1 m (Straub, 2010).

Wang et al. (2007) propose a RSS-based WiFi positioning system using PF to fuse the location estimated via WiFi with additional map and accelerometer data. Their system uses the in Section 2.3.4 mentioned Scene Analysis approach, especially the k-Nearest Neighbors (kNN) algorithm for the positioning. During the offline stage they collect fingerprints of WiFi signals in one meter distance to the access points. During the localization, i.e. the online stage, the algorithm compares the fingerprints of the collected WiFi signals with the stored ones from the offline stage and returns a location estimation. According to Wang et al. (2007), the 3-axis accelerometer's values can be used in theory, to estimate the traveled distance by integrating the values. But, due to the low signal and the sensors' noise it does not work in praxis. Thus, they use a zero-crossing approach to detect steps and estimate their size. The approach detects a step if the vertical acceleration crosses the zero-line. Additionally, their solution uses the environment's map to sort out impossible particles, like particles crossing a wall. Their simulation's results show, that by using map and accelerometer in addition to kNN, their location error improves by 40 %. Their solution is also 30 % better than the KF's result. During their real world tests, they achieved a mean error of 4.3 m with a standard deviation of 2.8 m (Wang et al., 2007).

Siddiqi et al. (2003) uses MCL to globally localize a *Pioneer 2DX* robot in an indoor environment using the robots odometers and the measured RSS of WiFi signals. The algorithm's action model, i.e. motion model, is based on the data measured by the odometers. They deliver data for short distances with a very good accuracy. The motion is divided into translations and rotations. Their action model samples the motion with different uncertain-

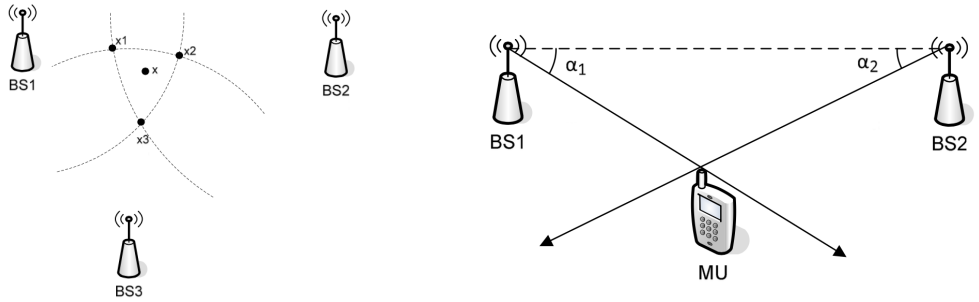
ties for translation and rotation. The observation model, i.e. measurement model, uses a signal strength map, which is divided into a grid with squared fields with a size of 0.5m. Before localization can take place, the RSS of each WiFi access point for each grid cell needs to be measured to store RSS mean and standard deviation for each cell. Due to the fact, that Siddiqi et al. (2003) assume that the signal attenuation does not change on short distances, they just measure it for a few cells and assume the same value for the neighbor cells. During the localization, the actual RSS values are compared with the collected values for the importance factor. Additionally, their observation model sorts out particles, by using the map constraints, but without taking the robot's orientation into account. According to Siddiqi et al. (2003), just using the map constraints without the RSS comparison, already enables very good localization results. During their tests they found out, that by increasing the amount of access points, the location error decreases (Siddiqi et al., 2003). They mention, that their approach should also work for localization of humans instead of robots. But they also note, that a robot's motion model is simpler than the one of a human. According to Siddiqi et al. (2003), they achieve during most test cases an accurate localization with an error of $\approx 2\,\mathrm{m}$.

2.3 Further Localization Algorithms and Related Work

This section focuses on further localization algorithms, which are suitable for indoor self-localization using smartphones. Some algorithms are discussed in more detail than others, depending on their closeness to the solution presented in this thesis. After introducing the algorithm itself, an overview of related work, based on that algorithm, is given.

2.3.1 Triangulation

Triangulation is a well-studied localization method which relies on triangle's geometric properties (Liu et al., 2007; Wang et al., 2013). Triangulation is the hypernym of the two characteristics *lateration* and *angulation*.



(a) Lateration (b) Angulation

Figure 2.3: Principle of the position estimation of a mobile measuring unit using triangulation methods. The wireless signals are transmitted by fixed base stations (Wang et al., 2013).

Lateration

Lateration estimates the location based on the distances from the measuring unit to the transmitters. To precisely estimate a position, three base stations are required. The measuring unit measures the distance to each base station and puts a circle with the radius of the measured distance around the base station. The circles's common intersection point, is the exact position of the mobile unit. In 2D space at least three signals must be received, in 3D, four signals are required. According to Wang et al. (2013), one common intersection point exists only in theory. In practice, errors, due to obstacles and imperfect propagation models, influence the distances. Thus, no common intersection point exists. Figure 2.3a depicts this case. Nevertheless, it is possible to estimate the mobile unit's position by using methods such as least-square, three-boarder, or the centroid-method (Wang et al., 2013; Liu et al., 2007).

The distances for lateration can be estimated by different methods. According to Liu et al. (2007), the distance between the measuring unit and the transmitter is directly proportional to the signal's propagation time. *Time of Arrival (TOA)* uses this fact to estimate the distance by measuring the signal's travel time. Consequently, the transmitted signal contains the timestamp of its transmission. Thus, the clocks of all transmitters and measuring units, that are part of the system, need to be precisely synchronized, in order to perform a correct estimation, which is not a trivial problem. For example a timing difference of 1.0 μs in a wireless localization system, such as Wifi or Bluetooth (BT), causes an error of $\approx$ 300m, according to Kotanen et al. (2003).

To overcome this disadvantage, a similar method called *Roundtrip Time of Flight (RTOF)* can be used. This method requires all components to be able to receive and transmit signals. The initial transmitter sends a signal, it is received and immediately returned, by the mobile unit. Thus, the transmitter sets the timestamp and compares it with its current timestamp, when the signal arrives. A relative time synchronization is sufficient. But RTOF has another problem. The signal is delayed by the processing time, which the unit requires to receive and to send it back to the original source.

The *Time Difference of Arrival (TDOA)* method calculates the distance from the different arrival times at the measuring units. Here, the measuring units need to be connected with each other to exchange the timestamps of the signals' arrival and to precisely synchronize their timestamps. The clocks' differences should not exceed tens of nanoseconds (Kotanen et al., 2003). The mobile unit, i.e. the transmitter, does not need to be synchronized.

All of the above mentioned methods are based on time, and thus need precisely synchronized or relatively synchronized clocks. According to Liu et al. (2007), TOA and TDOA require a LOS channel between transmitter and measuring unit in an indoor environment, because time and angle of the signals' arrival is affected by the multi-path effect, which is a problem of radio propagation. Wireless technologies, such as Bluetooth and WiFi do have a *Received Signal Strength Indicator (RSSI)* property, which expresses indirectly the signal's RX-power level, measured by the device (Kotanen et al., 2003). As a consequence, instead of measuring time to calculate the distance between sender and receiver, the loss of the signal strength on its path between the two units can be measured. In order to do that the measuring unit needs to know the initial RSS of the emitted signal. As mentioned by Liu et al. (2007), different theoretical and empirical models can be used to translate the signals attenuation to a distance estimation, but these models are not always reliable, due to multi-path fading and shadowing in indoor environment.

Angulation

Instead of using distances for the positioning, angulation uses the angles to multiple measurement units, as shown in Figure 2.3b. In 2D space two angles, in 3D space three angles are sufficient for the positioning. But, to measure the angle of an arriving signal, special directional antennas or arrays of antennas, are required. In the *Angle of Arrival (AOA)* method, the intersection point of the straights with the measured angles is the mobile units position. Here, highly precise angles need to be measured, which is a big problem in wireless environments due to multi-path reflections and shadowing, as already mentioned before (Liu et al., 2007; Wang et al., 2013).

Related Work

Wang et al. (2013) use Bluetooth to implement an indoor positioning system for mobile devices, like smartphones. The system is based on triangulation methods, i.e. on lateration, using RSS for the distance estimation. During their research, they evaluate the least-square, three-border, and centroid method for the position estimation. They write, without mentioning any details, that all three methods deliver satisfying results, if the mobile unit receives accurate RSS readings and a proper path loss model is used to calculate the distances. They also analyze the effect of placing a human body between transmitter and measuring unit and found out, that the signal is weakened by $6-8\,\mathrm{dB}$, which equals a distance of several meters, due to the fact, that decibel (dB) is a logarithmic unit. Due to the RSS values' fluctuation, they propose to filter the RSS readings by an average or weighted filter to improve the estimation accuracy.

Oksar (2014) also uses Bluetooth, based on RSS readings for indoor localization. In contrast to Wang et al. (2013), the mobile units are the transmitters, and the measuring units are the fixed base stations with known position. Thus, the system topology is remote positioning. They defined a *root-mean-square-error* function, which compares "the ratio of the square of distances to the base stations" with "the ratio of the signal levels" (Oksar, 2014). The used distances are calculated for assumed discrete positions of the transmitter. As shown by Oksar (2014), the better the assumed position matches the real position, the smaller is the result of the error function. Oksar (2014) assumes, that the received RSS decreases proportional to the squared distance. Thus, compared to Wang et al. (2013), Oksar (2014) does not directly calculate the distance between the transmitter and the measuring unit. In the mentioned experiment a root-mean-square-error of $2.309\,\mathrm{m}$ of localization accuracy was achieved in the best-case. But Oksar (2014) also mentions, that in a real life application the points are most probably non-discrete points; thus, some changes are necessary.

Hoflinger et al. (2014) develop an acoustic indoor localization system called ASSIST, which stands for *Acoustic Self-calibrating System for Indoor Smartphone Tracking*. According to Hoflinger et al. (2014), the system can track a smartphone with an error of $25\,\mathrm{cm}$. The smartphone sends out a high frequency acoustic signal using its speaker, which is not recognizable for humans. The signal's "amplitude A of sound decreases with distance r according to $\mathrm{A} \sim 1/\mathrm{r}$". To detect the signals, fixed measuring units with a sensitive Microelectromechanical Systems (MEMS) microphone are used. Despite the decreasing amplitude, they can detect the signals in $70\,\%$ up to $10\,\mathrm{m}$ distance. For the position estimation, the system uses the TDOA method. The measuring units are connected via a wired network, which is at the same time the units' power supply. The network is used to synchronize the units' timestamps and

to exchange the measured timestamps. According to Hoflinger et al. (2014), also the acoustic signals suffer from echoes and reflections, caused by multi-path propagation. To calculate a robust location, outliers are compensated by using PF and KF. They also mention, that the system can additionally use inertial smartphone sensors, such as accelerometer, gyroscope, and magnetometer besides the acoustic infrastructure (Hoflinger et al., 2014, 2012).

Further research projects based on triangulation methods are referenced by Liu et al. (2007).

2.3.2 Kalman Filter

The *Kalman Filter (KF)*, and the *Extended Kalman Filter (EKF)*, are techniques for filtering and prediction of linear and non-linear Gaussian systems with continuous space, such as an indoor robot localization system based on motion and measurements. As mentioned before, localization systems should be able to accommodate for uncertainty. Gaussian filters model the uncertainty of a state x, motion u, and measurement z as (multivariate) normal distributions. According to Kotanen et al. (2003), with knowing the uncertainties, KF optimally minimizes the estimation error's variance.

The algorithm is based on three probabilities. The *state transition probability* $p(x_t|u_t, x_{t-1})$ is the probability for a state x_t, i.e. a position, at time t after the transition from state x_{t-1} by applying the control i.e. motion u_t. The *measurement probability* $p(z_t|x_t)$ defines the probability for a certain measurement z_t in state x_t, e.g. a distance measurement to an obstacle at position x_t. The last probability is the *initial belief* $bel(x_0)$. It is the probability for the initial state at time $\mathrm{t} = 0$ (Thrun et al., 2005).

The algorithm is iteratively invoked. First it takes the current state and predicts the successive state by applying the motion. Next the measurement based on the predicted state is predicted. Afterwards, the predicted measurement is compared with the real measurement. Based on the comparison's result the algorithm corrects the before predicted state, which is the new state (Thrun et al., 2005).

A detailed description of KF and EKF is provided by Thrun et al. (2005).

Related Work

KF can be used for indoor localization by using Bluetooth technology as demonstrated by Kotanen et al. (2003). They use the signal's RSS to estimate the distance between the mobile unit and the transmitters. Then, they use EKF to merge the mobile unit's measurements with the current state, which represents the mobile unit's position in three-dimensional space. Their solution uses a constant state. Thus, no prediction of the successive state by integrating a motion takes place.

As mentioned by Kotanen et al. (2003), the "main source of errors is unreliability of RSSI". Their position estimation is based on filtered values instead of raw values. But in fact, it relies only on the wireless measurements and does not include additional sensor data or other information, as usual in the robotic's field. Their solution achieves an absolute positioning error of 3.76 m.

2.3.3 Proximity Method

The proximity method, which is also known as cell-id tracking, only provides a relative location information. For the localization, the mobile device listens for signals from different

transmitters. The transmitters' locations are known and can be identified by their identifier. As a result, the mobile device's location is *next to the transmitter's location* with the strongest signal (Liu et al., 2007; Wang et al., 2013).

As mentioned by Liu et al. (2007), the algorithm used by the proximity method is very simple to implement. Additionally, it can use different wireless technologies at the same time, such as Bluetooth, WiFi, etc. The device just needs to *listen* for all of them simultaneously.

To get a precise position information, many transmitters need to be distributed within the environment. If there are too little, and their range is up to several meters, the location information is very rough (Kotanen et al., 2003). As mentioned before, wireless signal's are heavily influenced by the environment. Consequently, it may also happen, that the device's location is not optimal, due to the fact that the signals are not attenuated in the same strength, and thus, the device location is being associated with a transmitter that is further away, than another one.

The solution presented in this thesis follows different approaches. Thus, the proximity method is not explained in more detail. Liu et al. (2007) gives a more detailed overview of the algorithm and related work.

2.3.4 Scene Analysis

Scene analysis localization algorithms, such as *k-Nearest-Neighbors (kNN)*, *Neural Networks*, and *Support Vector Machine (SVM)* are based on machine learning.

According to Liu et al. (2007), the use of such algorithms requires two stages, the offline and online stage. During the offline stage, fingerprints are collected at many positions in the area where localization should take place. Most often the fingerprints are RSS-based, which means, that at a specific position, the RSS values of all received transmitters are collected. Then the fingerprints are stored in a database together with their corresponding location. The online phase is the actual run-time phase, where the localization takes place. To do so, the application creates a fingerprint of the current received signals. Then the algorithm compares the fingerprint with the fingerprints stored in the database during the offline stage and returns the location of the best matching one.

As mentioned by Liu et al. (2007), the technique's main challenge is to deal with the signals that are affected by the environment. The scene analysis' offline stage requires a lot of effort, especially in large environments, which is its major disadvantage. Another disadvantage is, that if the environment changes the RSS fingerprints can change, too and thus the offline stage needs to be repeated.

The solution presented in this thesis follows different approaches. Thus, scene analysis is not explained in more detail. Liu et al. (2007) provides a detailed overview of the before mentioned algorithms and related work.

Chapter 3

iBeacon

At Apple's yearly WWDC 2013 Apple introduced a new technology, called *iBeacon*, to enhance smartphone applications' location awareness.

An iBeacon, in general a beacon, is a System-on-Chip which emits a *Bluetooth Low Energy (BLE)* signal in a predefined time interval, analogous to a lighthouse. Devices with compatible hardware and software, for example iPhone 4 S and Android version 4.3 and later versions, can receive the signals, to provide the user location based services. Typically, a beacon's size is less the size of a child's hand, and is powered by a small battery to be independent from its environment. Thus, it easily can be attached to another object to establish a region around the object. A smartphone app can then provide the user with a notification, once the user enters or leaves the region, by estimating the proximity to the beacon (Apple Inc., 2014a; BEACONinside GmbH, 2014). According to BEACONinside GmbH (2014), typical use-cases are:

- Location based marketing (e.g. digital coupons and digital signage of articles)
- Proximity sensing to an object
- Indoor localization and navigation (e.g. digital museum guide and airport guide)

After providing a brief idea what a beacon actually is, the next sections focuses on the communication technology which is used by a beacon to send its signal and the beacons itself. Afterwards, the API provided by Apple's mobile operating system iOS 8.1 to receive these signals is described. After providing the basics, the evaluation results from the proximity estimation to a beacon are presented. This results are important for the algorithm's implementation as described in Chapter 5.

3.1 Bluetooth Low Energy

BT is a wireless short range communication technology with the intent to replace, or reduce the amount of required cables. It is developed by the *Bluetooth Special Interest Group (Bluetooth SIG)* which is a large group of companies, such as chip manufacturers. The technology's key features are robustness, low cost and low power consumption. It is often used to connect a car's speaker phone with a smartphone, to connect a wireless keyboard or mouse with a computer, or to exchange files between two cell- or smartphones.

Originally, the Bluetooth specification contained just one wireless technology system called *Basic Rate (BR)/Enhanced Data Rate (EDR)*. Since version 4.0, released in 2010, the specification adds a second technology named *Low Energy (LE)*. The term LE is the technology's name as described within the specification. The technology's official (marketing) label is *Bluetooth Smart*. Devices labeled with it are only equipped with the LE technology. In contrast, devices labeled with *Bluetooth Smart Ready* are equipped with both technologies, the traditional BR and the LE technology (Bluetooth SIG, Inc., 2010). One of the differences is speed. BR/EDR has a max. transfer rate of 2-3 Mbps; whereas, LE is designed only for

1 Mbps. But, LE has the advantage of having a low current consumption. Its complexity is also lower than the one of BR/EDR (Bluetooth SIG, Inc., 2010).

Usually iBeacons are Bluetooth Smart devices (BEACONinside GmbH, 2014). To get a better idea of how the communication between a smartphone and an iBeacon works, the following sections provides an overview of the BLE technology.

Channels and Events

Both BT systems use the 2.4GHz frequency band. BLE divides the band into 40 channels. Three of them are advertisement channels, the others are data channels. These channels are used to transmit events, which consist of typical data packages. The BLE specification distinguishes between two event types, the *advertisement* and the *connection* event.

Advertisement events are used for unidirectional or broadcast communication between two or more devices. In addition *connectable advertisement* events to establish a pair-wise or bidirectional communication do exist. Advertisement events are transmitted via one of the advertisement channels. The device which sends the advertisement is called the *advertiser*. The device looking for these events is called *scanner*.

If a device looks for a *connectable advertisement* to establish a connection, it is called the *initiator*. It becomes the *master* after establishing the connection. The advertising device becomes the *slave*. Master and slave(s) form a piconet. Slaves in a piconet cannot communicate with each other. The connection establishment takes place on one of the advertisement channels. After establishing a connection, the communication takes place on the data channels by using frequency hopping instead of one specific channel (Bluetooth SIG, Inc., 2010).

Operations

According to Bluetooth SIG, Inc. (2010), BLE implements different operations for the above mentioned event behavior:

- To send advertisement events the *Advertisement Procedure* is used. It sends unidirectional broadcast messages without a connection to other devices. If the device itself is already connected to another device, it can only send non-connectable advertisements. Advertisements can contain user data. A scanner can respond to an advertisement to request more user data.
- The *Scanning Procedure* must be used to listen for advertisement packages. It can read the user data sent in an advertisement. Scanning is possible while being connected to a device.
- It is possible to listen just for a specific set of devices. In order to do that the *Device Filtering Procedure* needs to be used.
- The *Discovering Procedure* can be used to look for specific devices. It uses device filtering.
- To establish a connection the *Connecting Procedure* must be used.
- After the connection establishment the *Connected Procedure* is used to transfer data.

Security

As mentioned by Bluetooth SIG, Inc. (2010), the data is encrypted by the *AES-CCM* encryption method. For authentication it provides three different modes for different device capabilities:

- *Just Works*: Requires just a possibility to tell the device, that it should accept a connection request of an unknown device. A hardware button can be used, that the user needs to press for a few seconds for the initial connection establishment. The method is often used if the slave has no possibility to enter a key, e.g. to connect a smartphone with a bluetooth headset.
- *Passkey Entry*: Requires the input of a PIN code. Here, the master needs the possibility to show a PIN code and the slave needs some sort of keyboard. A typical example is the pairing process of a BT keyboard with a computer. The computer displays a code and the user has to enter it on the keyboard that wants to connect.
- *Out of Band*: Requires a separate channel which is used for the discovery and also for the exchange of the secure key; for instance, a wired connection.

Adverting and Response Data

Besides the mentioned user data, the advertisement and response always contain data such as the *local name* of the device, which can be used to identify it, information about the *manufacturer*, and the *TX-power level*, which was used to send the data packages. Furthermore, it includes *flags* which provide information whether the device is in limited discoverable mode, if the device supports also BR/EDR, if both modes can be used simultaneously, and further service-specific data.

Summary

Bluetooth Low Energy is a separate wireless communication technology which was added to the traditional Bluetooth system. Devices equipped with a BT chip of version 4, can either support both systems or just one. BLE is low cost and has low power consumption. Thus, it is possible to produce cheap devices like iBeacons which can be powered over months with a coin cell.

Often, the question comes up, whether beacons can be used to track users, i.e. their smartphones. To do so, a response from the smartphone must be sent. As shown, advertisements do not require a response; thus, tracking would only be possible by establishing, or at least trying to establish, a connection. But due to the fact, that just one connection at the same time can be established, and thus a device can only talk to one beacon, connection establishment cannot take place. Consequently, the claim made by Apple Inc. (2014a), that tracking or putting "user's private data at risk" is not possible, is true.

3.2 Beacon

As mentioned before beacons are small System-on-Chip devices, which send BLE signals. To manufacture beacons, compatible with Apple's iBeacon standard, or to get an insight in the official specification, a license from Apple needs be obtained. Thus, I am focusing on the

publicly available documentation, which describes the relevant parts with enough detail for the purpose of this thesis.

This section provides first an insight into the generally valid details. Afterwards, the possible configuration parameters of the beacons' hardware are outlined. Finally, the necessary steps for the beacon deployment are described.

3.2.1 General Details

In general, each beacon sends a BLE signal in a fixed time interval. On one hand, the signal contains information to uniquely identify a beacon, and on the other hand, data to be able to estimate the proximity, i.e. the distance, between the beacon and the device itself.

Identifier

The unique identifier consists of three parts, the `proximityUUID`, `major`, and `minor`, which together form a unique identifier. Usually, a smartphone's application is not interested in all beacons that are around, but rather in a specific subset. Thus, the application has to "tell" the Operating System (OS) in which it is interested in. The intention of the identifier's three parts is to build a hierarchical identifier system for beacons, to be able to search for a subset. The following example explains the three parts and their purposes:

Imagine a chain of stores' application which needs to support several shops. Each shop has multiple beacons deployed. Thus, the application is only interested in beacons belonging to the chain of stores. Furthermore, the application needs to know in which shop the user is located, or even in front of which beacon.

- **`proximityUUID`** (16 bytes): All beacons that belong to the same cain of stores share the same Universal Unique Identifier. Thus, the application can just search for this part of the identifier, i.e. for being notified once the OS receives any beacon signal belonging to the chain of stores.
- **`major`** (2 bytes): All beacons deployed in the same shop share the same `major` identifier. Consequently, the app is able to differentiate between shops. Furthermore, it can just search for beacons of a specific shop without having a lot of overhead.
- **`minor`** (2 bytes): Each beacon in a shop has a different `minor` identifier. Hence, by combining all three identifiers, the app can determine in front of which beacon in a certain shop the user is located.

Typically, these three parts can either be configured on the beacon itself, if not they need to be manufactured with the desired identifiers (Apple Inc., 2014a; BEACONinside GmbH, 2014).

Proximity estimation

As mentioned in Chapter 2, there are several physical methods to estimate the distance between a transmitter of a radio frequency signal and the receiver. In this case the Received Signal Strength (RSS) is used to estimate the proximity. Consequently, the receiver needs to know which TX-power level is used to send the radio frequency signal. As mentioned in

Figure 3.1: A 3D picture of sectioned beacons (BEACONinside GmbH, 2014).

Section 3.1, BLE, uses the 2.4 GHz frequency. According to Apple Inc. (2014a) and BEACONinside GmbH (2014), signals of this frequency band are heavily attenuated in obstructed environments and especially by water. Consequently, the knowledge of the sender's TX-power level is not sufficient to estimate the distance. Thus, a calibration value, which is the Received Signal Strength in a distance of 1m, is also sent to the receiver. Therefore, the later described calibration step, after deploying the beacons is necessary.

3.2.2 Hardware Specification

To test and evaluate the implementation I used ten beacons from a company called *BEACONinside GmbH* from Berlin, Germany. The beacons have the Model No.: B0001-A (HW: Rev 1.1, SW: 1.0). Figure 3.1 shows a 3D picture of sections of the beacons. According to BEACONinside GmbH (2014), the beacons' signal have a range from 0 up to 40 meters. The chipset they used was manufactured by Texas Instruments (Model CC2541). The company obtained a license from Apple to manufacture the beacons, which are compatible to Apple's iBeacon specification (BEACONinside GmbH, 2014).

The beacons's size is $5.8\,\text{cm} \times 7.96\,\text{cm} \times 2.25\,\text{cm}$. They can either be powered by two AAA batteries to provide a lifespan of approximately one year, or by an external power supply over micro-USB (BEACONinside GmbH, 2014).

The beacons are configurable with standard BLE tools, like compatible smartphone apps. Such an app establishes a BLE connection to the beacon. Each of the following configurable values can be accessed with a specific key. To configure the beacons an iOS app called *LightBlue — Bluetooth Low Energy*[1] from a company named *Punch Through Design LLC.*, recommended by BEACONinside is used. According to BEACONinside GmbH (2014), the following values are configurable:

- **`proximityUUID`:** For instance `F0018B9B-7509-4C31-A905-1A27D39C003C`
- **`major`:** A value between 1 – 65535
- **`minor`:** A value between 1 – 65535
- **`TX-power level`:** It defines the power level which is used to transmit the beacons advertisement packages, i.e. the signals. It can either be 0 dBm (default) which is the maximum power level, -6 dBm or -23 dBm which is the minimum power level.
- **`Advertising interval`:** Defines the time interval (100 ms – 10 s) in which advertisements are sent. The default time interval is 400 ms, configurable in steps of 625 μs.

[1] Link to the LightBlue App in the iOS AppStore (last access: 23.04.2015) `https://itunes.apple.com/de/app/lightblue-bluetooth-low-energy/id557428110`

- **Calibration value:** To estimate the distance of the receiving device to the beacon a calibration step is necessary. The measured RSS in a distance of 1m needs to be sent by the beacon with every advertisement. This value can be configured for each `Tx-power level`. The default is 65 dBm for the 0dBm power level. It needs to be encoded as the 2' complement in hexadecimal (e.g. 65 dBm = 0xBF).
- **Passkey:** To set one of this configuration parameters the (6 digit) passkey must be entered. The beacons' default value is `555555`.

Additionally, the following read-only properties about the device and its state are provided (BEACONinside GmbH, 2014):

- **Device Name:** The beacon's name consisting of `BEACON` and the latter three bytes of its MAC address.
- **Model Number**
- **Serial Number**
- **Hardware Revision Number**
- **Software Revision Number**
- **Manufacturer Name**
- **Battery Level:** The battery power level in percent.
- **Temperature:** The hardware's temperature in degrees Celsius.

To load the new values after changing the beacon's configuration, the beacon needs to be rebooted. The beacon can also be reset to factory defaults (BEACONinside GmbH, 2014).

3.2.3 Deployment and Calibration

As mentioned before, the BLE signal quality is heavily influenced by physical materials due to the fact that BLE uses the 2.4 GHz frequency. Especially, water which is a main part of the human body, attenuates the signal (Apple Inc., 2014a). Thus, the beacons need to be carefully deployed. Ideally, they should be deployed in a manner, that most often the line between beacon and receiver is unobstructed. There should also be enough space around the beacon, that its signal can spread out. By deploying them carefully, the signal's attenuation and other effects like multi-path fading and shadowing can be reduced (Apple Inc., 2014a; Liu et al., 2007).

The next step is the calibration step. To be able to estimate the distance between smartphone and beacon a reference value is needed. The reference value, as mentioned before, is the measured RSS in a distance of 1m to the beacon. The calibration value is configured on the beacon itself, which includes it in the sent advertisement together with its identifier.

According to Apple Inc. (2014a), the smartphone needs to be positioned in 1m distance to the beacon in portrait mode with the device's top half uncovered. Then the RSSI values were collected for at least 10 s. Meanwhile, the user should slowly move the device on an ≈ 30 cm line in front of the beacon, by maintaining distance and orientation. After collecting the values, the average value is calculated and configured on the beacon. It is advisable, to double check the device's estimated distance, after setting the calibration value. Due to the above mentioned problems, this needs to be repeated for every single beacon.

3.3 API — iOS 8.1

After describing what beacons are and how they work, this section focuses on the receivers side, especially on the API provided by Apple's iOS 8.1 platform. The CoreLocation (CL) framework's `CLLocationManager` component provides two functionalities, called *region monitoring* and *beacon ranging* to receive a beacon's signals within an iOS app (Apple Inc., 2014c).

Region monitoring

As the name already implies, the operating system monitors a specified region and notifies the application, when the user entered or left the region. This functionality is not restricted to regions established by beacons. It already existed long before, and monitors circular regions specified by a distance around a global position in longitude and latitude, e.g. a GPS position. iOS monitors the specified regions also if the application is in background mode or is currently not running. If the user enters or leaves a region, the application is launched by the OS; for instance, to display a notification to the user (Bruins, 2012, 2013; Apple Inc., 2014c).

Proximity to a Beacon

To continuously estimate the proximity to a beacon, *beacon ranging* must be used. To use this a `CLBeaconRegion` needs to be specified. The `CLBeaconRegion` object requires at least the beacon's `proximityUUID`. Furthermore, it is possible to specify a specific set of beacons, by adding the beacons' `major` identifier. Of course they have to share the same major identifier. By additionally providing a `minor` identifier, the framework searches only for one specific beacon (Bruins, 2013; Apple Inc., 2014c).

After starting the search, the `CLLocationManager` starts to notify the before specified `delegate` object in a continuous manner after it received the first advertisement from a beacon, which it is supposed to look for. This means that the `CLLocationManager` does not call the `delegate` until it receives the first matching beacon advertisement. After the first call it continuously calls the delegate approximately every second regardless of whether it actually received a beacon's advertisement or not.

The `CLLocationManager` passes an array of `CLBeacon` objects to its delegate. Each of it represents one beacon. Table 3.1 shows two example data sets, which the `CLLocation-Manager` passes to its delegate. Each data set contains four `CLBeacon` objects. The table's last entry represents a beacon, which is out of range.

A `CLBeacon` object contains on one hand the beacon's identifiers, as described in Section 3.2. On the other hand, it contains three values, which express the beacons proximity in different ways.

RSSI Property

The `rssi` property, stores the Received Signal Strength measured in dBm. It expresses how strong the received signal is. As mentioned before, a beacons' advertisement time interval can be specified. Remember, the used beacons send their advertisement per default every 400ms. The framework itself sticks to its $\approx 1\,\mathrm{s}$ time interval. It collects all received advertisements and calculates the average since the last reported `rssi` value. If a beacon's signal was received

time	proximityUUID	major	minor	proximity	accuracy	rssi
1.0400	F0018B9B-...-1A27D39C003C	21500	48945	2	1.2915	-67
1.0400	F0018B9B-...-1A27D39C003C	39821	56686	2	5.9948	-79
1.0400	F0018B9B-...-1A27D39C003C	41738	1201	3	6.8129	-80
1.0400	F0018B9B-...-1A27D39C003C	38722	6270	3	7.7426	-81
2.0387	F0018B9B-...-1A27D39C003C	21500	48945	2	1.4452	-68
2.0387	F0018B9B-...-1A27D39C003C	39821	56686	3	5.5235	-78
2.0387	F0018B9B-...-1A27D39C003C	41738	31201	3	6.8129	-80
2.0387	F0018B9B-...-1A27D39C003C	38722	6270	0	-1	0

Table 3.1: Two successive example datasets of `CLBeacon` objects passed by the `CLLocationManager` to the specified delegate. The column *time* shows a relative timestamp, to simplify the table.

before, but not during the current time interval, its `rssi` value is 0 (Bruins, 2013; Apple Inc., 2014c).

Proximity Property

The `proximity` property's value expresses the relative distance to a beacon. It can either be `unknown` = 0, `immediate` = 1, `near` = 2, or `far` = 3. The `unknown` state means, that the proximity could not be detected. One reason could be, that the *beacon ranging* just started but not enough advertisements were received. The `immediate` state "represents a high level of confidence that the device is physically very close to the beacon" (Apple Inc., 2014a). The `near` state is reported, if the user is in approximately $1-3\,\text{m}$ proximity to the beacon. Consequently, the `immediate` state is reported if the user is closer than $\approx 1\,\text{m}$ and the state `far` is reported if the "beacon device can be detected but the confidence in the accuracy is too low to determine either Near or Immediate" (Apple Inc., 2014a). Hence, the estimated distance is larger than $\approx 3\,\text{m}$.

Accuracy Property

The object's `accuracy` property actually expresses the proximity to a beacon in meters, but Apple Inc. (2014c) describes the property as follows:

> "The accuracy of the proximity value, measured in meters from the beacon. [...] Indicates the one sigma horizontal accuracy in meters. Use this property to differentiate between beacons with the same proximity value. Do not use it to identify a precise location for the beacon. Accuracy values may fluctuate due to RF interference."(Apple Inc., 2014c)

Thus, I first interpreted this description as the error of the `proximity` property's value, measured in meters. But, this makes of course not much sense, because the `proximity` value stores a relative value, which already implies that the user's proximity to the beacon is in a certain range.

$$d = 10^{-\frac{RSSI+A}{10*n}} \tag{3.1}$$

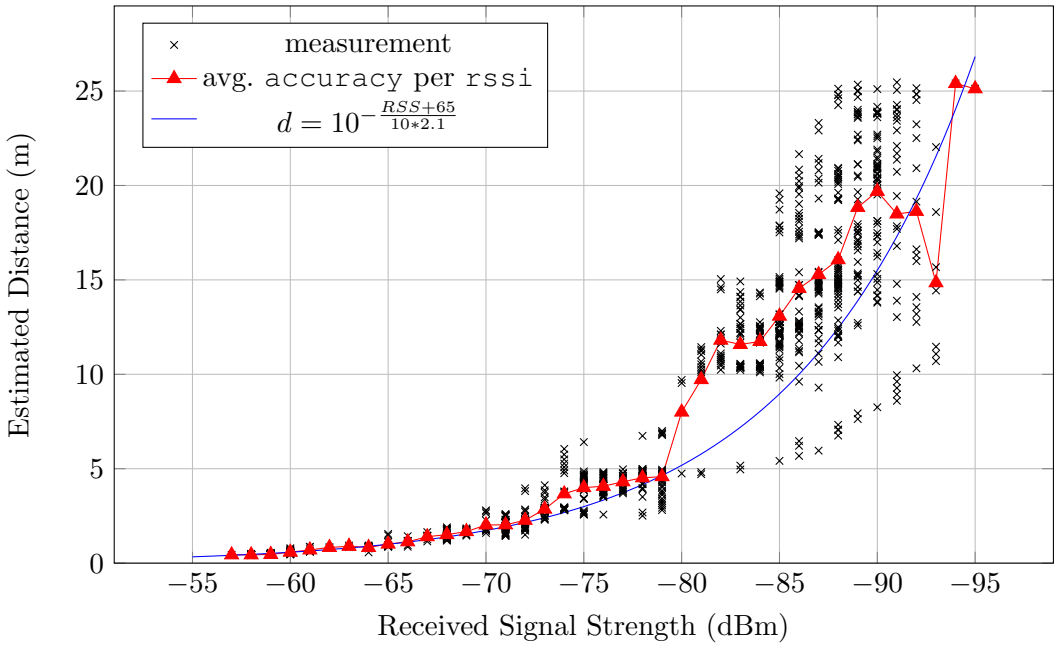


Figure 3.2: Dependency of the `CLBeacon` object's `rssi` and `accuracy` property. Proves that the `CLBeacon` object's `accuracy` is the estimated distance between smartphone and beacon because d is the estimated distance depending on the RSS according to Wang et al. (2013); Kotanen et al. (2003).

According to Wang et al. (2013), the distance d, between an iBeacon and a smartphone, which depends on the Received Signal Strength, can be calculated with the formula shown in Equation 3.1, where A is the RSS at a distance of 1 m, and n an environmental constant. To prove the assumption, that the `accuracy` value is actually the distance, test measurements for certain reference distances were performed. The measurements were taken outdoor with LOS between beacon and smartphone in order to have the signal's quality as less impacted by physical materials as possible. After collecting the `rssi` and `accuracy` of 60 measurements for different reference distances the data set was sorted according their `rssi` values. Then the `accuracy` value's average per `rssi` was calculated. Afterwards, the above mentioned formula shown in Equation 3.1 was used to calculate the distance for each measured `rssi` value. During the measurements, the beacon's calibration value was set to 65 dBm; thus, A = 65 dBm. To approximate the `CLBeacon` object's `accuracy` property best, an environmental constant of n = 2.1 was used, which is similar to the one used by Wang et al. (2013). Figure 3.2 illustrates the result and the fact, that the `CLBeacon` object's `accuracy` property is the estimated distance, between beacon and smartphone. Additionally, it implicitly depicts, that the unit decibel is a logarithmic unit.

3.4 Evaluation

After explaining how beacon advertisements can be received within an iOS app and which data can be accessed, this chapter evaluates the `CLBeacon` object's data especially the `accuracy` property's values, which are the ones used by the solution's algorithm.

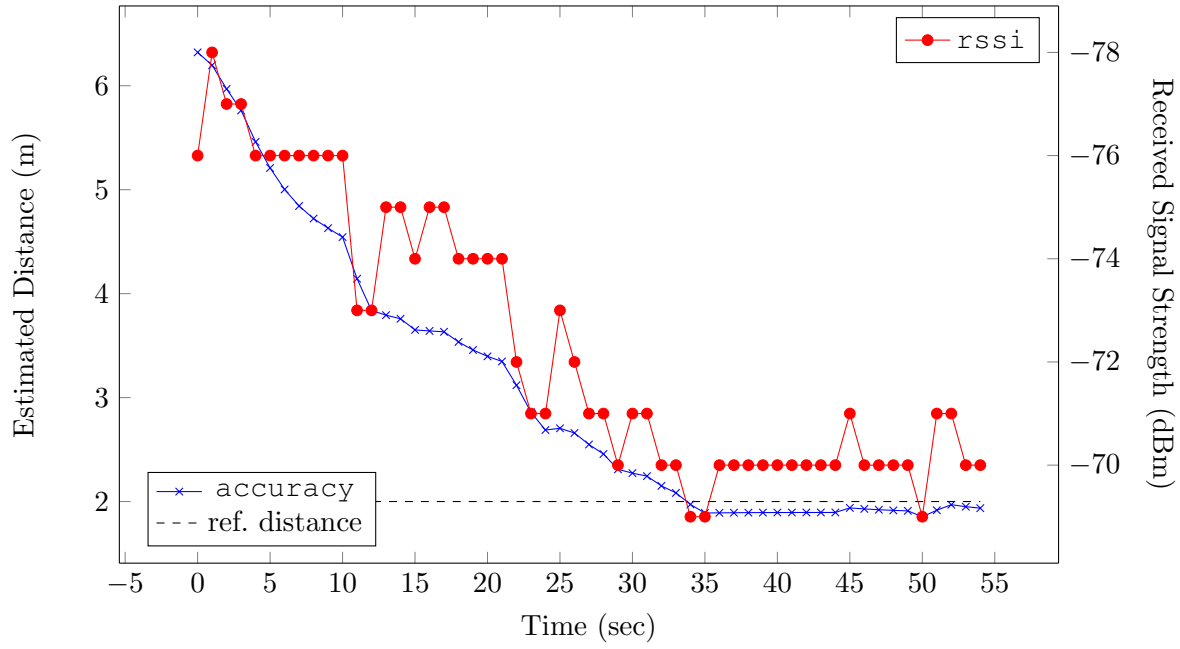


Figure 3.3: Dependency of the `CLBeacon` object's `accuracy` and `rssi` property over time with an actual distance of 2 m. The recording started directly after positioning the smartphone.

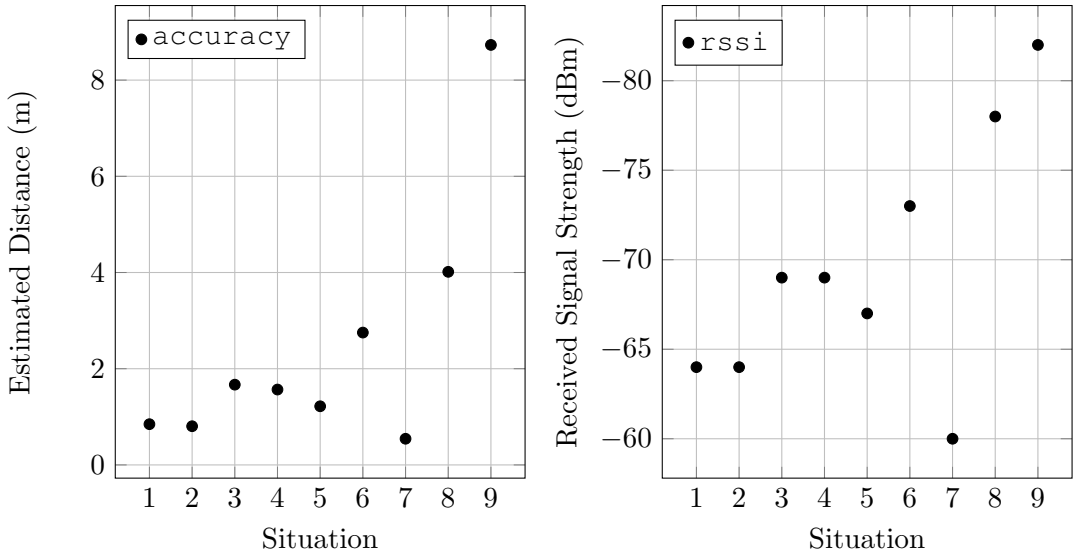
First, the distance estimations' quality and its error in different situations is evaluated. Afterwards, the result of an experiment which uses only multilateration to determine a user's position is presented.

3.4.1 Distance Estimation

The measured distances provided by the `CLBeacon` object's `accuracy` property varies in dynamic, but also in steady and unobstructed environments, due to the following reasons.

Filtering

The first reason is, that the measured RSS is not directly used to calculate the `accuracy` value. The CoreLocation framework seems to apply a filter to the `CLBeacon` object's values. To test this the `CLBeacon` object's properties were recorded over time. During the test the smartphone was positioned with a distance of 2 m to the beacon. The data recording started directly after positioning the smartphone. Figure 3.3 illustrates the recorded `rssi` values and their corresponding `accuracy` values over time. The graph shows, that the `accuracy` values slowly converge against the actual distance of 2 m. Furthermore, it shows that outliers in the `rssi` property do not have a direct impact on the estimated distance. Consequently, it is very likely that CL filters the values provided by the `CLBeacon` object. Furthermore, it shows that the `accuracy`, i.e. estimated distance, not only depends on the corresponding `rssi` value.



(a) The average estimated distance in different situations.

(b) The average RSS in different situations.

Figure 3.4: The impact on the beacon signal's quality, i.e. the `CLBeacon` object's `accuracy` and `rssi` values in different situations, such as obstacles between smartphone and beacon, and the smartphone's orientation relative to a beacon. For each value 60 – 100 measurements were recorded with an actual distance of 1 m.

Attitude & Obstacles

The second reason is the device's attitude relative to the beacon. Figure 3.4 illustrates the impact on the estimated distance and the RSS in typical situations. The actual distance between smartphone and beacon is 1 m. The measurements were taken indoor. For each situation 60 – 100 values were recorded.

Situations:

- *(1)* Horizontal iPhone in front of vertical beacon with LOS.
- *(2)* Horizontal iPhone sideways to vertical beacon with LOS.
- *(3)* Horizontal iPhone with angle of 45° below vertical beacon with LOS.
- *(4)* Vertical iPhone in front of vertical beacon with LOS.
- *(5)* Horizontal iPhone in front of horizontal beacon with LOS.
- *(6)* Horizontal iPhone in front of vertical beacon with person in between.
- *(7)* Horizontal backwards turned iPhone in front of vertical beacon with LOS.
- *(8)* Horizontal backwards turned iPhone in front of vertical beacon with person in between.

- *(9)* Horizontal iPhone in front of vertical beacon with 30 cm ferro concrete wall in between.

Situation 1, 2, 5 and 7 shows, that if the iPhone is horizontal in front of, or sideways of the vertical beacon with LOS, the RSS attenuation is the lowest. In contrast, if the beacon's signals are received with a different angle as shown by situation 3 and 4 the attenuation increases.

Furthermore, Figure 3.4 depicts the third reason, which is static obstacles such as walls or the user's own body. A human body between the phone and a beacon, shown in situation 6 and 8, has a much higher impact. The estimated distance increases by approximately 3 m. Whereas a ferro concrete 30 cm wall, shown in situation 9, drastically impacts the distance. The smartphone estimates a distance of 9 m instead of 1 m.

The above illustrated scenarios are very common. The problem is, that without knowing its own exact position, the beacons attitude, and the positions of all obstacles one cannot distinguish, whether an estimated distance is impacted by the attitude, a wall, or a person.

Distribution and Error

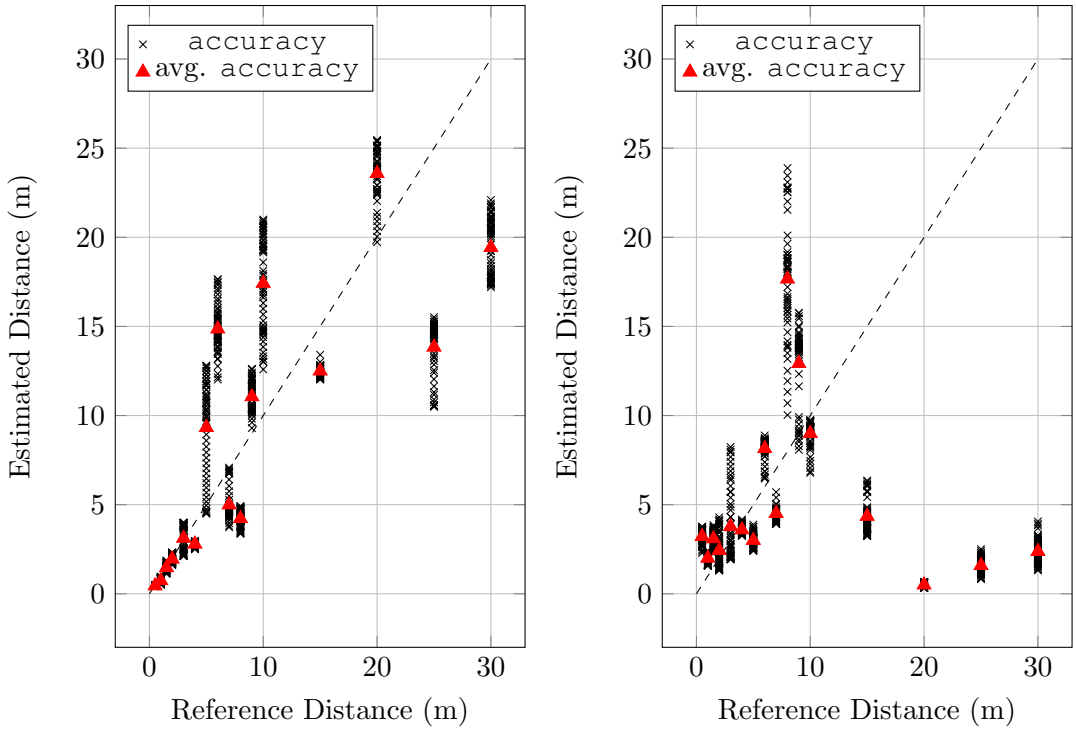
To depict that the estimated distance also varies in steady unobstructed environments with LOS, the `accuracy` values for different reference distances in different environments were measured. For each reference distance 60 values were recorded. To do so, the same beacon was used, which was calibrated for each environment.

Figure 3.5a illustrates the collected `accuracy` values measured at $1, 2, \ldots, 5, 10, \ldots, 30$ m distance to the beacon. The measurements were taken outdoor with Line-of-sight between the beacon and the smartphone. The graph shows, that the average `accuracy` measured at $1-4$ m distance is very close to the actual distance. It also shows that the value's distribution for these distances is very low. For distances ≥ 5 m, the distribution increases and varies between small and large. Thus, it is not correlated to the reference distance. As mentioned before, the beacons manufacturer claims a maximum range of 40 m. In this scenario the beacon's signals could be received up to a distance of 42 m.

The exact same experiment was repeated in an indoor environment. The results are depicted in Figure 3.5b. Also in this case the average `accuracy` comes very close to the actual distance with a very low spreading for the first 4 distances. At 5 m the distribution and error increases and decreases, too. Interestingly, the value's distribution starts to decrease between $10-30$ m from very large to a much smaller value. At the same time the error grows because the smartphone estimates a very small distance due to the RSS; whereas, the actual distance is very large. The assumed reason is the test environment's construction form. The measurements were recorded in a straight, long, but narrow corridor in a basement. There are multiple doors made of steel on both sides of the wall, that maybe caused reflections.

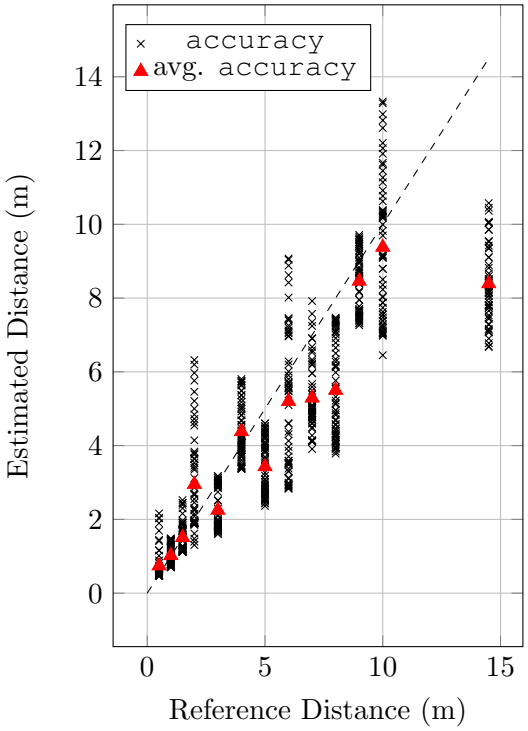
The measurements depicted in Figure 3.5c were recorded in a ≈ 15 m long and $8-11$ m wide lecture hall. It shows, that also the first few meters on average are matching best, but it also depicts, that there are sometimes huge differences in the estimated distance at the same position.

As mentioned in Section 2.1, it is necessary to know the uncertainty of a measurement to implement a Particle Filter. Thus, the distance estimation's uncertainty were determined based on the data shown in Figure 3.5a and 3.5c. The Gaussian's parameters to model the distance estimation's uncertainty are depicted in Figure 3.6. The two dashed straight lines



(a) Measured outdoor

(b) Measured in basement (HTWG-KN F)



(c) Measured in lecture hall (HTWG-KN F007)

Figure 3.5: The distribution of the estimated distances, i.e. the measured `accuracy` values, depending on the actual distance and the environment.

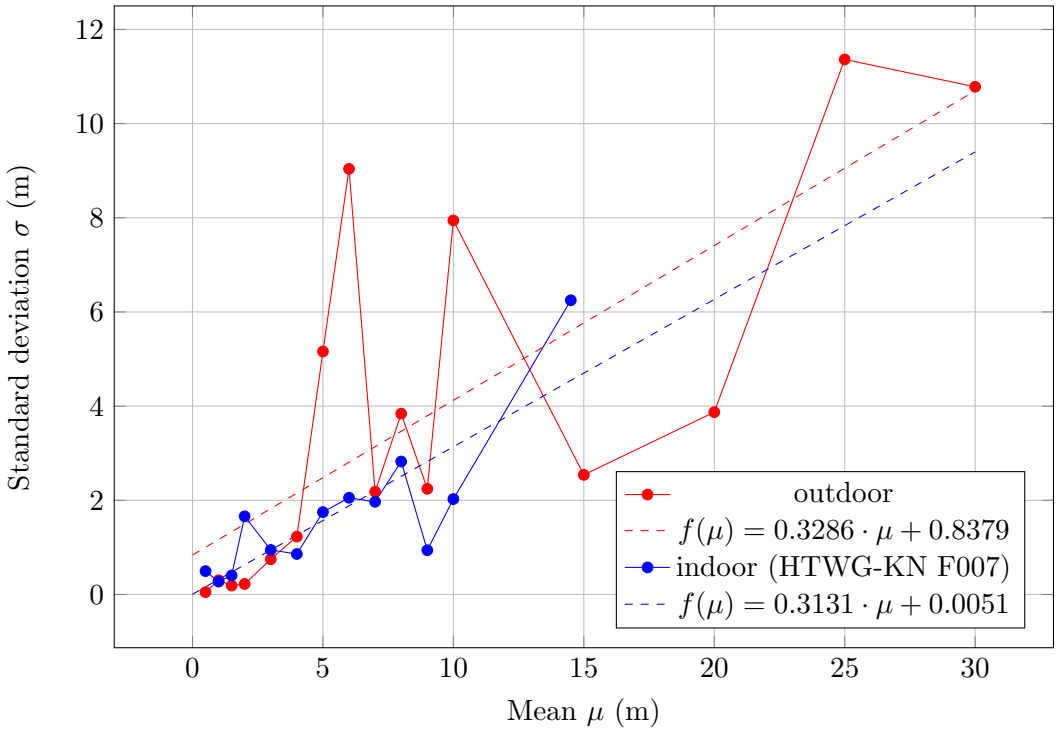


Figure 3.6: The distance estimations' uncertainties, modeled as Gaussians, depending on the actual distance. The two straights approximate the uncertainties using linear regression.

show the linear function that approximates the standard deviation σ depending on the mean μ best.

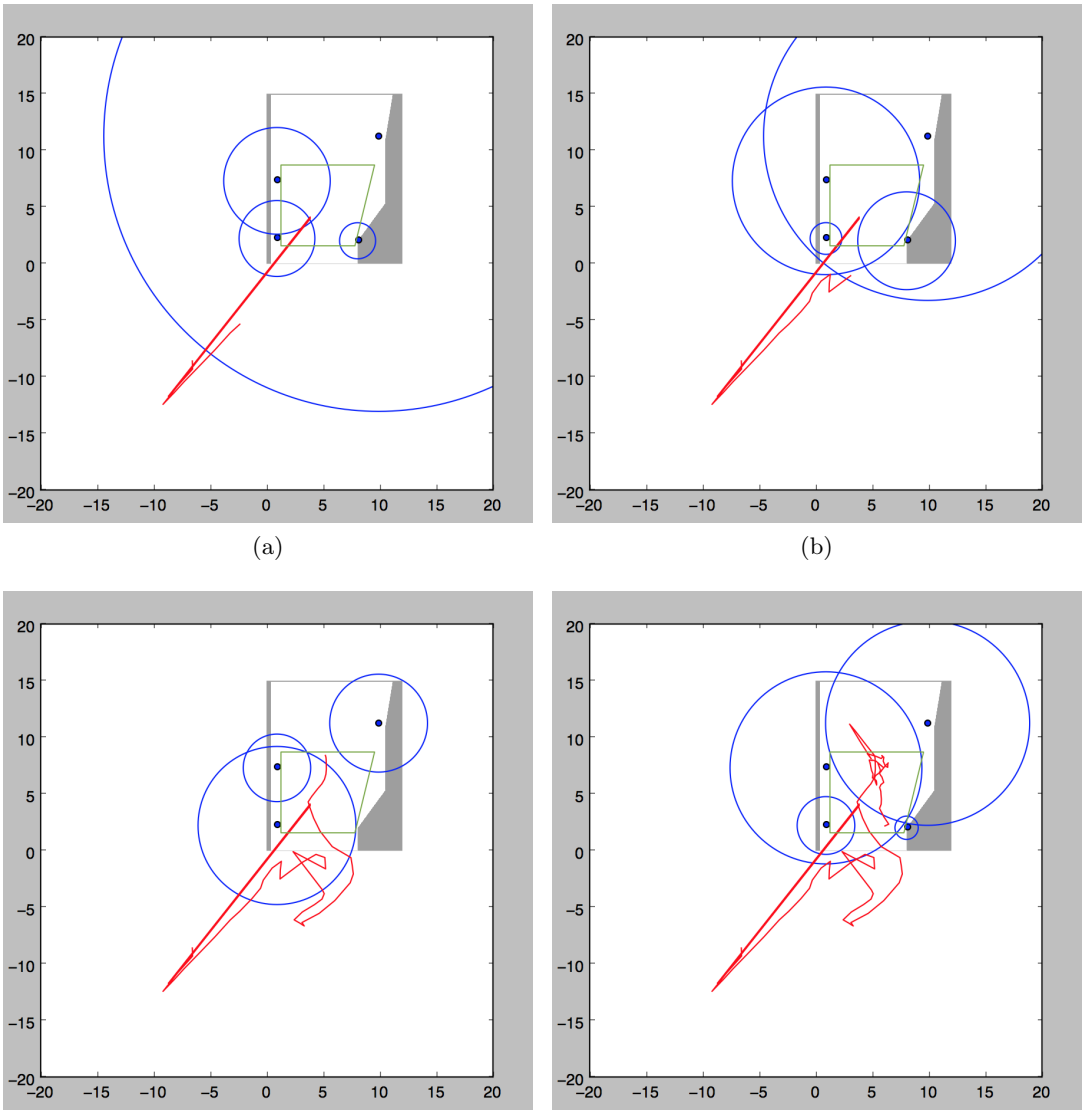
3.4.2 Multilateration

After evaluating the distance estimation's quality of one beacon, some beacons were deployed in a lecture hall to get a better feeling of how good the distance estimation actually works. On one hand, the goal was to visualize the estimated distances on a map to proof their plausibility, and on the other hand, to test if indoor self-localization with smartphones is possible by just using iBeacons without integrating additional sensor or map data.

As already mentioned in Section 2.3.1, *multilateration* is a method to estimate a position by using the distances to several transmitters. If one uses only one sender then the receiver's position is somewhere on a circle around the transmitter with the estimated distance as radius of the circle. By using two tranmitters t_1 and t_2, the two circles around the transmitters can have zero, one or two intersections, if the position of $t_1 \neq t_2$. Thus, the receiver's position must be at one of the circles' intersections. By adding a third transmitter t_3 the three circles have either no or exactly one, common intersection point, which is the receiver's position. This applies also to more than three transmitters.

To be able to use the implementation with variable transmitter count the *Least Square* method to determine the receiver's position was used. Another advantage is that the least square method can also estimate a position if no common intersection point exists.

For the experiment the same lecture hall as before was used. There, four beacons were



(a) (b)

(c) (d)

Figure 3.7: Four successive screenshots of the indoor self-localization experiment in a lecture hall (HTWG-KN F007) using multilateration. The screenshots show a map of the lecture hall, the walked path in green, the beacons as blue filled circles and their estimated distances as large blue circles. The red line depicts the estimated path.

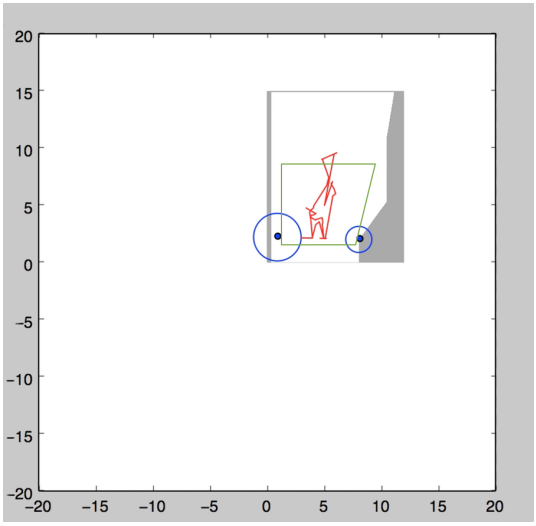


Figure 3.8: Screenshots of the indoor self-localization experiment in a lecture hall (HTWG-KN F007) using multilateration by only ignoring estimated distances more than 5 m. The screenshots show a map of the lecture hall, the walked path in green, the beacons as blue filled circles and their estimated distances as large blue circles. The red line depicts the estimated path.

positioned. Then the test device, which is an iPhone 5S, was used to recorded the received beacon identifiers with their corresponding estimated distances, i.e. the `accuracy` values, provided by CL, during a test walk on a defined path. To visualize the estimated distances to the beacons and the user's estimated position, i.e. the walked path, a small Python program was used. Figure 3.7 shows four screenshots of the program's output. Each of it shows a map, which shows the lecture hall walls in grey. The white space in the hall is free space. The filled small blue circles next to the walls are the four beacons. The beacon is only shown if its signal was received during this time step. The big blue circles, are the circles around the beacons, where the radius represents the estimated distance to the beacons. The test person walked on the green tetragon, beginning at the bottom right corner in counter clockwise direction by holding the test device in front of oneself. The beacons were positioned at approximately the same height as the device in the test person's hand. The red line shows the estimated path.

The first screenshot depicted in Figure 3.7a shows, that the estimated distance sometimes is completely wrong, as shown by the blue circle around the top-right beacon. It also shows that the estimated position is somewhere several meters outside of the lecture hall. The second screenshot depicted in Figure 3.7b, shows, that the distance to the top-right beacon shrinks, but the one to the top-left massively grows. The distances to the other two beacons seem to be plausible. Figure 3.7c shows a plausible position. It also shows, that in this step the beacon's signal, positioned at the bottom-right corner, was not received. The last screenshot, depicted in Figure 3.7d shows also a more or less plausible position. The distances, except the one to the beacon, positioned at the bottom-left corner, seem to be plausible.

The experiment was repeated with the same setting and recorded data, but only used those beacon signals where the `accuracy` was < 5 m. Signals with larger estimated distances were ignored. Figure 3.8 shows the result. The estimated path actually is inside of the lecture hall. It is not possible to determine that the test person walked on the green path, but ignoring

values more than 5 m seems to improve the location estimation.

This experiment shows, that due to the very unreliable distance estimations the beacons' signals must be heavily influenced. It also shows that pure multilateration, without filtering the data and without integrating additional data sources, is not sufficient for indoor self-localization.

3.4.3 Summary

The results show, that the distance estimation not just depend on the beacons calibration, it also depends on the buildings construction form, the phones relative orientation to the beacons, and the obstacles in between.

It also points out, that the `CLBeacon`'s `accuracy` i.e. the estimated distance, comes very close to the actual distance up to 4 – 5 m with LOS. Larger distances cause a much higher spreading and thus, a larger error. Physical material like humans and walls cause attenuation, which results in a larger distance estimation than it actually is. Besides that, the estimated distance is sometimes very small compared to the actual distance.

To conclude, the estimated distances are very unreliable. Thus, their plausibility should be proven to filter the measurements. But due to the fact, that the validity cannot be proven without knowing the exact position of the receiver and the properties of all other influencing factors, which is not a trivial problem in large buildings, the only one possible improvement that could be applied, is to ignore beacons with an estimated distance larger than 5 m. This solves of course not the problem with those beacons, which appear to be closer than they actually are. But from my point of view no real solution exists to solve this problem.

Chapter 4

Built-in Sensors

This chapter describes first the suitable built-in sensors of the current smartphone generation. Afterwards, it provides an overview of the appropriate APIs to access either raw or processed sensor data. Finally, the data provided by the sensors' APIs is evaluated.

4.1 Built-in Sensors

Todays smartphones contain several sensors to provide applications the possibility to measure physical properties of the smartphone's environment. By measuring environmental properties applications can process and react on them. This section gives an overview of available sensors of the latest smartphone generation that are suitable for this project. It also describes the physical properties that can be measured by them and their functionality.

Magnetometer Today, most smartphones have a built-in magnetometer to measure the strength and direction of a magnetic field. Typically, the magnetometer is built with at least one hall sensor which usually measures the magnetic field, i.e. the magnetic flux density, in micro teslas. For instance, Apple's iPhones includes a three-axis magnetometer; thus, it is able to determine the orientation in three-dimensional space, which is, for example being used, by the digital compass (Pham, 2012; Apple Inc., 2014d).

Accelerometer The accelerometer, which is also implemented in most smartphones, measures the acceleration applied to the device. The measured acceleration is the gravity plus the acceleration that the user applies to the device, measured in $\frac{m}{s^2}$. Thus, the measured acceleration is $1\,\mathrm{g} = 9.81\,\frac{m}{s^2}$ if the device is stationary, i.e. no user acceleration is applied to the device. Typically, smartphones include a three-axis accelerometer to measure the acceleration along the three spatial axis to determine for example the device's tip and tilt (Pham, 2012; Apple Inc., 2014d).

Gyroscope Modern smartphones usually include a three-axis gyroscope to measure the rotation rate along the device's spatial axis. Rotation rate is measured in $\frac{rad}{s}$. Thus, the gyroscope can be used to determine the device's attitude. Like every sensor data, the gyroscope's measurements include some bias. Hence, determining the device's attitude with the raw gyroscope measurements includes a growing drift over time (Pham, 2012; Apple Inc., 2014d).

Barometer Some of the latest smartphones also include a barometer to measure air pressure. It is measured in *kilopascals*. Thus, the relative change in altitude can be calculated. The barometer can be used for hiking apps, to determine the change in altitude for example (Apple Inc., 2014d).

4.2 APIs — iOS 8.1

To use the sensors within an application running on the device the OS provides different APIs. As mentioned before, the raw sensor data usually include some bias; for instance, the device's hardware biases the measured magnetic field. Apple's API provides the possibility for all mentioned sensors, to access either raw sensor data including bias, or already filtered and processed data with less bias. This section provides an overview of the suitable higher level APIs with focus on Apple's iOS 8.1 platform.

Motion Processing With the iPhone 5S Apple introduced in 2013 a motion coprocessor, called M7. The latest iPhone, iPhone 6 / iPhone 6 Plus, contains an improved version named M8, with improved accuracy. Additionally, it includes a barometer, which is processed by the motion coprocessor. The key features of the CoreMotion (CM) framework and the motion coprocessor is sensor fusion, energy efficiency, and motion awareness, over the past seven days (Pham and Chow, 2014).

CM implements algorithms to filter and combine the input of multiple sensors. Thus, the APIs can provide more accurate results for values such as the device's attitude. Together with the motion coprocessor it can offer more convenient interfaces to developers with already processed data, such as step counting and motion activity classification (Pham and Chow, 2014).

Earlier iPhones used their CPU to gather and process sensor data, which is very energy consuming and inefficient, because it eats up a lot of CPU time. According to Pham (2012), one of Apple's CM engineers, the motion coprocessor is very energy efficient. He mentions, that the energy consumption of 24 hours motion processing, e.g. motion activity classification, or the pedometer, is equal to a three minute FaceTime[1] call.

Motion awareness gives applications the possibility to query the user's motion data for the past seven days.

Push and Pull Interface The following APIs provide to types of interfaces.

The *push* interface continuously provides the application with new sensor data, usually within a certain time interval. To do so, the developer needs to provide and register a callback function. This function gets passed the new data.

The *pull* interface is designed to query historical data within a specified range of time; for instance, the past seven days. According to Pham and Chow (2014), querying historical data is more accurate than the data provided via the push interface, because the filters can operate better on a larger data set. They also mention, that some values are adjusted, i.e. they adapt to the user's behavior over time; for instance, the pedometers stride estimation.

Both interfaces provide their data asynchronously to the developer's application.

4.2.1 MotionActivity

Apple's `CMMotionActivityManager`, which is part of the CM framework, is able to detect and classify different activities, such as walking, running, cycling, etc. For the activity detection and classification the motion coprocessor is used together with the accelerometer (Pham and Chow, 2014).

[1] FaceTime is Apple's video telephony application.

Device scenarios	stationary	walking	running	automotive	cycling	unknown
On a table	**true**	false	false	false	false	false
Person checking email	false	false	false	false	false	false
Person walking	false	**true**	false	false	false	false
In idling vehicle	**true**	false	false	**true**	false	false
In moving vehicle	false	false	false	**true**	false	false
After reboot	false	false	false	false	false	**true**

Table 4.1: Example scenarios illustrating the `CMMotionActivity` object's activity classification (Pham and Chow, 2014).

As mentioned before, the CM framework's MotionActivity APIs provides a push interface to receive motion activity changes, and a pull interface to query historical data between two dates. Both APIs are reporting every single change in motion activity, that was detected.

Table 4.1 shows an overview of the different motion activities and illustrates some example scenarios. The states `walking`, `running`, `automotive`, `cycling`, and `unknown` are mutually exclusive motion activity types, whereas `stationary` is not mutually exclusive to the other types.
Thus,

- a device lying **on a table** does not move; hence, it is stationary.
- a **person checking email** usually does not hold its smartphone completely steady.
- a **person walking** can be detected across body location.
- in a **moving vehicle** the state is automotive.
- in an **idling vehicle**, e.g. in front of a stop sign, the state is stationary and automotive at the same time. The device needs to be mounted in car.
- immediate after a **reboot** the device's state is unknown because it first needs to collect some data to determine its real state.

As mentioned by Pham and Chow (2014), the `CMMotionActivityManager` has some latency depending on the activity and its location. For example, if a person is walking and holds the device in hand, it takes $\approx 5-10\,\text{s}$ to detect walking. If it is carried in a pocket it takes only $\approx 3-5\,\text{s}$. To detect walking takes relatively long compared to the running state. Running can be detected within a couple of steps. According to Pham and Chow (2014) one reason is that they can assume, that running persons does not check their emails or Facebook messages at the same time. If the smartphone is mounted in a car, the driving state can also be detected very fast. They also mention that the cycling state is very difficult to detect.

Another important point is the accuracy across body location. They point out, that the average accuracy across body location is always the same.

Every `CMMotionActivity` object which is reported to the application includes a `confidence` property. Confidence is measured at three different levels: `low` = 0, `medium` = 1, and `high` = 2. Thus, the more confident the `CMMotionActivityManager` is about a motion activity state the higher the confidence value. Usually, it increases over time if the activity type does not change (Pham and Chow, 2014; Apple Inc., 2014d).

timestamp	startDate	endDate	distance	steps
9.3564	0.0019	9.2981	4.1351	7
14.4272	0.0019	11.8354	4.1351	7
14.4357	0.0019	14.3802	9.0606	15
16.9566	0.0019	16.9193	9.7806	16
22.0541	0.0019	19.4658	9.7806	16
22.0553	0.0019	22.0038	13.5247	22
24.6033	0.0019	24.5478	14.2447	23
29.6747	0.0019	27.0876	14.2447	23
29.6754	0.0019	29.6292	18.5546	30
32.2152	0.0019	32.1722	19.2546	31

Table 4.2: Recorded pedometer example data with additional timestamp. Remark: To simplify the table, relative values for timestamp, startDate and endDate are used instead of the absolute timestamps. The *timestamp* column is actually not part of the `CMPedometerData`.

4.2.2 Pedometer

The pedometer's API is a component of Apple's CM framework. It provides the distance a person traveled over time in meters. The iPhone needs to be equipped with a motion coprocessor[2]. The motion coprocessor processes the accelerometer's data to count a person's steps and to estimate the person's stride length. The stride estimation adapts over time. Thus, the more often the pedometer is used the better the accuracy (Pham and Chow, 2014). According to Pham and Chow (2014), the pedometer has a consistent performance and accuracy across the phone's body location.

As mentioned before, there are two different interfaces to request the pedometer data. Via the pull interface the application can ask the pedometer component for the distance a person traveled by passing a start and end date. The second possibility is the push interface. With it the application can register a callback function to receive updates while the person is walking. If the user is walking or running the function is called every $\approx 2.5\,\mathrm{s}$, assuming the motion processor detects the taken steps. If it does not detect steps the handler is not called. The received steps and distances are the cumulative step count and distance since the start date, which is for all successive `CMPedometerData` objects the same (Pham and Chow, 2014).

According to Apple Inc. (2014d), the received `CMPedometerData` contains the following values:

- `startDate`, absolute start date
- `endDate`, absolute end date
- `steps`, i.e. step count, as integer
- `distance` estimated in meters
- `floorsAscended`, M8 coprocessor with barometer required
- `floorsDescended`, M8 coprocessor with barometer required

[2]Note: iPads with motion coprocessor do not provide step counting and distance estimation.

Table 4.2 depicts a recorded test walk. The first column shows a relative timestamp to the recording start, when the data was received by the application.

The table depicts some entries with the same step count and distance, but with different timestamp and endDate. Between these data sets with the same step count and distance the user actually stopped walking, i.e. the pedometer did not recognize any steps. At the point in time, where the pedometer recognizes that the user continues walking, it pushes the same step count and distance with different endDate. This repeated data is usually passed to the callback function, together with the successive pedometer data (Table 4.2, Row 2 – 3). Their end dates usually have a difference of $\approx 2.5\,\mathrm{s}$; whereas, the timestamp is roughly the same.

If the user puts the application into background and resumes it later on, CM immediately provides the application with an update, i.e. before the common $\approx 2.5\,\mathrm{s}$ (Pham and Chow, 2014).

4.2.3 DeviceMotion

As mentioned before, the iPhone contains a magnetometer, which is good for heading, an accelerometer, which can be used to calculate the phones tip and tilt, and a gyroscope to measure a device's rotation rate. To calculate the device's attitude, its position in three-dimensional space by using raw sensor data is very difficult, among things like uncertainty and ambiguity. Fortunately, Apple's CM framework does not only provide raw sensor data, it also provides computed data like the `userAcceleration` and the device's `attitude` stored in their `CMDeviceMotion` component. To calculate these values, a technique, called *sensor fusion*, is used. It combines the measurements of the before mentioned sensors, to reduce bias and uncertainties, and to remove ambiguities. Besides the `attitude`, the component also provides other data, such as:

- **gravity**, in complete unconstraint motion
- **user acceleration** is the acceleration without gravity
- **rotation rate**, with compensated bias
- **magnetic field**, with removed disturbances

The mentioned values are all by-products of the device's attitude calculation (Pham, 2012; Apple Inc., 2014d).

Attitude The device's attitude is provided in three different mathematical forms, Euler angles, Quaternions, and as rotation matrix.

To start receiving the device's `attitude` via the `CMDeviceMotionManager`, a reference frame needs to be specified. Figure 4.1 shows the iPhones local coordinate system, which is important to know, to understand the differences between the four possible reference frames. The four `CMAttitudeReferenceFrames`, illustrated in Figure 4.2 have the vertical z-axis aligned with the gravity in common. According to Pham (2012), the specified reference frame also specifies the type of sensor fusion. The following reference frames exist (Pham, 2012; Apple Inc., 2014d):

- **XArbitrary** (Figure 4.2a): The x- and y-axis are unspecified. The initial position fixes the x and y orientation. There is no correction of the heading; it drifts over time.

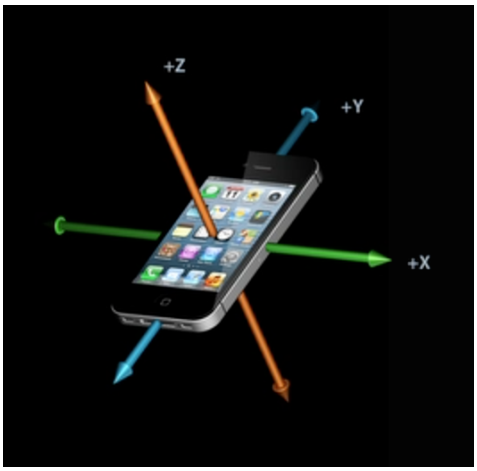
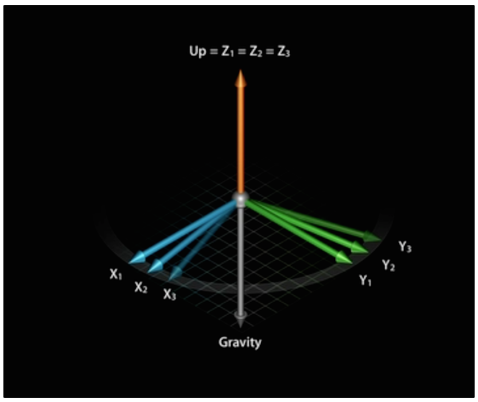
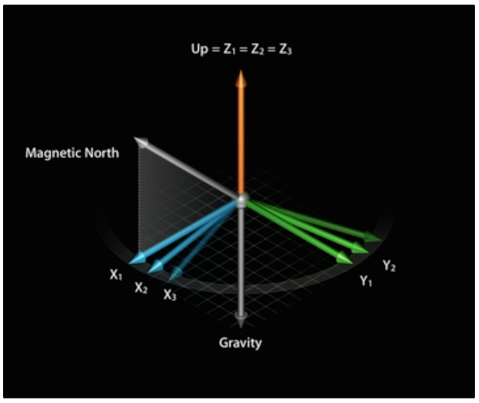


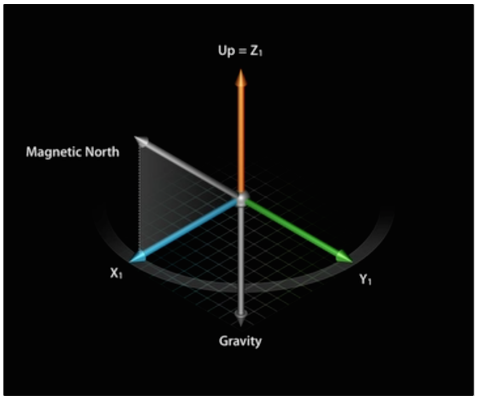
Figure 4.1: The iPhone's local coordinate system (Pham, 2012).



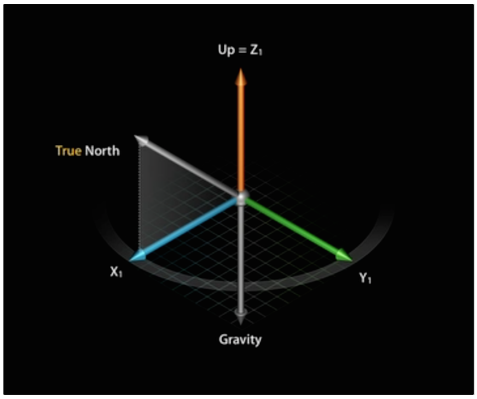
(a) XArbitrary



(b) XArbitraryCorrected



(c) XMagneticNorth



(d) XTrueNorth

Figure 4.2: The CoreMotion framework's attitude reference frames (Pham, 2012).

- **XArbitraryCorrected** (Figure 4.2b): The x- and y-axis are unspecified. The initial pose fixes the x- and y-orientation. Additionally, the magnetometer is used to correct heading and to improve yaw accuracy over time.
- **XMagneticNorth** (Figure 4.2c): The x-axis is absolutely tagged to magnetic north. It uses the compass for orientation.
- **xTrueNorth** (Figure 4.2d): The x-axis is absolutely tagged to true north[3]. It uses the compass to determine true north.

4.2.4 Compass

The compass is part of the CL framework. It provides the device's *magnetic heading* and also its *true heading* in degrees. The heading depends on the specified device orientation, such as portrait, landscape left, or landscape right mode (Apple Inc., 2014c). According to Pham (2012), the compass uses sensor fusion and not only the magnetometer.

To access the device's heading, the `CLLocationManager` must be configured. It informs the application when the heading changes. The developer can specify a heading filter. Thus, a new heading is only reported, if the heading filter is exceeded (Apple Inc., 2014c).

As already mentioned before, the magnetometer measures the magnetic field, which is not only the earth's magnetic field. It is biased; for example, by iron bars and AC current. To reduce bias, the compass asks the user to calibrate it, by moving the device in a specific manner. CL by itself can detect whether or not the compass needs calibration. Thus, it automatically shows the user the calibration view (Apple Inc., 2014c).

4.3 Evaluation

The purpose of this section is to find out if the above referenced sensors and APIs are suitable for this project. In the following the provided data is evaluated. Furthermore, the above referenced statements made by Apple's engineers are verified.

4.3.1 MotionActivity

As mentioned above, the purpose of the `CMMotionActivity` component is to detect and classify a user's activity. To prove the statement concerning the recognition duration of the walking state made by Apple's engineer Pham and Chow (2014) some test walks were performed. The results of the experiment show, that the claim of a recognition time of 10 – 15 s for the walking state is very optimistic. Table 4.3 shows the beginning of one of these test walks.

From the start, the test user walked with constant speed and with the phone in hand. It shows that the component recognize when the device was moving (not stationary) even before the application started (Table 4.3, Line 1). In reality, the application was started before starting to walk, but *not stationary* means that the user just somehow moves the device. After ≈ 10.7 s the component increases its confidence for the current state (Table 4.3, Line 2). Another ≈ 5 s later it set the confidence to `high` (Table 4.3, Line 3); thus, the

[3]The earth's magnetic field moves. Thus, true and magnetic north are not the same. Maps are oriented towards true north.

start date	confidence	unknown	stationary	walking	running	automotive	cycling
-2.0246	0	false	false	false	false	false	false
10.7026	1	false	false	false	false	false	false
15.7922	2	false	false	false	false	false	false
25.9676	0	false	false	**true**	false	false	false
32.3255	1	false	**true**	false	false	false	false
32.6434	1	false	false	false	false	false	false
33.2792	1	false	**true**	false	false	false	false
...	...	...	...	...	...	...	...

Table 4.3: Recorded `CMMotionActivity` data during a test walk. The user stopped at $\approx 32\,\text{s}$. Remark: To simplify the table the start date is shown as relative timestamp instead of an absolute date.

component is very confident that the user somehow moves the device. After $\approx 25\,\text{s}$ of walking the component finally detects with `low` confidence, that the user actually walks (Table 4.3, Line 4).

Compared to detecting the users walking state, it detects nearly in real-time, that the user stops walking after ≈ 32 (Table 4.3, Line 5). At this point in time the user holds the device very steady. The component only switches between stationary and not stationary, in a very short time period (Table 4.3, Line 5 – 7).

This experiment shows that the user needs to actually walk a very long distance until the phone recognizes that the user is walking. It also shows that the user has to hold the device very steady until the device claims that the device is stationary. Consequently, CM's `CMMotionActivity` cannot really be used to determine a user's walking state for short distances. Also the stationary state cannot be used to claim that the user is not walking due to the high sensibility.

4.3.2 Pedometer

To evaluate the accuracy of CM's `CMPedometer` component, test walks of different distances were performed, to compare the actual distance and step count with the measured values. Due to the fact that users usually carry their smartphone in their trouser pockets or hold it in their hand, the experiment was performed for both positions. Figure 4.3 shows the measured distances and the corresponding step counts for different distances with the phone in hand. The results of the same experiment with the phone in the trouser pocket are shown in Figure 4.4.

The experiments visualization shows the measurements dispersion. Sometimes, the `CMPedometer` overestimates or underestimates the distance, i.e. the step count. The experiment also shows, that the phone on average underestimates the values, if the user holds the device in hand. If the user walks with the phone in the trouser pocket the device also underestimates the measured distances; whereas, the measured step counts are more precise.

Figure 4.5 depicts the `CMPedometer`'s standard deviation σ depending on the distance and phone's position. To model the uncertainty for any walked distance, depending on the smartphone's position, linear regression based on the measurements was used. Initially, I expected a higher accuracy of the estimated distance if the user carries the phone in the trouser pocket, but the diagram prooves the opposite. Consequently, σ is significantly smaller

startDate	endDate	distance	steps
0.0019	8.2329	4.8273	7
0.0019	10.7811	7.9850	11
0.0019	13.3238	11.4785	16
...	...	...	...
0.0019	31.1349	37.2110	50
0.0019	54.0149	37.2110	50
0.0019	56.5503	41.4032	56
...	...	...	...
0.0019	69.2608	57.2046	78
0.0019	84.5138	57.2046	78
0.0019	87.0569	64.8046	88

Table 4.4: Recorded `CMPedometer` example data. Remark: To simplify the table relative timestamps for startDate and endDate are used. All values except the steps are truncated.

if the user holds the phone in hand.

Besides the pedometer's accuracy also the duration until the component detects that the user is walking and thus delivers the first distance estimation was evaluated. The `CMPedometer` component on average needs 5 – 10 s to deliver the first estimation, which is also consistent with a statement made by Pham and Chow (2014). The first estimation typically contains a step count of 6 – 8. Once it detects that the user is walking, it continuously delivers updates every ≈ 2.5 s.

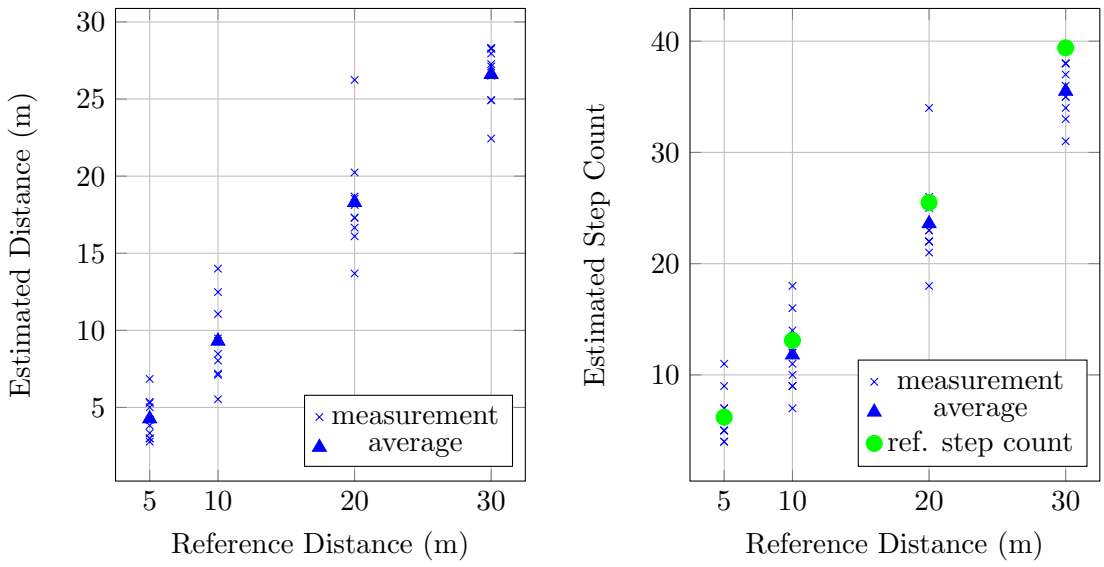
Another interesting value is the time it takes the pedometer to recognize that the user continues walking after a short break of ≈ 10 – 15 s. Against my expectation, it takes approximately the same amount of time and also the same amount of steps as the component requires to recognize it the first time. This behavior is depicted in Table 4.4. First it shows the amount of time and steps it took the component to recognize that the user is walking (Table 4.4, Line 1). After 50 steps the user took a short break of around 15 s (Table 4.4, Line 5). Furthermore it shows that the component requires 6 steps to recognize that the user continued walking (Table 4.4, Line 7). The third data block shows another break that the user made during the test walk (Table 4.4, Line 9 – 11).

4.3.3 Heading

As mentioned earlier, the solution's localization algorithm, introduced in Chapter 5, needs heading for the motion tracking to determine in which direction the user is moving. To get the devices' heading, either the `CMAttitude` provided by CM, or the `CLHeading` provided by CL can be used. In this section first the `CMAttitude`'s data is evaluated and afterwards the data provided by `CLHeading`.

CMAttitude

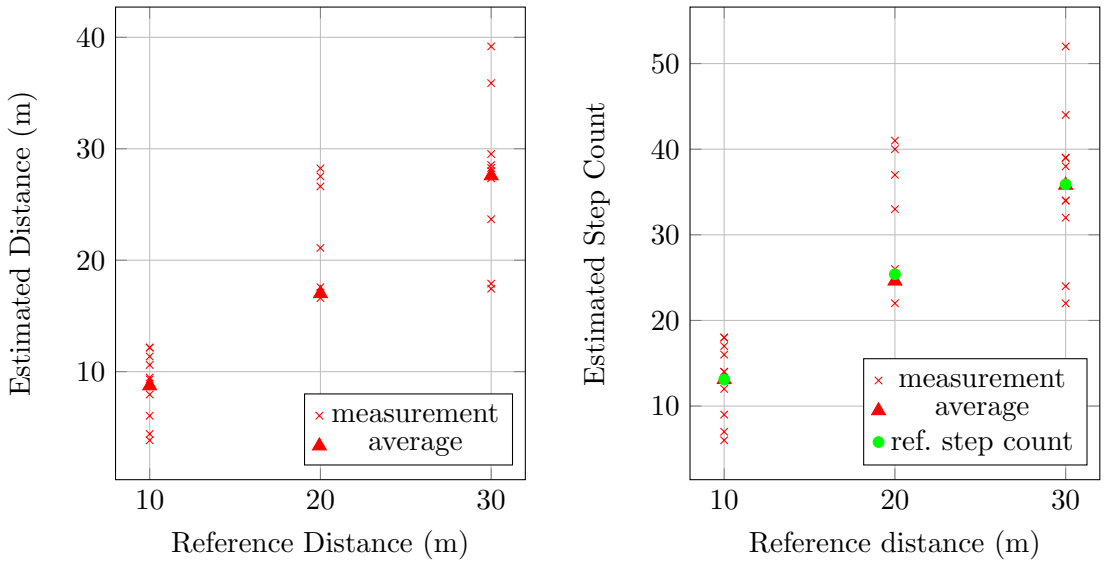
As mentioned before, the `CMAttitude` data is provided as Euler angles, Quaternions and as rotation matrix. To be able to compare attitude with compass heading, it is of advantage to transform the attitude into a heading in compass degrees. This means, if the device top is pointing towards magnetic north, the heading $\theta = 0°$, east $\theta = 90°$, south $\theta = 180°$, and west



(a) Estimated Distances

(b) Estimated Step Count

Figure 4.3: Measured distances and step counts for different reference distances estimated by CM's `CMPedometer`. The measurements were taken indoor on a hard floor with the smartphone in hand.



(a) Measured Distances

(b) Measured Step Count

Figure 4.4: Measured distances and step counts for different reference distances estimated by CM's `CMPedometer`. The measurements were taken indoor on a hard floor with the smartphone in trouser pocket.

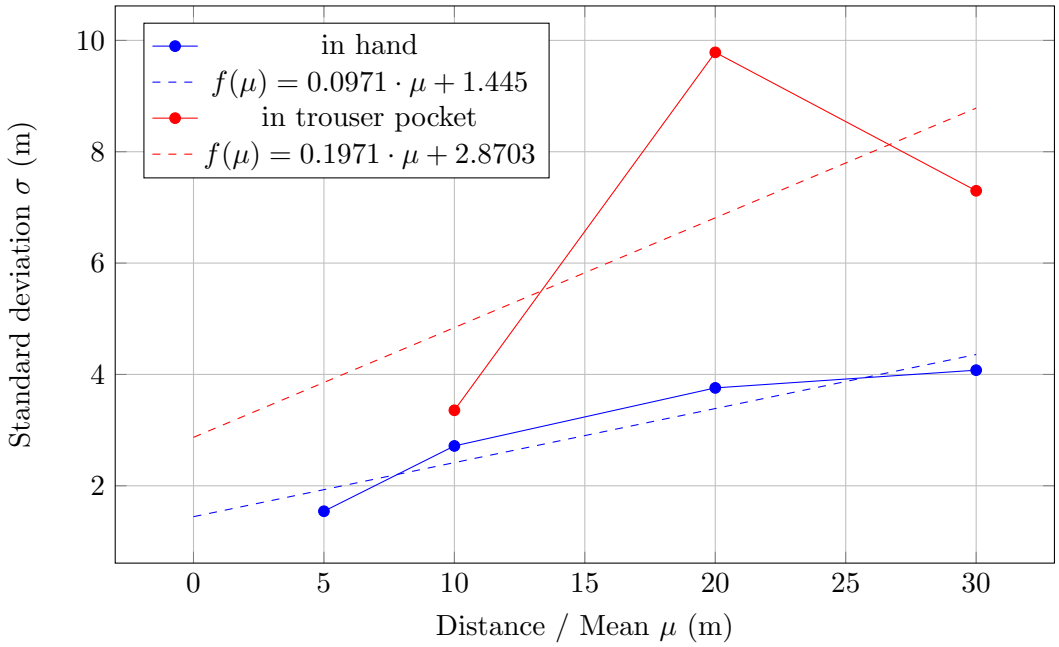


Figure 4.5: The `CMPedometer` component's uncertainty and their approximation using linear regression depending on the walked distance and the phone's position.

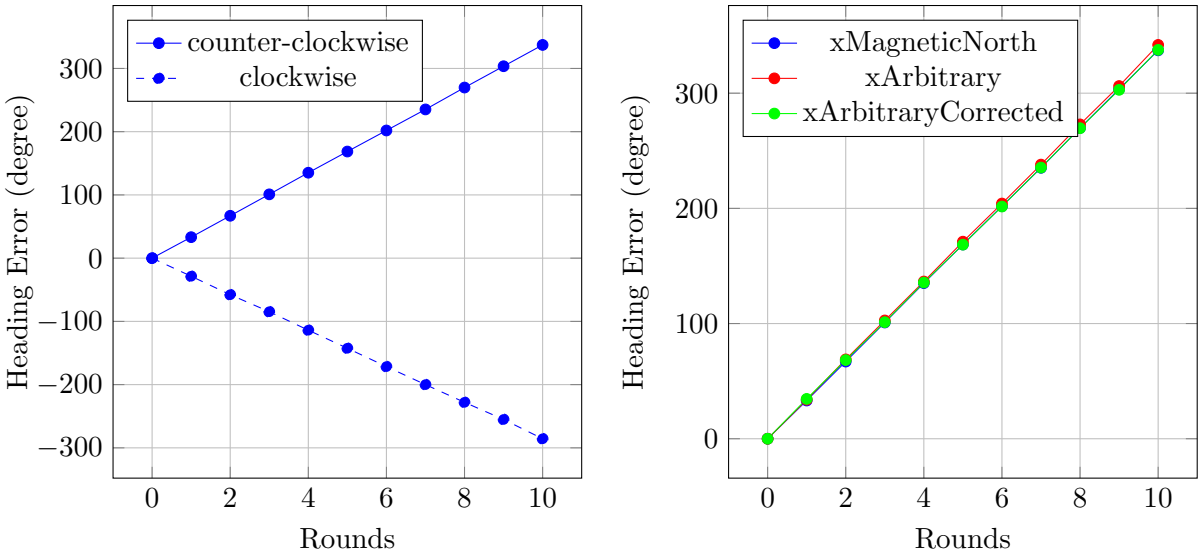
$\theta = 270°$. Equation 4.1 shows the 3-dimensional rotation matrix provided by `CMAttitude` as specified in the CM's documentation (Apple Inc., 2014d). The calculation of θ in compass degrees, where $m_{1,2}$ and $m_{2,2}$ describe the transformation of the phone's y-axis by the rotation around its z-axis, is shown in Equation 4.2.

$$M_{3,3} = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \tag{4.1}$$

$$\theta = (\pi + \operatorname{atan2}(m_{2,2}, m_{1,2})) \cdot \frac{180.0}{\pi}, \quad \text{for } m_{2,2}, m_{1,2} \neq 0 \tag{4.2}$$

As mentioned earlier, the `CMAttitude`'s values depend on the specified reference frame, which also affects the sensor fusion algorithm. To be able to easily compare the calculated heading θ directly with the compass heading the `xMagneticNorth` reference frame is used. Thus, z is aligned to gravity and x points towards magnetic north.

The most important requirement for heading is long-term accuracy. To test this, the test person walked 10 times around a small table in the same direction and recorded the heading after each round. To record the heading after each round the device was put on the table at the exact same position. Thus, the heading should be roughly the same after each round. Figure 4.6a illustrates the error of each measurement compared to the initial measurement. First, the test person walked counter-clockwise around the table. An enormous drift of $\approx 30°$ after each round was observed. This sums up to a total drift of $\approx 300°$ after 10 rounds. To confirm this, the person walked also clockwise around the table and observed the same amount of drift in the opposite direction. Apple's engineer, Pham (2012), explicitly mentioned that the magnetometer is used to provide long-term yaw accuracy, if the `xArbitraryCorrected`



(a) `xMagneticNorth` Reference Frame

(b) Different Reference Frames

Figure 4.6: Depicts the `CMAttitude`'s heading drift using the `xMagneticNorth` reference frame in clockwise and counter-clockwise direction. Furthermore, it compares the heading drift in counter-clockwise direction using different reference frames. To evaluate this a test person walked 10 times around a small table in clockwise and counter-clockwise direction.

reference frame is specified. Thus, the same test was repeated for the `xArbitrary` and the `xArbitraryCorrected` reference frame. Then the results were compared with the results from before, shown in Figure 4.6b. There is nearly no difference in long-term accuracy between the three tested reference frames.

CLHeading

During the before mentioned experiment, shown in Figure 4.6a and 4.6b, the compass, i.e. `CLHeading`, values were recorded as well. Figure 4.7 shows the heading's error over time. The chart shows no drift over time compared to the data measured by `CMAttitude`. However, it also shows that the compass can be biased by other magnetic fields. The outliers in the experiment with counter-clockwise direction led me to assume that another magnetic field biased the before measured magnetic field.

That the compass reacts on other magnetic fields, than the earth's magnetic field can easily be shown by moving a small magnet around the phone. In contrast to the compass, the `CMAttitude` does not react to other magnetic fields, like the one of a magnet.

According to the 40 measurements depicted in Figure 4.7, the compass's standard deviation σ amounts to 3.1°. During the experiment the `CLHeading` components heading filter was set to 1.0°.

4.3.4 Summary

As advertised, the solution explained in Chapter 5, is based on MCL. Thus, motion tracking is required, which seems to be feasible by combining CM's distance estimation and CL heading.

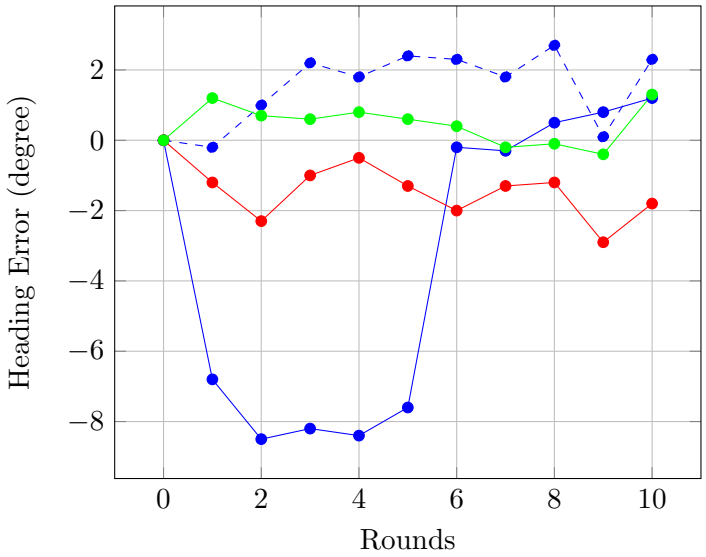


Figure 4.7: `CLHeading`'s, i.e. the compass' magnetic heading's drift. The test person walked 10 times around a small table in clockwise and counter-clockwise direction. The initial measurement is used as reference value. The heading filter was set to 1.0°. The measurements (lines colors) belong to the ones presented in Figure 4.6a and 4.6b.

If the `CLHeading` is heavily influenced by other materials, like iron bars, it could probably be reasonable to also integrate the discussed heading based on CM's attitude. But, due to its huge drift problem, only the change in heading, i.e. the rotation rate, can be used.

Furthermore, CM's motion activity classification seems to be useless for the later presented solution due to its huge latency.

Chapter 5

Localization Algorithm

This chapter presents the solution's implementation. As mentioned before, it is based on Monte Carlo Localization (MCL). The terminology used in this chapter builds upon the PF's introduction in Section 2.2.

First the reasons for choosing MCL instead of another algorithm are outlined. Afterwards, an overview of the system's setup is provided. Next the algorithm's *motion model* is described, followed by a detailed insight in the solution's *measurement model*. In the end, the algorithm's implementation is described, which builds upon the motion and measurement model.

5.1 Design Decision

This section outlines the reasons for choosing Monte Carlo Localization instead of another algorithm, as introduced in Chapter 2.

As shown in Chapter 3 and mentioned by Liu et al. (2007), wireless signals are heavily influenced by obstacles and their environment. According to Wang et al. (2007) and Siddiqi et al. (2003), location accuracy can be drastically improved by additionally using map information and motion tracking. Thus, the main reason for choosing MCL is the fact, that Particle Filter is the only algorithm, which has the ability to combine the RSS-based measurements to the beacons with motion tracking by additionally using map information. Triangulation, the proximity method, and the scene analysis approach are not able to improve their location estimation by combining motion tracking with RSS-based measurements. Kalman Filter is the only algorithm which is also capable of using motion besides PF. But according to Wang et al. (2007), "map information is impossible to be integrated for tracking by EKF".

The second reason for choosing PF is its ability to solve the *global localization problem*. Thus, the algorithm can start determining the user's location without knowing the user's initial position. Furthermore, the algorithm is able to detect and to recover if the estimated location is completely wrong, e.g. due to short-term sensor failure.

The third reason is the algorithm's advantage of taking uncertainties into account. Besides PF, KF is the only mentioned algorithm which is also capable of modeling uncertainties. As mentioned before, PF is a non-parametric filter with the advantage of expressing a location estimation in the form of a multi-modal posterior belief. Whereas KF is a parametric filter which is fixed to normal distributed position estimations. Furthermore, the PF's multi-modal belief can be visually illustrated very well and the users can benefit from this as shown in Figure 2.2.

The fourth reason are the, in Chapter 1 defined, requirements. By using PF, based on RSS-based measurements, less pre-deployment effort is required. Furthermore, the required infrastructure is of little complexity compared to other solutions outlined in Chapter 2, which reduces the maintenance effort and the initial and maintenance costs.

Besides the above mentioned reasons, Particle Filter is an easy-to-implement algorithm. Furthermore, MCL is a well-known and well-studied approach for landmark-based localization in robotics, as mentioned by Thrun et al. (2005).

5.2 System Setup

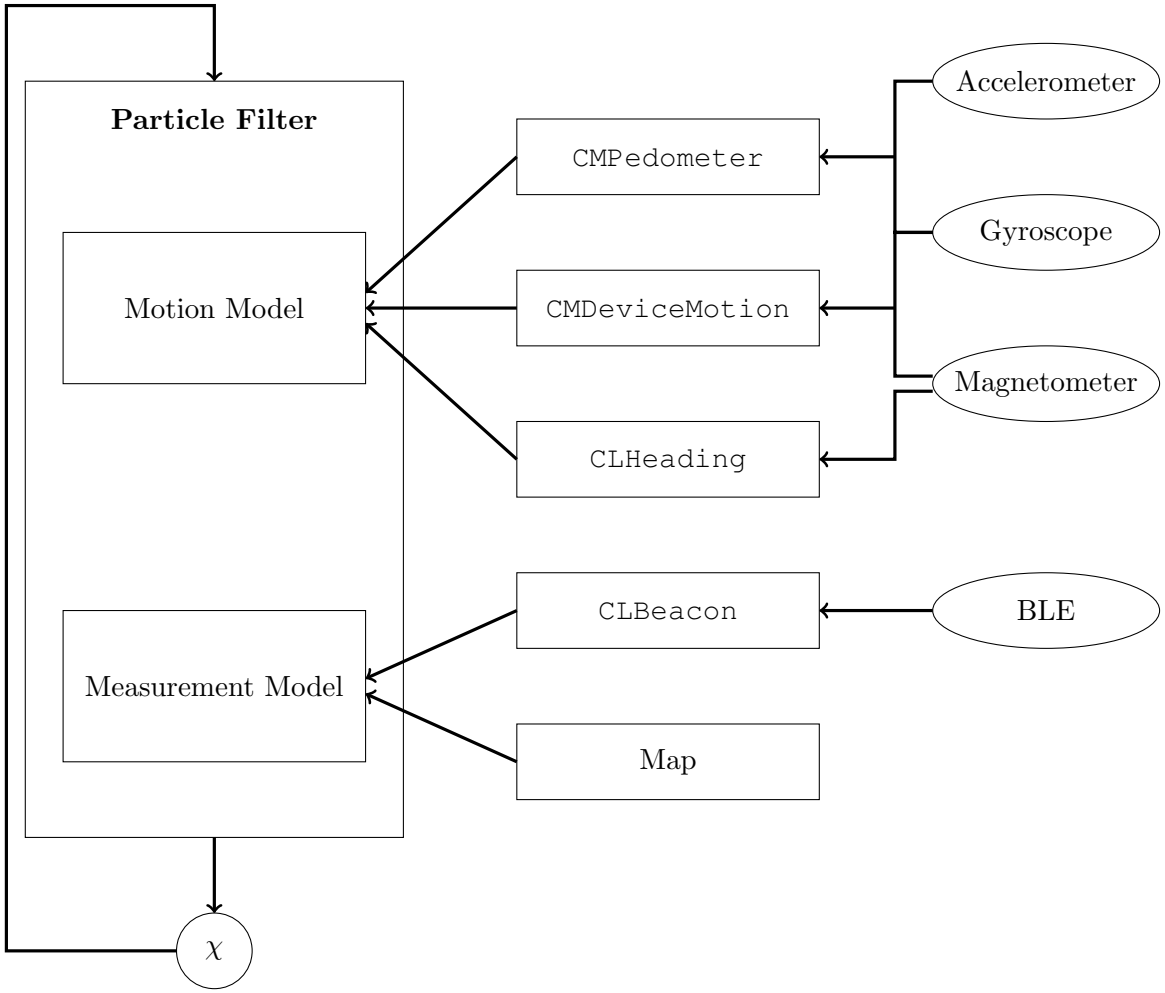


Figure 5.1: The solution's system setup including the used sensors and APIs.

Before going into the detailed description of the solution, this section provides an overview of the system setup.

The implementation is written in Apple's new programming language *Swift* (version 1.2), which is the successor of *Objective-C*. The language is based on modern programming concepts, such as functional programming, tuples, generics, etc. Furthermore, Apple tries to remove unsafe code, such as not initialized variables, null-pointers, array overflows, etc. by defining the language's elements in a very concise and expressive syntax, and by adding new language constructs and types, such as the Optional type. The language compiles to native code by using LLVM compiler. Swift code can also call C and Objective-C code, which provides the possibility to use Swift in existing projects (Apple Inc., 2014e).

The system, depicted in Figure 5.1, consists of the PF, its components, and the used APIs. Furthermore, the figure depicts the sensors used by the APIs. As explained in Section 2.2, the algorithm uses two models, the *Motion Model* and the *Measurement Model*.

The Motion Model, described in Section 5.3, is on one hand, responsible for tracking the

t	θ_C	θ_A	θ_M	$\theta_{C_{\text{current}}}$	$\theta_{C_{\text{last}}}$	$\theta_{A_{\text{current}}}$	$\theta_{A_{\text{last}}}$	θ
t_0	90.3		20.0	70.3				70.3
t_1		78.2	20.0		70.3	58.2		
t_2		79.0	20.0		70.3	59.0	58.2	
t_3	91.3		20.0	71.3	70.3	59.0	58.2	71.2
t_4		79.7	20.0		71.3	59.7	59.0	
t_5	92.3		20.0	72.3	71.3	59.7	59.0	72.1

Table 5.1: Example calculation of the internal heading θ according to the algorithm depicted in Listing 5.1.

user's motion by combining different sources. On the other hand, it samples from the motion model. To implement this, the Core Motion framework's pedometer and device motion component are used, which both use the smartphone's accelerometer, gyroscope and magnetometer, as described in Chapter 4. Furthermore, the component uses the Core Location framework's compass, i.e. `CLHeading`, which uses the magnetometer.

The Measurement Model, described in Section 5.4, implements the PF's importance factor calculation. For that it uses Core Location's `CLBeacon` component, which provides the RSS-based distances estimations to the beacons. It uses the smartphone's Bluetooth Low Energy (BLE) module, to receive the beacons' BLE signals. Furthermore, the Measurement Model uses the building's map.

The Particle Filter, described in Section 5.5, outputs the posterior, i.e. the position estimation represented by the particle set χ. Furthermore, χ is passed to the function by its next iterative call.

5.3 Motion Model

The here presented solution's motion model component is responsible for three tasks: Firstly, for tracking the users motion, secondly, to determine if the user is stationary, and thirdly, to allow the PF to sample from the motion model. The three tasks are prerequisites for the PF's implementation.

5.3.1 Motion Tracking

As mentioned in Chapter 4, CoreMotion provides a component called `CMPedometer`, which estimates the traveled distance based on the steps a user has taken. In addition, CoreLocation provides the smartphone's heading, based on the magnetic field, called `CLHeading`. Furthermore, CM's `CMDeviceMotion` provides the device `attitude`, which is determined by using sensor fusion which can also be used to calculate the device's heading. By combining these three sources the user's motion can be tracked and a path can be constructed.

Heading

As mentioned in Chapter 4, the heading provided by `CLHeading` can be influenced by magnetic fields other than the earth's magnetic field, which may cause wrong values. `CMDeviceMotion` uses sensor fusion instead of relying only on the magnetometer's values. Thus, it is not influenced by other magnetic fields. In contrast to the claims made by Apple's engineer

```
1  // instance variables
2  θ_A_current, θ_A_last = nil
3  θ_C_last = nil
4
5  // combined internal heading
6  θ = 0.0
7
8  // called by CoreMotion if new heading is available
9  didMeasureDeviceMotionHeading(θ_A_t) {
10   θ_A_current = θ_A_t − θ_M
11 }
12
13 // called by CoreLocation if new heading is available
14 didMeasureCompassHeading(θ_C_t) {
15   θ_C_current = θ_C_t − θ_M
16
17   if θ_A_latest ≠ nil && θ_A_last ≠ nil && θ_C_last ≠ nil {
18     Δθ_A = θ_A_current − θ_A_last
19     Δθ_C = θ_C_current − θ_C_last
20
21     θ_A_last = θ_A_current
22
23     θ = θ_t−1 + (Δθ_C+Δθ_A)/2
24   } else {
25     θ = θ_C_current
26   }
27
28   θ_C_last = θ_C_current
29 }
```

Listing 5.1: The algorithm used to calculate the heading θ by combining `CLHeading` θ_C and θ_A, which is the heading calculated from `CMDeviceMotion`'s `attitude`, relative to the maps orientation θ_M.

Pham (2012), the values tend to drift away. Consequently, relying only on `CMDeviceMotion` is not sufficient. As a result, both sources are combined to determine the smartphone's heading.

Both frameworks provide their values asynchronously. `CLLocationManager` calls its delegate method if a certain threshold of the heading's change, set to 1°, exceeds. Whereas CM calls its delegate periodically, every $\approx 0.02\,\text{s}$. The high update rate would not be necessary for heading, but as mentioned earlier, `CMDeviceMotion` also includes `userAccleration`, which is used for the stationary detection, as shown later.

Listing 5.1 illustrates the used algorithm to calculate the internally used heading. It combines the two heading sources for more robustness against influences of disturbing magnetic fields without the drift problem. Furthermore, Table 5.1 provides a calculation example for better understanding.

The `CMDeviceMotion` heading, further denoted as θ_A, measured at time t, is calculated from the rotation matrix, given by `CMDeviceMotion`'s `attitude`, as explained in Chapter 4. Due to its drift problem, only the change between two values, denoted as $\Delta\theta_A$, is used. For the absolute orientation, `CLHeading`'s `magneticHeading`, denoted as θ_C, is used. Internally the motion tracking uses the heading θ, which takes the map's orientation θ_M into account. θ is calculated if CL's heading filter is exceeded and thus, a new magnetic heading is provided. It is not updated if CM provides a new $theta_A$, due to its high update frequency, which would result in many useless updates of θ with very small change.

Motion Path Construction

As explained in Chapter 4, CM delivers every $\approx 2.5\,\text{s}$ a new `CMPedometer` object with the estimation of the user traveled distance. Besides the distance, the object contains the start date, which is the start date of the very first distance estimation. It is the same for all successive estimations. Additionally, it contains the end date. Thus, the user walked an (cumulative) estimated distance, beginning at start date and ending at end date. During the walk, the user's direction θ changes several times at a certain point in time.

A user's motion path is stored as an array of motions u. A motion consists of $u = (\theta, d, t_{\text{start}}, t_{\text{end}})^T$, the orientation θ at the motion's start date t_{start}, the motion's distance d in meters, and the motion's end date t_{end}. t_{start} and t_{end} are not really necessary for constructing the estimated motion path, but are necessary for the later explained motion integration.

To calculate the motions u of a walked distance d, the distance is split according to the timestamps corresponding to the smartphone's orientational changes, which occurred during the estimated distance since the latest stored motion. To do so, constant velocity over the distance d is assumed. Figure 5.2 illustrates an estimated path a user traveled in 2-dimensional space. The individual distances d_1 and d_2, estimated by CM are colored differently. By integrating the measured headings $\theta_0, \ldots, \theta_4$, the path is split into motion $u_0, \ldots, u_5$.

5.3.2 Stationary Detection

As mentioned in Chapter 4, the `CMPedometer` component requires at the beginning $\approx 6-8\,\text{steps}$ to deliver the first distance estimation. During a walk it continuously updates the estimation every $\approx 2.5\,\text{s}$. Due to the asynchronously incoming motion and beacon data, the filter cannot be run continuously with a fixed time interval, because the later discussed

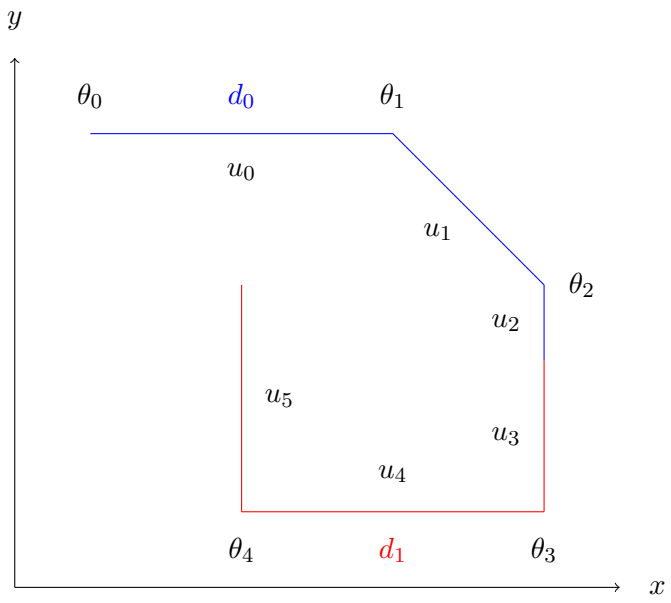


Figure 5.2: Illustration of the motion path construction by integrating heading $\theta_0, \ldots, \theta_4$ into the two estimated distances d_1, d_2. $u_0, \ldots, u_5$ are the resulting motions.

measurements need to be integrated at the right position on the user's walk. Thus, it is important to know whether or not the user is currently walking, i.e. is stationary or not.

Wang et al. (2007) propose a system based on acceleration data to detect a user's steps by detecting the acceleration's zero-crossing. However, the system does not actually need to count the user's steps, which would require too much effort to get this information. Also, Shanklin et al. (2011) use the user's acceleration to estimate the distance the user traveled. First they filter the values with a low-pass filter. Then they integrate the acceleration two times to get the distance. Thus, to detect if a user is stationary, one integration step would be sufficient to get the user's velocity. This requires a projection of the measured acceleration data into the global coordinate system as described by Shanklin et al. (2011). Unfortunately, their solution works very unreliable. According to Wang et al. (2007), integration of acceleration data for distance estimation works only in theory, but not reliably in indoor environments. Furthermore, their proposed projection is based on `CMDeviceMotion`'s `attitude` property, which suffers from the drift problem as shown in Section 4.3.

Shanklin et al. (2011) considers also another solution for step detection, which uses the user's minimum acceleration. Steps are detected by using a threshold, that needs to be exceeded by the acceleration's Euclidian-Norm $\|a\| = \sqrt[2]{x^2 + y^2 + z^2}$. Figure 5.3 depicts a user's 3-axis acceleration during a walk with two clearly visible stops. The corresponding Euclidian-Norm is shown in Figure 5.4. For the actual detection of the stationary and not stationary state, a simple moving average can be used. It is calculated over the last 50 values which correspond to 1 s using a update frequency of 0,02 s. If the user walks the simple moving average exceeds a threshold of $0.1\, ms^{-2}$.

The solution's advantage is, that it is quite easy to implement and does not need a projection of the user's acceleration data. As a result, the later discussed integration of measurements, i.e. the importance factoring can be delayed when the user starts to walk until CM provides a distance estimation.

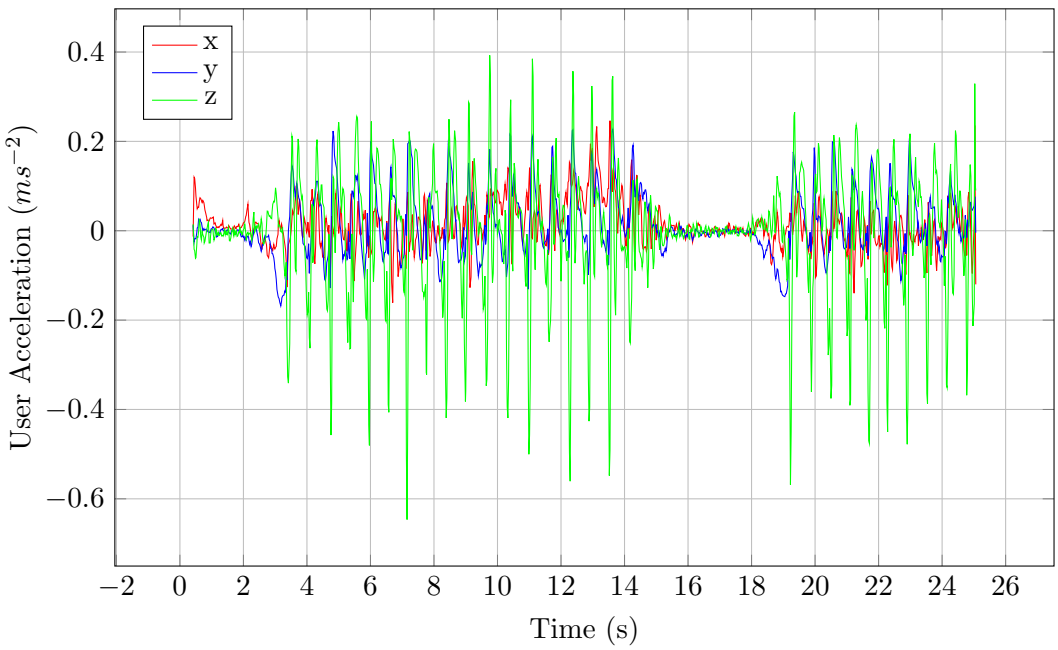


Figure 5.3: Example data of the 3-axis `userAcceleration` provided by CM's `CMDeviceMotion` object. The user's two stationary phases are clearly depicted by the low amplitude. Remark: `userAcceleration` is without gravity.

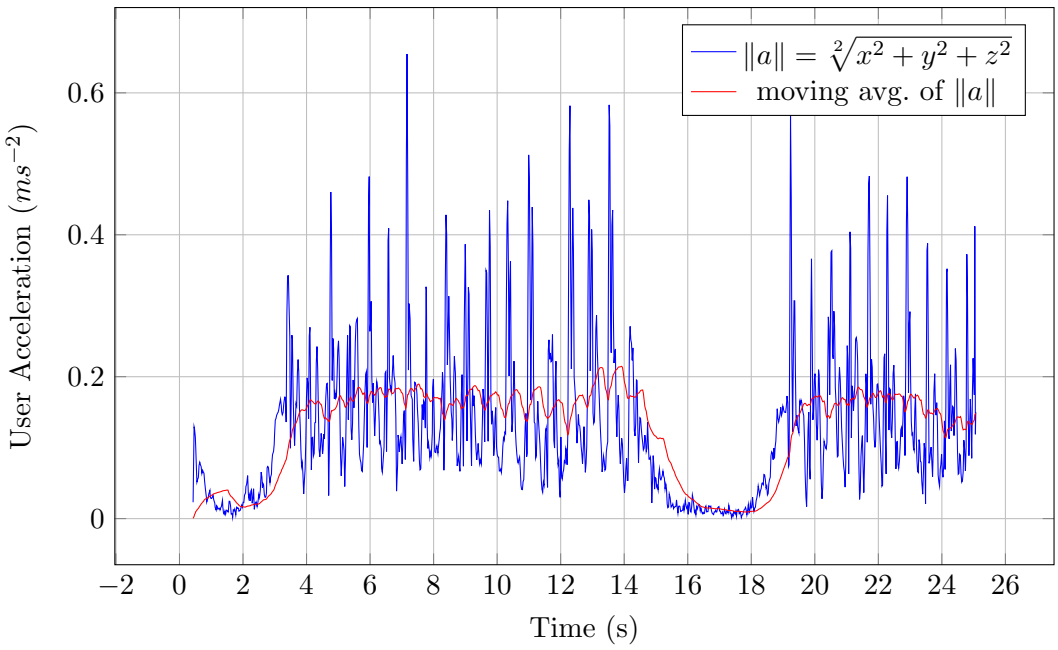


Figure 5.4: The euclidian norm of the 3-axis `userAcceleration` shown in Figure 5.3, and its simple moving average with a window of 1 s which corresponds to 50 measurements.

5.3.3 Sample Motion

As described in Section 2.2.2, MCL has a `sample_motion_model` function to sample from the motion model, i.e. to apply a motion u to a state hypothesis $x_{t-1}^{[m]}$ by taking the motion's uncertainties into account, as shown by Equation 5.1.

$$x_t^{[m]} = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \begin{pmatrix} x_{t-1} + \cos(\theta_u + \theta_{\text{noise}}) \cdot (d_u + d_{\text{noise}}) \\ y_{t-1} + \sin(\theta_u + \theta_{\text{noise}}) \cdot (d_u + d_{\text{noise}}) \\ \theta_u + \theta_{\text{noise}} \end{pmatrix} \tag{5.1}$$

The new state hypothesis is denoted as $x_t^{[m]}$. A state is defined as $x^{[m]} = (x, y, \theta)^T$, where x and y denote the position in 2-dimensional space and θ the user's orientation. d_u and θ_u are the distance and heading of motion u. The added noise d_{noise} and θ_{noise} are the translational and rotational uncertainties, modeled as Gaussians. First, the uncertainties, determined during the sensor evaluation shown in Section 4.3 were used, but as usual, they do not fit best. By trying different values the following better fitting values were derived.

$$d_{\text{noise}} = NDF(\mu_{\text{trans}}, \sigma_{\text{trans}}), \quad \mu_{\text{trans}} = 0, \quad \sigma_{\text{trans}} = \max(0.3, 0.3 \cdot d_u) \tag{5.2}$$

$$\theta_{\text{noise}} = NDF(\mu_{\text{rot}}, \sigma_{\text{rot}}), \quad \mu_{\text{rot}} = 0, \quad \sigma_{\text{rot}} = 20^\circ \tag{5.3}$$

d_{noise} depends on the motions distance d_u, shown in Equation 5.2, whereas θ_{noise} uses constant parameters, shown in Equation 5.3. Due to the lack of a built-in algorithm to sample from a Gaussian distribution the `sample_normal_distribution` algorithm proposed by Thrun et al. (2005, p. 124), is used.

5.4 Measurement Model

The measurement model component is responsible for the calculation of the importance factor $w^{[m]}$ for a given state hypothesis $x_t^{[m]}$ by taking the measurements z_t, its uncertainties, and the `map` into account. The `map` represents the environment, i.e. the building in form of a simple occupancy grid.

The algorithm depicted in Listing 5.2 illustrates the calculation of the importance factor `weight` of one state hypothesis. If the state is out of the map's boundaries or the position is not free, e.g. the map contains an obstacle at this position, the weight for the state is set to 0.0. If the states is valid, an importance factor w for each of the measurements, i.e. for each distance estimation between the smartphone and the received beacons, is calculated.

To calculate w, the euclidian distance between the state hypothesis x, i.e. the particle, and the beacon $z_{i_{\text{pos}}}$ need to be determined. It is the mean value μ_d required by the Probability Density Function (PDF). The PDF's standard deviation σ_d depends on the distance $z_{i_{\text{dist}}}$ between smartphone and the beacon. By trying different values, $\frac{1}{4}$ of $z_{i_{\text{dist}}}$ seems to fit best. The weight is then calculated by the PDF (Listing 5.2, Line 16). It is important to note, that only measurements with $z_{i_{\text{dist}}} < 5\,\text{m}$ are taken into account. Larger values are sorted out and not passed to the `measurement_model` function, because estimated distances to a beacon larger than 5 m are very unreliable, as shown in Section 3.4.

Finally, the weights w of each measurement are multiplied with each other, which is the importance factor `weight` of this state hypothesis x, i.e. particle. The reason for adding the

```
1  measurement_model(z = {z_0, z_1, z_{k-1}}, x, map) {
2    weight = 0.0
3
4    if position x on map && x is free {
5
6      for z_i in z {
7
8        z_{i_dist} = measured distance between phone and beacon
9        z_{i_pos} = beacons position on map
10
11       μ_d = euclidian distance between x and z_{i_pos}
12       σ_d = 0.25 · z_{i_dist}
13
14       w = PDF(z_{i_dist}, μ_d, σ_d)
15
16       weight = weight · w · 10.0
17     }
18   }
19   return weight
20 }
```

Listing 5.2: Algorithm to calculate the importance factor w of a state hypothesis x by taking the measurements z and the `map` into account.

additional factor of 10 for each subsequent weight w is that otherwise the `weight`'s order of magnitude depends on the count of beacons. Of course, this only works because usually $0.1 \leq \mathrm{w} < 1$. Table 5.2 illustrates the exponential decline of the `weight` by increasing beacon count k. For the illustration, all subsequent weights w of their importance factor `weight` have the same value $\mathrm{w} = 0.1$. If the additional factor of 10 is added the beacon count is irrelevant for the weight's magnitude. Of course, during one run of the Particle Filter the weight's magnitude is irrelevant, but to compare the sums of all importance factors over time, as used for the in Section 5.5.5 discussed recovery, this is very important.

k	w	`weight`
1	0.1	$0.1^1 = 0.1$
2	0.1	$0.1^2 = 0.01$
3	0.1	$0.1^3 = 0.001$
9	0.1	$0.1^9 = 1 \cdot 10^{-9}$

(a) Without additional factor

k	w	`weight`
1	0.1	$(0.1 \cdot 10)^1 = 1$
2	0.1	$(0.1 \cdot 10)^2 = 1$
3	0.1	$(0.1 \cdot 10)^3 = 1$
9	0.1	$(0.1 \cdot 10)^9 = 1$

(b) With additional factor

Table 5.2: Illustration of the importance factor's, i.e. the `weight`'s exponential decline by increasing beacon count k if no additional factor is added (Listing 5.2, Line 16).

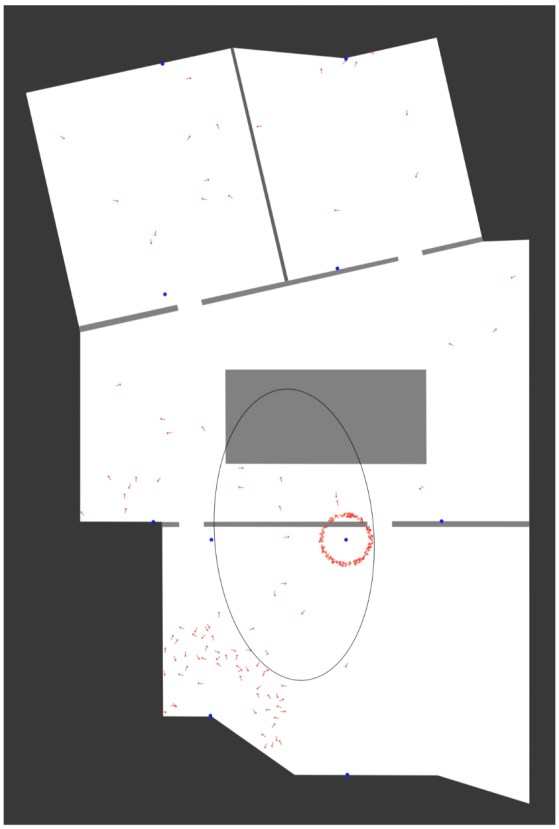


Figure 5.5: Example of the initial particle set. Particles are shown as red arrows, the beacons as blue filled circles. The large black circle is the 1σ-ellipse introduced in Section 5.5.6.

5.5 Particle Filter

The Particle Filter is the solution's heart, which is responsible for the position estimation by combining the before mentioned components. In this section first the *initial generation of the particle set* is explained. Then two functions, named `integrateMotions` and `filter`, which are often called by the actual PF implementation are introduced. Afterwards the PF's implementation is explained. In the end the *kidnapping problem's* solution, i.e. the recovery's implementation is presented.

5.5.1 Initial Particle Distribution

Before the PF can start to continuously run the initial posterior belief χ_0, i.e. the particle set, needs to be generated. Tests showed, that a static particle set size of 200 particles is sufficient. As mentioned in Chapter 3, each of the beacons can be uniquely identified by combining the beacon's three identifiers. In addition, their position on the map, i.e. the local coordinate system, is known. Consequently, this information can be used to specifically distribute the initial particles around the received beacons. This is a huge advantage instead of uniformly distributing them on the map's free space. For that reason, the implementation starts searching for beacons and then waits until CL reports the first ranged beacons. Then, the particles are distributed in a circle around the beacons with the radius of the estimated distance to that

```
1  integrateMotion(χ_{t-1}, u_t) {
2
3    χ̄_t = ∅
4
5    for m = 1 to M {
6      x_t^[m] = sample_motion_model(u_t, x_{t-1}^[m])
7    }
8    return χ̄_t
9  }
```

Listing 5.3: Helper function to apply the motion u_t to the particle set χ_{t-1} by sampling from the motion model.

beacon. To take the distance estimation's uncertainty into account each distance of a particle to its beacon gets added some Gaussian noise with $\mu = 0$ and $\sigma = 0.2 \cdot \mathrm{d}$, which depends on the estimated distance d. A particle's orientation θ is a uniformly distributed random value within $[0, 2\pi)$.

Additionally, the amount of particles distributed around a beacon depends on the estimated proximity. 50 % of the particle set size are distributed around the particular beacon, which is most probably closest to the smartphone according to the beacons proximity, i.e. the estimated distance between smartphone and beacon. 25 % are distributed around the second closest beacon, 12.5% around the third closest beacon, and so on. The beacon which's proximity is furthest from the smartphone, gets the remaining particles.

Figure 5.5 shows a screenshot of the implemented iOS app depicting the initial particle set. The particles are depicted as small red arrows, the beacons as blue filled circles. The black large circle shows the 1σ-ellipse, which is explained in Section 5.5.6.

5.5.2 Motion Integration

The `integrateMotion` function, shown in Listing 5.3, is a helper function to reduce the complexity of the later introduced algorithm. The function's arguments are the current particle set χ_{t-1} and the motion u_t which it applies to each particle. Thus, it samples from the motion model as discussed in Section 5.3.3. In the end it returns the new particle set χ_t.

5.5.3 Filtering

The `filter` function is another helper function, which is responsible for the *importance factoring* and the *resampling*, depicted in Listing 5.4. The function's arguments are the particle set χ_{t-1}, the measurements z_t, and the `map`. It determines the importance factor of each particle as explained in Section 5.4. Afterwards, the resampling takes place to transform the old particle set into the new particle set as described in Section 2.2. The solution uses *roulette wheel resampling*, which is a common resampling method based on *independent sampling* proposed by Thrun et al. (2005, p. 108–111).

The shown implementation is the basic implementation which does not solve the kidnapping problem. To better understand the implementation, the PF's complexity was reduced

```
1  filter(χ_{t-1}, z_t, map) {
2
3    χ̄_t = ∅
4
5    // importance factoring
6    for m = 1 to M {
7      w_t^[m] = measurement_model(z_t, x_t^[m], map)
8      add ⟨x_t^[m], w_t^[m]⟩ to χ̄_t
9    }
10
11   χ_t = ∅
12
13   \\ resampling
14   while size of χ_t less M {
15     draw i with probability ∝ w_t^[m]
16     add x_t^[i] to χ_t
17   }
18
19   return χ_t
20 }
```

Listing 5.4: Helper function for the PF's importance factoring and resampling.

by first showing a simplified filter function, which neglects the recovery. Later in the chapter the shown `filter` function is enhanced to explain the implemented recovery feature.

5.5.4 Particle Filter

After explaining all prerequisites this section provides a detailed insight into the solution's Particle Filter implementation. To reduce complexity the PF's code is divided into three logical parts, shown in Listing 5.6, 5.7 and 5.8. Listing 5.5 indicates their call order. It also contains several instance variables, which are accessible from the three parts.

The `particleFilter` function is continuously called every $\approx 1\,\mathrm{s}$ when CL updates its measurements. To remember, CL calls its delegate to update the ranged beacons whether it received a beacon's signal or not. Consequently, when the function gets called, the latest measurements are always available and ready for processing. The function takes the particle set χ_{t-1} as argument and returns the new particle set χ_t.

The implementation uses a static particle set size of 200 particles. Tests showed, that 200 particles are sufficient and increasing the particle set does not really improve the estimation. The `particleFilter` function requires for one execution on average $\approx 10\,\mathrm{ms}$. Thus, a dynamic particle set size to safe processing power is also not required.

Instance Variables

As mentioned, the motions and measurements are delivered asynchronously by their frameworks. The motion model's implementation already reduces this problem by combining the

```
1  // instance variables
2  u_buffer // list of not yet integrated motions u_t
3  u_latest // stores the latest motion
4  z_buffer // list of not yet integrated measurements z_t^[k]
5  map // occupancy grid and landmarks
6
7  particleFilter(χ_{t-1}) {
8
9    χ̄_t = χ_{t-1}
10
11   χ̄_t = PART_1(χ̄_t)
12
13   χ̄_t = PART_2(χ̄_t)
14
15   χ̄_t = PART_3(χ̄_t)
16
17   return χ_t
18 }
```

Listing 5.5: Shows the Particle Filter's implementation and the used instance variables. To reduce complexity the code is divided into three parts, shown in Listing 5.6, 5.7 and 5.8.

two headings and the estimated walked distance into one motion. Thus, the PF's implementation has just to deal with two asynchronous sources. To overcome the asynchrony, the motions u and measurements z are buffered in two ascending ordered lists, the `u_buffer` and `z_buffer`, according to their timestamps. The stored motions and measurements are provided by the motion model and the measurement model, and defined as mentioned in Section 5.3 and Section 5.4. The instance variable u_{latest} stores, as the name already implies, the latest motion that was calculated by the motion model. The `map` variable stores the environment's map in the form of an occupancy grid and the beacons' positions together with their unique identifiers.

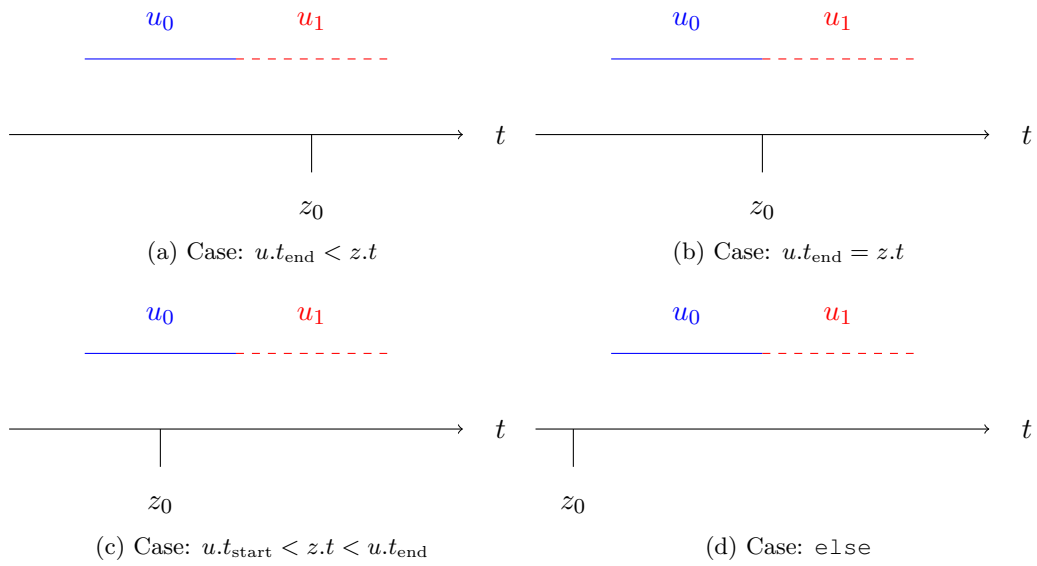
Particle Filter Function — Part 1

The algorithm's first part, shown in Listing 5.6, consists of basically one loop, which tries to integrate the buffered motions and filter with the buffered measurements at the right point in time. The loop is executed as long as both buffers are not empty. The four if-cases are visualized in Figure 5.6. In each example, the `u_buffer` contains two motions u_0, u_1, and the `z_buffer` contains one measurement z_0. Each subfigure illustrates only the loop's first iteration. Thus, u_1 is not relevant during this iteration.

- Case 1 (Figure 5.6a): If a motion ends before the measurement was taken ($u.t_{\text{end}} < z.t$) the motion is integrated and the measurement is reserved for the next iteration.
- Case 2 (Figure 5.6b): If a measurement is taken exactly at the end of a motion ($u.t_{\text{end}} = z.t$) the motion is integrated. Afterwards, the set is filtered with the measurement.

```
1  PART_1(χ_{t-1}) {
2
3    χ̄_t = χ_{t-1}
4
5    while u_buffer isNotEmpty && z_buffer isNotEmpty {
6      u = u_buffer.first
7      z = z_buffer.first
8      if u.t_end < z.t {
9        χ̄_t = integrateMotion(χ̄_t, u)
10       u_buffer.remove(u)
11     } else if u.t_end = z.t {
12       χ̄_t = integrateMotion(χ̄_t, u)
13       χ̄_t = filter(χ̄_t, z, map)
14       u_buffer.remove(u)
15       z_buffer.remove(z)
16     } else if u.t_start < z.t && z.t < u.t_end {
17       split u at timestamp z.t in u_1 and u_2
18       χ̄_t = integrateMotion(χ̄_t, u_1)
19       χ̄_t = filter(χ̄_t, z, map)
20       u_buffer.remove(u)
21       insert u_2 at beginning of u_buffer
22       z_buffer.remove(z)
23     } else {
24       χ̄_t = filter(χ̄_t, z, map)
25       z_buffer.remove(z)
26     }
27   }
28
29   return χ̄_t
30 }
```

Listing 5.6: Part 1 of the solution's `particleFilter` function shown in Listing 5.5.



(a) Case: $u.t_{\text{end}} < z.t$ (b) Case: $u.t_{\text{end}} = z.t$

(c) Case: $u.t_{\text{start}} < z.t < u.t_{\text{end}}$ (d) Case: `else`

Figure 5.6: Illustration of the four if-cases shown in Listing 5.6.

- Case 3 (Figure 5.6c): If a measurement is taken during a motion ($u.t_{\text{start}} < z.t < u.t_{\text{end}}$) the motion needs to be divided. The first sub motion is then integrated into the particle set. Afterwards it is filtered with the measurement. The second sub motion is stored in the buffer at the first place to be processed during the loop's next iteration.
- Case 4 (Figure 5.6d): If the measurement was taken before the motion ($u.z.t < t_{\text{start}}$ / `else`) the filter is executed without integrating the motion. The motion is kept back for the next iteration.

If one of the buffers is empty the algorithm continues with the function's second part.

Particle Filter Function — Part 2

If, after executing the first part, motions are still buffered, these motions are integrated in this part shown in Listing 5.7. The reason for the remaining motions is that the `particleFilter` function is also called if no measurements are received as mentioned at the beginning. The reason why they can be integrated without current measurements is that they are definitely older than the not yet occurred measurements and thus there is no reason to keep them back.

After integrating a motion the filter method is called. The filtering is executed without an actual measurement. Thus, only the map's occupancy grid is used to calculate the particle's importance factors. Consequently, particles that are moved onto an occupied position are sorted out.

The purpose of not waiting for the next measurement to integrate the remaining motions is on one hand, that if the user is out of the beacons range, the motion is still integrated. Thus, the particles on the view are still moving according to the users motion. On the other hand the algorithm's performance does not get worse, because `u_buffer` does not collect motions of several time steps that need to be integrated at once.

```
1  PART_2(χ_{t-1}) {
2
3    χ̄_t = χ_{t-1}
4
5    // integrate remaining motions and filter with map
6    while u_buffer isNotEmpty {
7      u = u_buffer.first
8      z = ∅
9      χ̄_t = integrateMotion(χ̄_t, u)
10      u_buffer.remove(u)
11
12      // empty measurements
13      χ̄_t = filter(χ̄_t, z, map)
14   }
15
16   return χ̄_t
17 }
```

Listing 5.7: Part 2 of the solution's `particleFilter` function shown in Listing 5.5.

Particle Filter Function — Part 3

The first two parts are important for integrating and filtering while a person is walking. But integrating measurements if a person is stationary can drastically improve the estimated location, as shown in Chapter 6. Thus, the received measurements are directly integrated instead of buffered, as shown in Listing 5.8. Furthermore, by applying the measurements directly the user sees without a long delay how the estimated position changes.

To be able to integrate them directly the algorithm needs to be sure that the user is stationary. As mentioned in Chapter 4, CL has the problem of requiring 6 – 8 steps to recognize that a person is walking, i.e. until it delivers this information. Consequently, if a user is stationary and starts to walk, the measurements collected during the first steps need to be buffered until the motion data is available, otherwise the measurements would be integrated at the wrong position. The second important information is to know, that a person is now stationary and no further motions are being delivered. Only then the measurements can be directly integrated without delay.

The latter problem can be solved easily. Motions are delivered every $\approx 2.5\,\text{s}$. Thus, if the difference between now and the latest motion u_{latest} is at least 2.7 s, CM does not deliver any new motion until the person starts to walk again. 2.7 s was chosen, because the 2.5 s are an approximate value. Besides that, the `particleFilter` function is only invoked every $\approx 1\,\text{s}$. Thus, no additional delay is created.

To be able to buffer the measurements during the first steps when the user starts to walk again, the stationary detection presented in Section 5.3.2 is used. Due to the solution's simple moving average over the last $\approx 1\,\text{s}$ it detects within $\approx 1\,\text{s}$ if a user is stationary or not. Thus, if the user is not stationary, the measurement integration is not executed and the values are kept in the buffer.

Furthermore, tests showed, that additionally integrating a small Gaussian distributed

```
1  PART_3(χ_{t-1}) {
2
3    χ̄_t = χ_{t-1}
4
5    Δt = currentDate - u_latest.t_end // in seconds
6
7    if motionModel.deviceIsStationary && Δt ≤ 2.7 {
8      u = motion with distance 0.0
9      z = z_buffer.first
10
11     χ̄_t = integrateMotion(χ̄_t, u)
12     χ̄_t = filter(χ̄_t, z, map)
13     z_buffer.remove(z)
14   }
15
16   return χ̄_t
17 }
```

Listing 5.8: Part 3 of the solution's `particleFilter` function shown in Listing 5.5.

random motion with $\sigma = 0.3\,\text{m}$ helps to faster improve the location accuracy. If a person is stationary, the person's orientation does not matter. Consequently, also the motion's orientation θ is set to an uniformly distributed random value between $[0,\ 2\pi)$. Using a random orientation has the advantage that the particles can move in any direction.

5.5.5 Recovery

The implemented recovery feature has two purposes. First, it can happen that all particles are moving out of bounds, and thus are filtered out. Second, if the particles are on a totally wrong position, which can happen if the user walked too softly. Thus the pedometer could not recognize the user's steps. It can also happen that the orientation does not fit the actual one. For instance, the user's phone does not points straight forward during the walk.

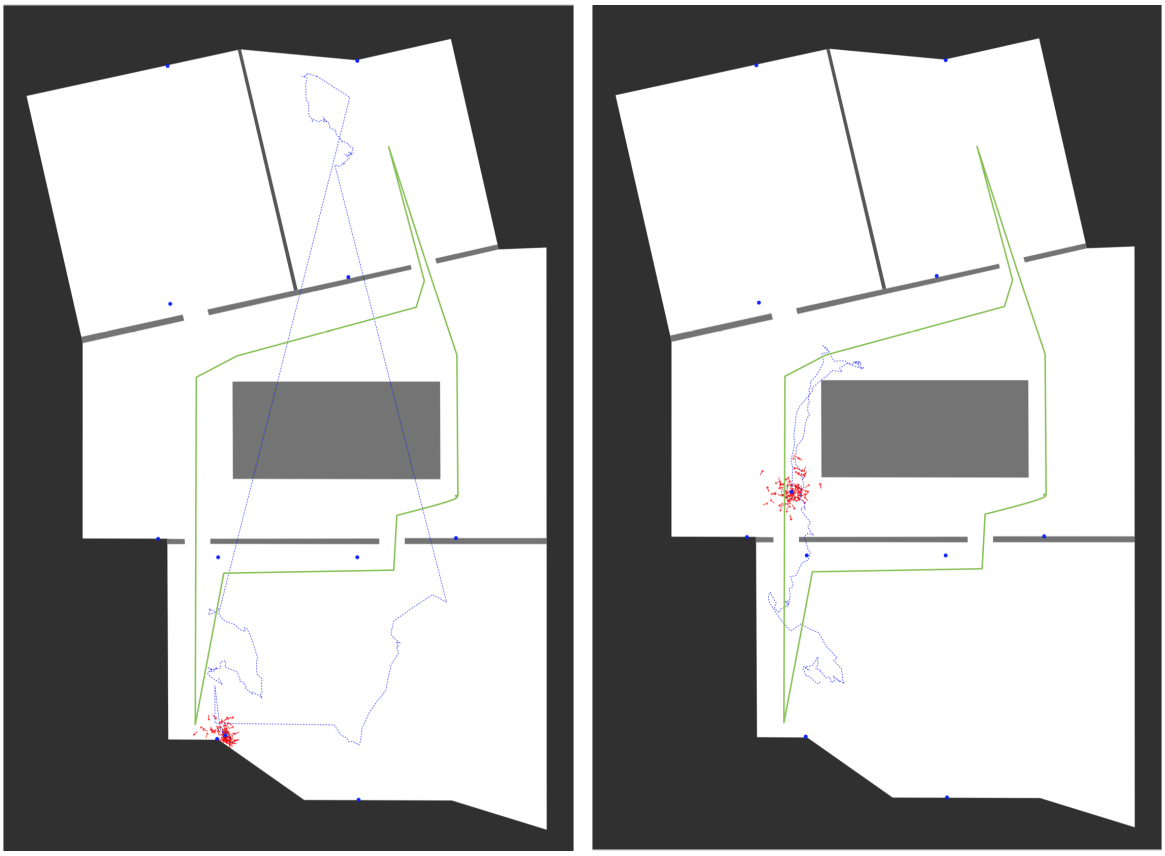
To determine both situations, the before introduced `filter` function, shown in Listing 5.4, is enhanced to support recovery. The enhanced version is shown in Listing 5.9. During the resampling, the new particle set's `weightSum` $= \sum_{m=0}^{M-1} w_t^{[m]}$ is calculated, which is the sum of all drawn particles during the resampling.

If the `weightSum` $== 0$ χ_t is empty and thus all particles have moved out of bounds, or their location is already occupied, and thus they are ignored. Consequently, a new random particle set is generated. The particles are distributed as described in Section 5.5.1, i.e the position estimation is restarted.

If the `weightSum` > 0 it is normalized by the particle set size. Then, it is stored in the `particleSums` array, which contains the latest three weightSums (Listing 5.9, Line 24). When the `filter` function is invoked the next time it first checks if the array already contains three weight sums, and if true the average of the three weight sums is calculated (Listing 5.9, Line 8). If the average drops below a certain threshold it is very likely that the posterior's distribution does not match the user's true position, i.e. it is totally wrong. To recover from

```
1   // instance variable
2   weightSums // array that stores the last 3 weight sums
3
4   filter(χ_{t-1}, z_t, map) {
5
6     χ̄_t = ∅
7
8     if size of weightSum == 3 && avg.\ weightSums < 1.0 {
9       add 20% additional random particles to χ_{t-1}
10    }
11
12    // importance factoring
13    ...
14
15    weightSum = 0.0
16
17    \\ resampling
18    while size of χ_t less M {
19      ...
20      weightSum += w_t^[m]
21    }
22
23    if weightSum > 0 {
24      override oldest value in weightSums with (weightSums/
          particleSetSize)
25    } else {
26      χ_t = generate new random particle set
27      clear weightSums
28    }
29    return χ_t
30  }
```

Listing 5.9: The solution's enhanced filter function including recovery based on Listing 5.4.



(a) With Recovery (b) Without Recovery

Figure 5.7: Example localization track with and without the recovery feature. The green line approximates the actual path. The blue dashed line shows the estimated path. Particles are depicted as red arrows, beacons as blue filled circles.

this state, 20 % additional random particles were added to the particle set. The random particles are distributed as described in Section 5.5.1. Due to the fact that the particles are added before the importance factoring and resampling, the added particles help to recover from the state by getting higher importance factors and thus they are drawn with a high probability.

Test of different random particle counts showed that 20 % seems to achieve the best result. Adding too less particles does not really help to recover, but adding too much results in a *jumping* position estimation. The reason for taking the average over the last three weight sums is that sometimes the weight sum drastically drops during one filter call. This happens for instance, if the measurements to the beacons were heavily influenced by an obstacle and thus are totally wrong.

Figure 5.7 illustrates the recovery's benefit. The weight sums shown in Figure 5.8 correspond to the two screenshots. During the experiment, the kidnapping problem was simulated by walking very smoothly approximately on the green path from the lower lecture hall through the building foyer by passing the large obstacle at the left side into the upper right lecture hall. After being stationary for a few seconds, the test person returned by passing the obstacle

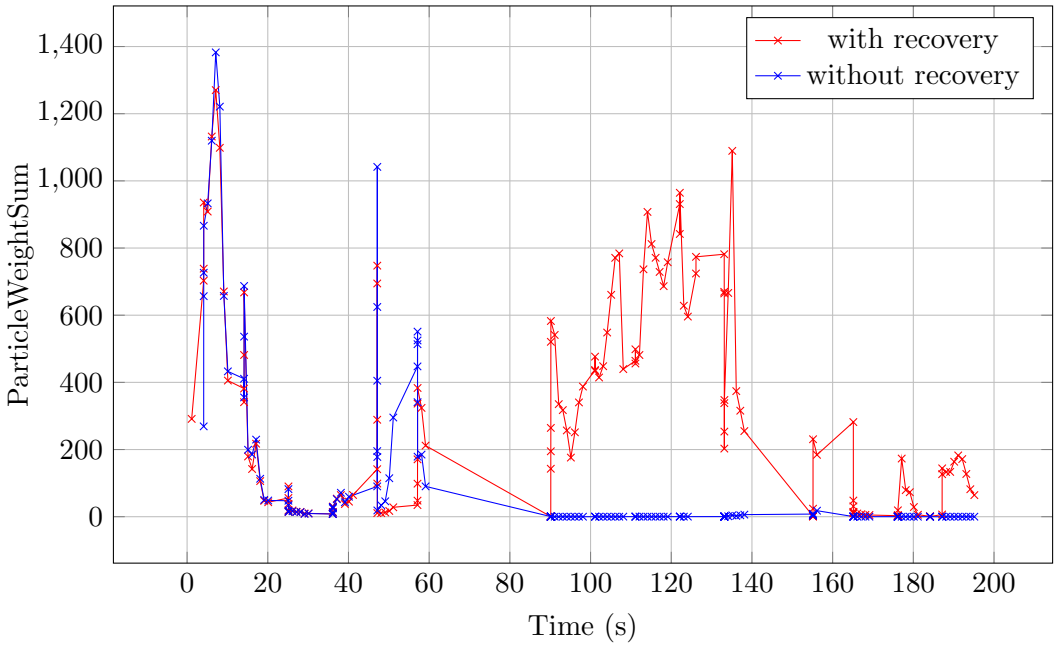


Figure 5.8: The particle weight sum over time which belongs to the example localization track shown in Figure 5.7.

on the other side. During the walk, the iPhone's pedometer was not able to recognize any step. Both experiments, the one with recovery, shown in Figure 5.7a, and the one without recovery, shown in Figure 5.7b, are simulated by using the exact same recorded data as input.

By testing different thresholds in different scenarios, a threshold of 1.0 was chosen which seems to fit best. In this example, recovery was executed the first time after $\approx 90\,\text{s}$. During the long break from $60-90\,\text{s}$ the test user walked through the foyer where no beacon signals with an estimated distance of less than $5\,\text{m}$ were received. Thus, the `filter` function was not executed. After receiving measurements again the average particle weight sum dropped below the threshold and recovery is executed. Thus, the particles jump from the lower to the upper lecture hall. The same happened at $\approx 155\,\text{s}$ on the way back. The third recovery is executed at $\approx 184\,\text{s}$, which caused the small jump next to the end position.

Without recovery, shown in Figure 5.7b, the algorithm is not able to approximate the true location. After $\approx 90\,\text{s}$ the weight sum drops to ≈ 0, and stays there.

5.5.6 Location Estimation

To be able to display a concrete location estimation instead of a particle cloud, the PF's posterior belief, which is a multi-modal distribution represented by the particle set χ_t, needs to be converted into another distribution. As mentioned in Chapter 2, uncertainties are typically modeled as (multivariate) Gaussian distributions. Consequently, it is appropriate to transform the PF's posterior into a two-dimensional Gaussian distribution $N(\mu, \Sigma)$, where the mean $\mu = (x, y)^T$ represents the concrete location. The location's uncertainty is expressed as the covariance matrix $\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_y^2 \end{pmatrix}$. The mean μ can either be χ's arithmetic mean or its weighted mean by using the particles' importance factors as weights. Based on the estimated

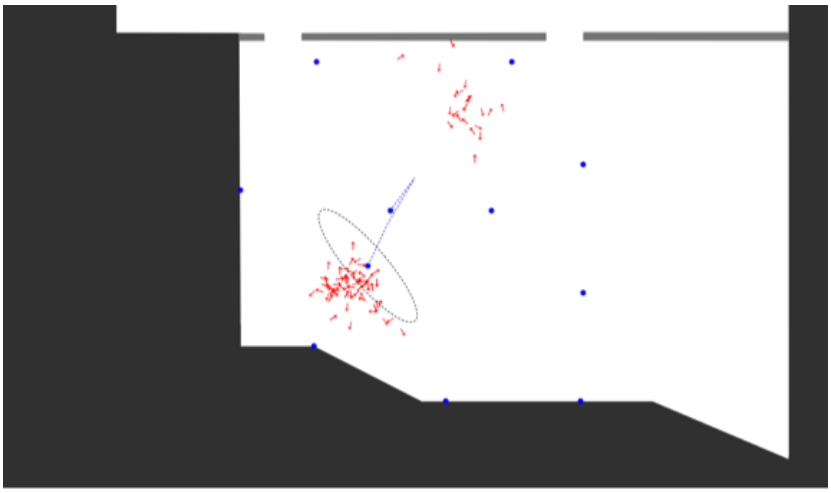


Figure 5.9: Illustration of the particle set's transformation into a Gaussian distribution. The 1σ-ellipse with weighted mean is shown as black dashed ellipse. The deployed beacons are depicted as blue filled circles. The distributed particles as red arrows. The blue dashed line is the estimated path.

mean, the covariance Σ can be determined. The Gaussian distribution's parameters can then be used to visualize the user's location on the map in form of a 1σ-ellipse. Thus, the user's true location is with a probability of $\approx 39\,\%$ somewhere in the 1σ-ellipse. Figure 5.9 illustrates the transformation of the particle set χ into a 1σ-ellipse depicted as black ellipse.

Chapter 6

Evaluation

After explaining the solution's implementation this chapter presents the evaluation's results The evaluation is based on the metrics introduced in Section 2.1.

6.1 Accuracy and Precision

To determine the solution's accuracy and its precision different tests in a real world experiment were performed. For this purpose, the 10 beacons, bought from BEACONinside, were deployed in different parts of the university building. The tests were done in a static environment; thus, the test person was the only dynamic object in the experiment. Besides the deployed beacons, the building is equipped with several WiFi access points, which use the 2.4GHz baseband, too. Thus, their signals may influence the bacons' signals. During the experiments a static particle set size of 200 particles was used, as mentioned in Chapter 5.

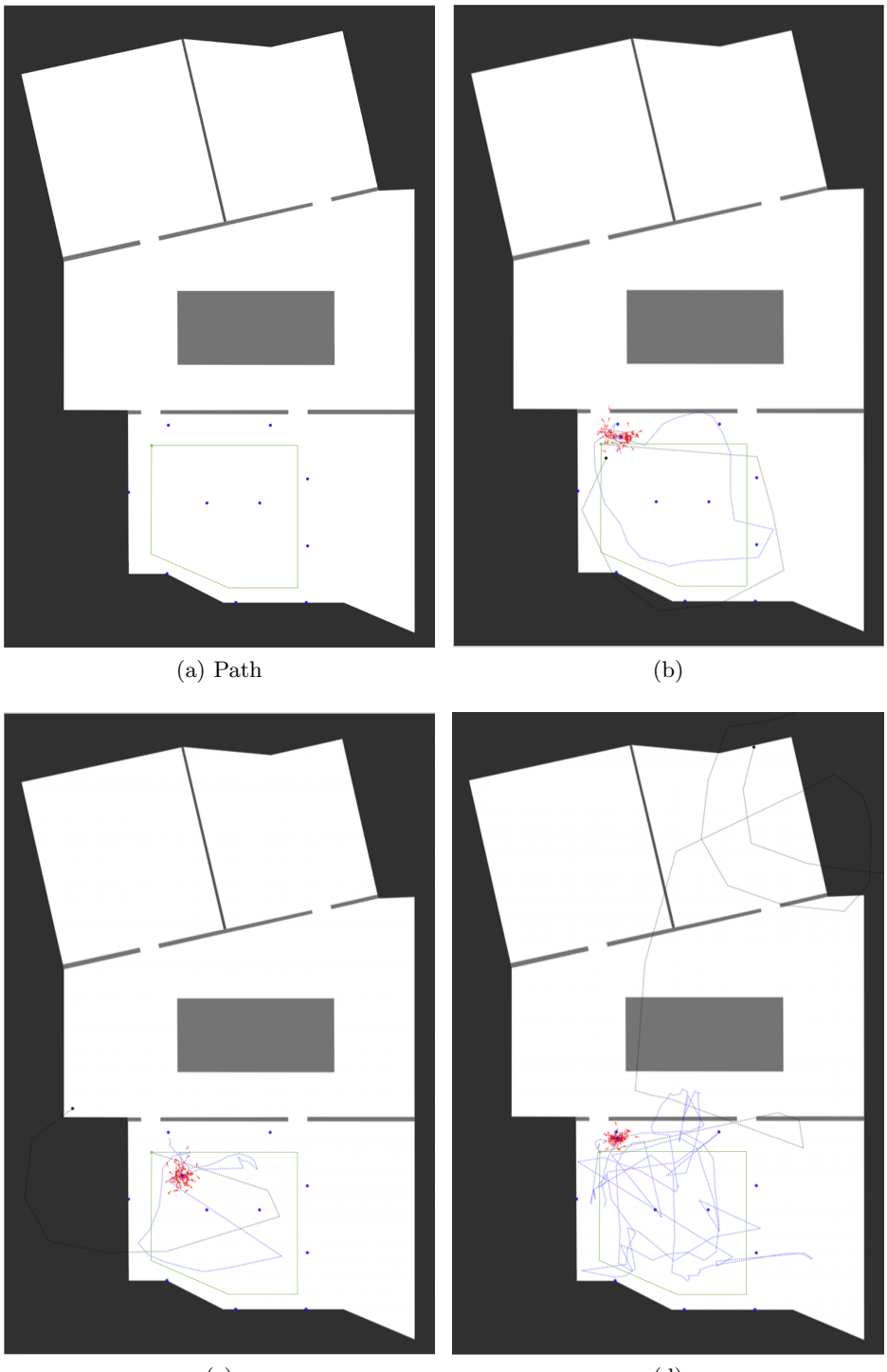
Experiment 1

For the first experiment only one lecture hall was equipped with all 10 beacons. Figure 6.1 shows a picture of the $\approx 15\,\text{m}$ long and $\approx 8-11\,\text{m}$ wide lecture hall for illustration purposes. On the following screenshots the lecture hall is depicted on the map's lower part. The picture was taken from the hall's lower left corner.

The purpose of this experiment is to test how good the localization works inside the shown lecture hall. Figure 6.2a depicts the beacons' positions as blue circles.

To determine the algorithms accuracy, the test person walked on a specified path, depicted in green, beginning and ending at the upper left corner. At the end of each walk the application's estimated end position is compared with the true end position by calculating the error in form of the euclidian distance. As mentioned in Section 5.5.6, the location can be estimated by either calculating the mean of χ, or by calculating its weighted mean. To be able to compare them the error of both approaches is calculated.

Figure 6.1: The lecture hall depicted on the map's lower part.



(a) Path (b)

(c) (d)

Figure 6.2: The screenshots illustrate example tracks of experiment 1. The green line shows the actual path. The blue filled circles depict the beacons. The blue dashed line shows the estimated path. The particles are depicted as red arrows.

Scenario	μ	σ	μ_w	σ_w
1	2.23	2.74	2.08	2.55
2	2.16	2.53	2.16	2.53
3	3.00	3.64	3.55	4.23
1–3	2.47	2.83	2.60	3.01

(a) Error at stop

Scenario	μ	σ	μ_w	σ_w
1	2.19	3.13	2.17	3.13
2	1.94	2.52	1.95	2.53
3	2.73	3.71	2.74	3.72
1–3	2.29	2.97	2.29	2.97

(b) Error after being $\approx 5\,\mathrm{s}$ stationary

Scenario	μ	σ
1	3.63	4.24
2	7.11	8.05
3	10.81	13.21
1–3	7.19	13.21

(c) Motion Tracking Error

Table 6.1: Results of experiment 1. μ is the mean error and σ the error's standard deviation by using the particle set's mean for the location estimation. μ_w and σ_w are determined by using the particle set's weighted mean.

Additionally, the error of the motion tracking's end position is compared with the true position. To be able to compare the motion error with the estimated location error by taking into account the motion drift, the experiment was divided into three scenarios. First, the test person walked just one round on the green path. Second, two rounds, and third three rounds. For each scenario the test user walks three times in clockwise and three times in counter-clockwise direction on the path.

Furthermore, this experiment is used to determine if the accuracy improves after being stationary at the end position for a short time. Thus, two location errors are determined. The error of the estimated position directly after integrating the last reported motion and the location error $\approx 5\,\mathrm{s}$ later.

Results

The results are shown in Table 6.1 and are visualized in Figure 6.3.

Table 6.1a depicts the average distance error and the error's distribution, for each scenario, directly after integrating the last motion. Table 6.1b shows the determined values recorded $\approx 5\,\mathrm{s}$ after the last motion integration. The calculated average error using χ's mean as location is denoted with μ. The errors standard deviation is denoted with σ. μ_w and σ_w are the average error and its standard deviation using the weighted mean of χ for the position estimation. Table 6.1c shows the motion tracking's error μ and the error's distribution in form of its standard deviation σ.

The results show that during the experiment an accuracy of $2.29\,\mathrm{m} - 2.6\,\mathrm{m}$ is achieved. It also shows that the location estimation tends to improve after being stationary at the end position, as expected.

The difference between the two position estimation approaches is very small. The result sways, sometimes χ's mean, and sometimes its weighted mean is more accurate. Supposedly the minimal difference is based on the particle cloud's high concentration.

The result's visualization also clearly shows that the motion error grows with increasing

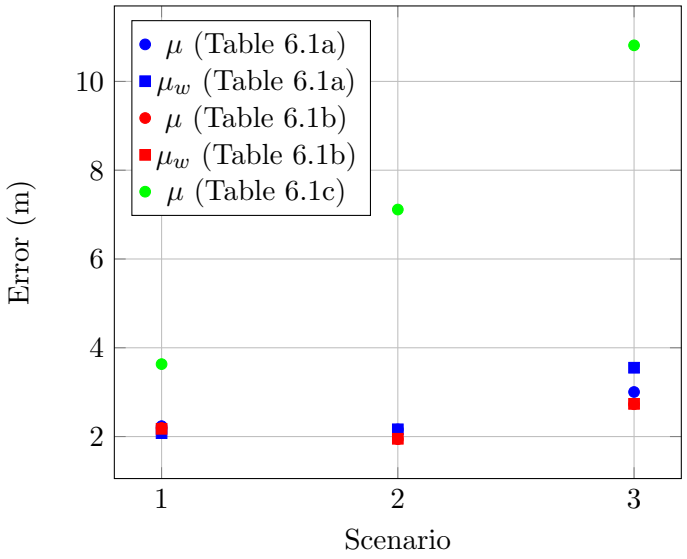


Figure 6.3: Visualization of the results of experiment 1 shown in Table 6.1.

number of rounds, whereas the estimated position error only grows minimally. Supposedly, the minimally higher position error is the result of a few outliers in the motion error, which of course affects the position estimation. Besides the path, Figure 6.2 depicts three examples of the experiment. They clearly show, that the motions' accuracy influences the position estimation. If the motion is completely wrong (Figure 6.2d), the in Section 5.3.2 introduced recovery feature often needs to intervene, which causes several jumps in the position estimation. Whereas Figure 6.2c shows, that if the motion is acceptably wrong, the algorithm can correct the path (of course not perfectly). If the motion estimation fits the true path the estimated path depends just on the beacons' signal quality, as shown in Figure 6.2b.

Experiment 2

The purpose of the second experiment is on one hand, to double check the implementation's accuracy by using another position, and on the other hand, to demonstrate the dependency of the estimation's accuracy and the amount of deployed beacons.

This time the test person walked on a random path. The person started at the orange circle, shown in Figure 6.5 on the upper left corner, and stopped at the green circle, approximately in the lecture hall's center.

To demonstrate the impact of reducing the number of deployed beacons, five test walks with all 10 beacons deployed were recorded first. Then the simulation was used to play back the recorded values by reducing the input to a lower number of deployed beacons. Thus, the values are directly comparable, because for all test cases the exact same input data is used. Also for this experiment the location error is calculated directly after integrating the last motion and $\approx 5\,\mathrm{s}$ later.

Results

The experiment's results are shown in Table 6.2 and visualized in Figure 6.4. The corresponding beacon deployment is shown in Figure 6.5.

Beacons	μ	σ	μ_w	σ_w
10	3.38	4.04	3.33	3.99
8	3.18	3.94	3.15	3.91
6	4.39	5.72	4.37	5.70
4	4.48	5.28	4.50	5.30
2	3.68	4.63	3.69	4.66

(a) Error at stop

Beacons	μ	σ	μ_w	σ_w
10	1.52	1.80	1.52	1.79
8	1.67	2.07	1.67	2.06
6	2.87	3.96	2.89	3.96
4	3.23	3.88	3.23	3.88
2	2.95	3.60	2.98	3.63

(b) Error after ≈ 5 *sec*

Beacons	μ	σ
2–10	5.68	7.04

(c) Error of motion tracking

Table 6.2: Results of experiment 2. μ is the mean error and σ the error's standard deviation by using the particle set's mean for the location estimation. μ_w and σ_w are determined by using the particle set's weighted mean.

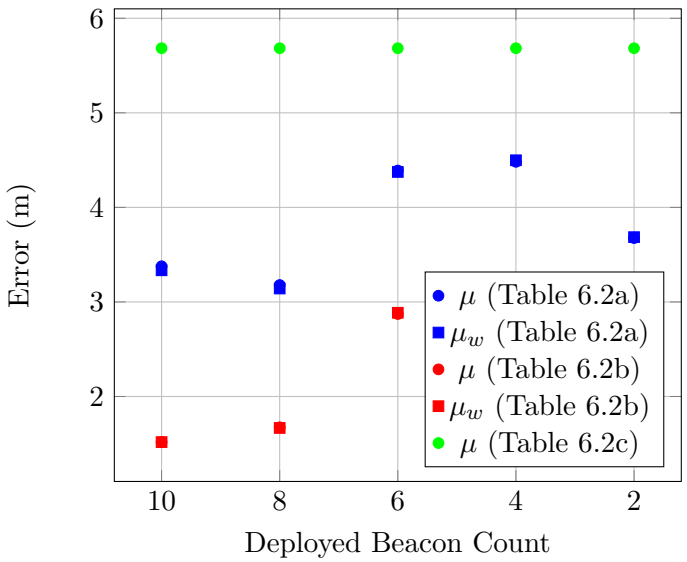
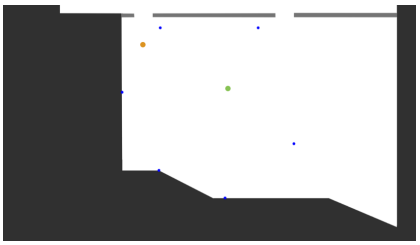


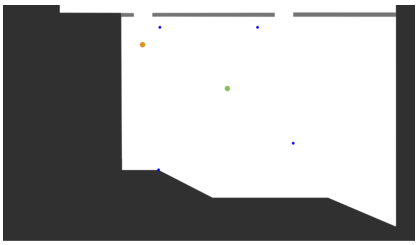
Figure 6.4: Visualizes the results of experiment 2 shown in Table 6.2.

(a) 10 Beacons

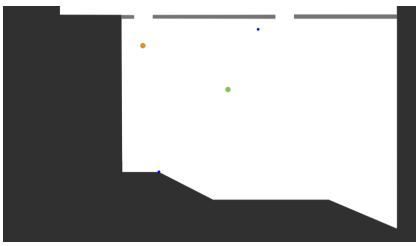
(b) 8 Beacons



(c) 6 Beacons



(d) 4 Beacons



(e) 2 Beacons

Figure 6.5: Beacon deployment during experiment 2. The blue circles depict the beacons' positions. The orange circle shows the person's start position. The person's random walk stopped at the green circle.

First of all, the result clearly shows that the difference between χ's mean and its weighted mean is negligible. Secondly, it clearly shows that the estimated location compared to the motion is more accurate. Additionally, it depicts that the location estimation drastically improves after being stationary, for some seconds, i.e. for $\approx 5\,\mathrm{s}$. The result also indicates that the estimation's accuracy depends on the deployed number of beacons. Thus, the more beacons, the less error in the position estimation. But it also shows, that at a certain beacon density the error's reduction converges. Furthermore, deploying more beacons does not always improve accuracy. This seems to be the case in the experiment by reducing the beacon amount from four to two deployed beacons. Supposedly, the signals of the two removed beacons were too bad, and thus produced a worse estimation result.

Experiment 3

After evaluating the solution's accuracy in the first two experiments, this experiment shows how good the solution is able to track a users path in a larger scenario. Hence, the buildings foyer, depicted in Figure 6.6, and one / two lecture halls were additionally used. For the experiments, only 10 beacons were used, as shown on the maps. The maps depict an actual

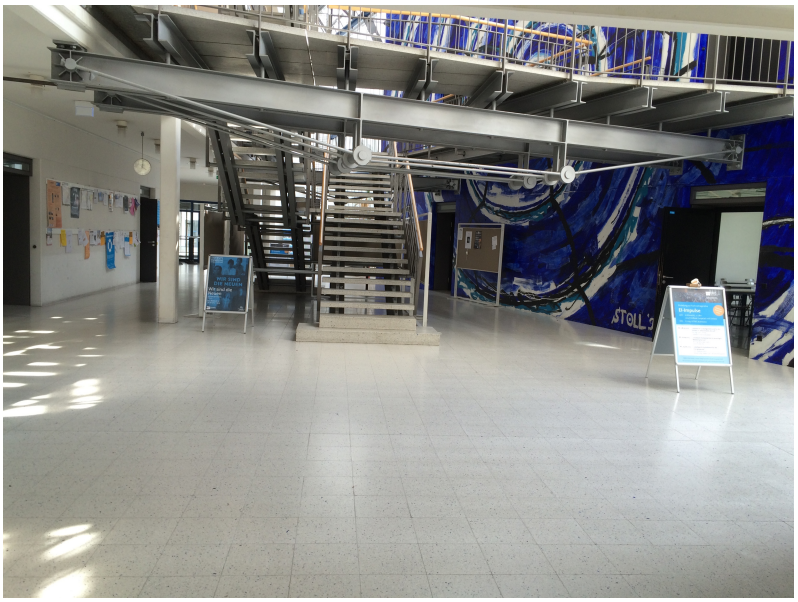


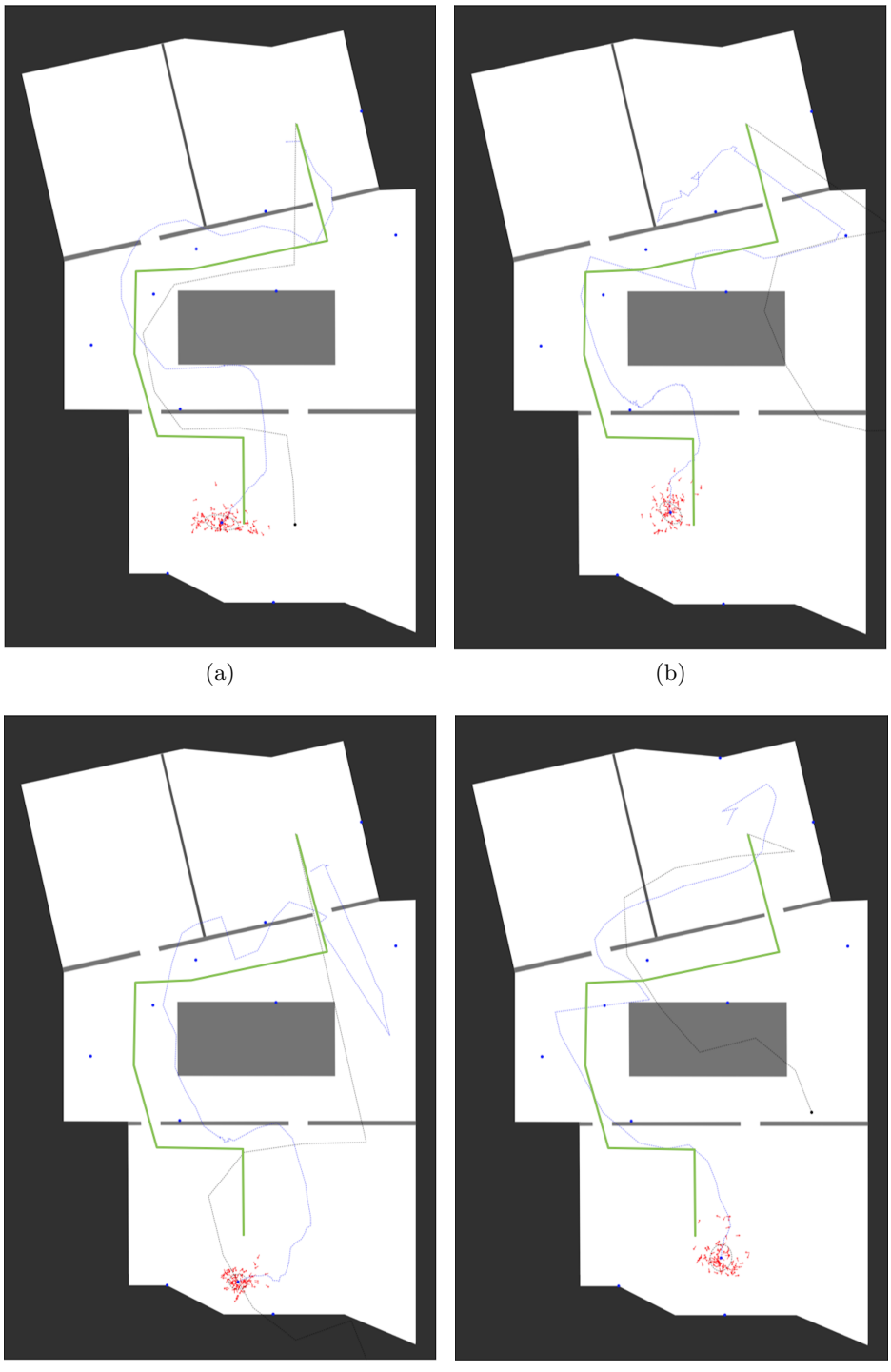
Figure 6.6: The building's foyer depicted on the map's center.

space of $\approx 22.22\,\text{m} \times 33.06\,\text{m}$.

The first example is depicted in Figure 6.7. The green line approximates the actual path, beginning at the top right. The black dashed line shows the motion path, which is just the combination of pedometer and heading. The estimated path is marked as a blue dashed line. Figure 6.7a shows an example where the motion was very accurate, actually more accurate than the estimated path. Whereas, Figure 6.7b, 6.7c, and 6.7d depict examples, where the motion path is bad or completely wrong, but the algorithm was more or less able to approximate the actual path.

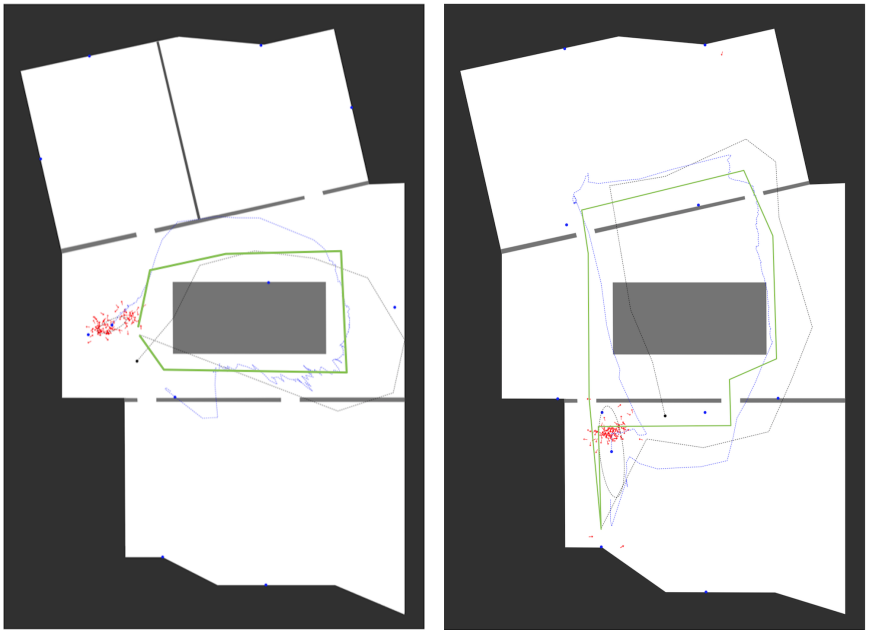
Figure 6.8a depicts another example, where the test person walked around the big stairs in the middle of the building's foyer. The screenshot shows, that the algorithm was able to approximate this path as well. During this test the foyer was equipped with only 4 beacons as shown on the map. But, the experiment also showes that 4 beacons are not sufficient for such a large area if the motion is incorrect.

Another example path is shown in Figure 6.8b. During this test the wall between the two upper halls was opened. As shown, the algorithm was able to track the path. But usually this would not be possible without having such an accurate motion, even by using 10 beacons. Furthermore, the screenshot shows that the estimated end position is actually not at the true position, but rather $\approx 4-5\,\text{m}$ too distant. But two particles are located next to the true position, and the plotted $1\sigma-$ellipse is oriented towards these two particles. The reason for that is, that the algorithm's weight sum dropped below the recovery's threshold, and thus recovery was executed. Unfortunately, the data recording was stopped at this point in time, otherwise it is very likely that the particle cloud would have jumped to the true location where the two particles were already located.



(a) (b) (c) (d)

Figure 6.7: The screenshots illustrate example tracks of experiment 3. The green line shows the actual path. The blue filled circles depict the beacons. The blue dashed line shows the estimated path. The particles are depicted as red arrows. The black dashed line shows the motion path.



(a) (b)

Figure 6.8: The screenshots illustrate example tracks of experiment 3. The green line shows the actual path. The blue filled circles depict the beacons. The blue dashed line shows the estimated path and the black line the motion path. The particles are depicted as red arrows.

Summary

The algorithm was tested in a typical environment with enabled WiFi, obstacles and many reflecting materials, as shown on the building's images.

The experiments demonstrate that the estimation's accuracy depends on the amount of deployed beacons, but also on their position and attitude relative to the user, which influences the beacons' signals. Furthermore, the experiments showed, that the estimated position not only depends on the beacons' signals. It also depends on the motion's quality. As shown by experiment 3, location estimation is possible with having only a few beacons deployed, but for that the motion must be relatively correct. Besides that it also proves, that if enough beacons are deployed, the solution is able to correct the path on the basis of the beacon's signals.

Furthermore, the experiment's results show, that the location estimation improves after being stationary for a few seconds, if enough beacons are deployed.

Due to the fact that the application's accuracy and precision depend on so many different factors, it is possible to determine, what the system achieved during the experiments. But it is not possible to make a generally valid statement about the system's precision and accuracy. During the experiments it reached an accuracy of up to 1.52 m (experiment 2). By combining the results of experiment 1 and 2 (with 10 beacons), the system reached an accuracy of 2.12 m ($\mu = 2.12$, $\sigma = 2.72$). The solution's precision, expressed as Comulative Distribution Function (CDF), is shown in Figure 6.9.

To summarize, the solution provides at least room level or even better accuracy, depending on the deployed beacon count and the above mentioned factors. Furthermore, the solution

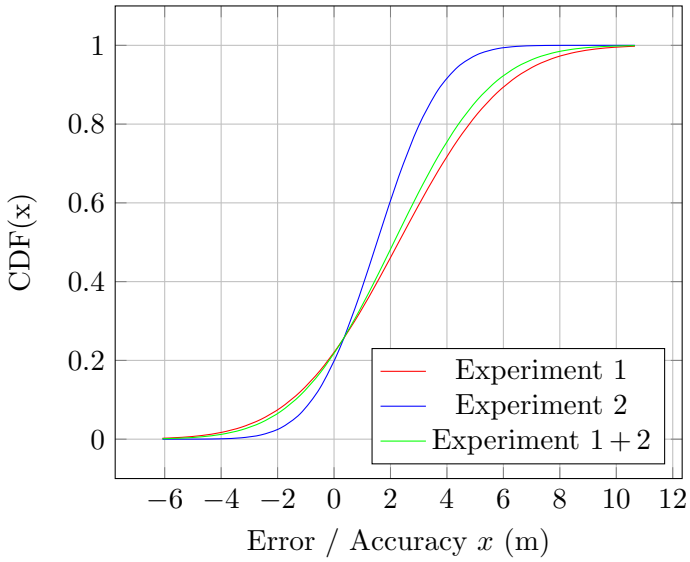


Figure 6.9: The solution's precision achieved during the experiments.

is able to track a user's path in environments with a lot of free space, as shown by the experiments. Consequently, the system's ability to track a user's path in a corridor should be very good, due to the fact that it takes the map into account. The solution's only disadvantage is, that the user has to hold the device in the hand by pointing it into the user's walking direction. Otherwise the motion's orientation is completely wrong, and thus recovery often needs to be executed.

6.2 Scalability

The area covered by the localization system depends only on the provided map and the deployed beacons. Thus, the system is highly scalable. It can be used within a single room. But it also can be extended to a very large, two-dimensional space.

6.3 Robustness

The system is very robust. It can deal with wrong and missing beacon signals, which is a very common problem. Furthermore, it is able to deal with wrong motion data, caused by influenced heading or wrong distance estimations provided by the pedometer, as shown by the experiments.

Due to the implemented recovery, it is also able to detect completely wrong position estimations. The only requirement for the recovery is that the application needs to at least receive the signal from one beacon.

6.4 Complexity

The solution's complexity is relatively low. Besides the beacons and a typical smartphone, no additional infrastructure, such as a network connection or external power supply, is required.

Also, the software complexity is relatively low, due to the used PF algorithm. Of course, if a platform does not provide such high level API's, especially such as CoreMotion's pedometer to estimate the walked distance, this low level of complexity would not be achieved.

6.5 Cost

Besides the beacons no additional infrastructure is required. Furthermore, the users can use their own smartphones. Thus, the building owner just needs to deploy the beacons. Consequently, the system's cost mainly depends on the area to be covered, and thus on the required amount of beacons and the initial deployment and setup costs. 100 beacons (incl. batteries) from BEACONinside cost 1790EUR, but cheaper manufacturers might exist.

Furthermore, the system's maintenance effort is relatively low. Typically, the beacons' battery lifetime is up to a year or more. And as long as no big changes to the beacons' environment are made, recalibration is not necessary.

6.6 Usability

The localization system is very easy to use. The user just has to download the app from the platform's app store, e.g. a chain of store's app. After launching and providing the application access to Bluetooth the localization can start.

The user interface can be reduced to a minimum by only showing the environments map, the estimated position and the corresponding $1\sigma -$ ellipse, instead of showing the beacons, the motion path, the user's estimated path, and the particle cloud.

There are only two disadvantage from a usability point of view, which is on one hand that the smartphone needs to point into the user's walking direction, and on the other hand the position estimation's delay. If the user walks the estimated position is only updated ever $\approx 3\,\mathrm{s}$, due to the fact, that CM just provides distance estimations every $\approx 2.5\,\mathrm{s}$, as mentioned in Chapter 5. If the user is stationary, the delay drops to $\approx 1\,\mathrm{s}$, which is negligible from my point of view.

Chapter 7

Conclusion

At the beginning three example use-cases for indoor self-localization, which could provide benefit to our lives, were mentioned. These and other use-cases are now, from my perspective, achievable goals, because as of now, the problem was rather the localization approach than the large beacon deployments, as recently shown at the *South By South West (SXSW)* music and film festival in Austin (TX, USA). There the festival organizers deployed more than 1000 beacons to enhance the festival with social networking features (Ulanoff, 2015).

In this work I presented an indoor self-localization approach for smartphones using iBeacons to exactly solve this problem. The solution is a cross-platform solution, that works with common smartphones, equipped with built-in sensors and a Bluetooth Smart, i.e. Bluetooth Low Energy, chip.

The system's complexity is very low. Besides the smartphones and the beacons, no additional hardware or infrastructure is required, which also reduces the initial and maintenance costs. The whole localization takes place on the user's smartphone. Furthermore, the location estimation is very robust compared to other approaches.

Because of high uncertainty of the RSS based distance estimation to the deployed known beacons, the solution relies not only on the wireless signals. In addition, it uses the smartphone's built-in sensors to determine the user's motion. Furthermore, the building's map, which represents its free and occupied space, is integrated. Finally, the solution combines all three sources by using a Particle Filter to estimate the user's location and to reduce the overall uncertainty.

As a result, the approach solves the global localization problem, and is also able to detect and to recover from wrong position estimations. During the experiments, a user's stationary location with a mean error of 2.29 m, was achieved. Furthermore, it was possible to track a user's path in the test environment.

7.1 Future Work

There are still a few open questions. Some of them are merely interesting, others are necessary for future improvement.

Companies such as Walmart and Walgreens announced in May 2014, that they will deploy beacons, by using General Electric's in-store LED lighting fixtures, which additionally act as beacons. Thus, it would be interesting to know, if the localization could be improved by deploying beacons at the building's ceiling, i.e. above the users (Kahn, 2014).

Additionally, it would be interesting to know, if a better location estimation could be achieved by using Apple's new iPhone 6/ iPhone 6 Plus, which is shipped with a new motion coprocessor.

The new iPhone also is equipped with a new sensor, a barometer. Thus, it would be worthwhile to extend the solution to three-dimensional space and to integrate CoreMotion's floor ascending and descending capability.

Furthermore, it would be good to know, if there are differences in the beacon signal's quality, especially if a more accurate proximity estimation would be possible, by using beacons

from other manufacturers.

Another very interesting topic, maybe a research topic, would be to find out, if the solution would benefit from solving the multi-"robot" problem, by turning the device itself into a beacon, and thus having the additional possibility of communicating with other devices.

But of course, the most interesting would be to see how good the application would perform in large deployments, for instance at a trade fair.

References

Apple Inc. (2014a). *Getting Started with iBeacon*. Apple Inc. Version 1.0.

Apple Inc. (2014b). iBeacon for Developers. `https://developer.apple.com/ibeacon/`. Last access: 23.04.2015.

Apple Inc. (2014c). *iOS 8.1 CoreLocation framework documentation*. Apple, Inc.

Apple Inc. (2014d). *iOS 8.1 CoreMotion framework documentation*. Apple, Inc.

Apple Inc. (2014e). *Swift*. Apple Inc. Published published in Apple's iBooks store.

BEACONinside GmbH (2014). *BEACONinside iBeacon Datasheet*. BEACONinside GmbH. Model No.: B0001-A HW: Rev 1.1 SW: 1.0.

Bluetooth SIG, Inc. (2010). *Specification of the Bluetooth System Version 4.0*. Bluetooth SIG, Inc. Vol. 0.

Bruins, J. (2012). Staying on track with location services. Video recording of talk at Apple's WWDC 2012. Session 303.

Bruins, J. (2013). What's new in core location. Video recording of talk at Apple's WWDC 2013. Session 307.

Hoflinger, F., Hoppe, J., Zhang, R., Ens, A., Reindl, L., Wendeberg, J., and Schindelhauer, C. (2014). Acoustic indoor-localization system for smart phones. In *Multi-Conference on Systems, Signals Devices (SSD), 2014 11th International*, pages 1–4.

Hoflinger, F., Zhang, R., Hoppe, J., Bannoura, A., Reindl, L., Wendeberg, J., Buhrer, M., and Schindelhauer, C. (2012). Acoustic self-calibrating system for indoor smartphone tracking (assist). In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*, pages 1–9.

Kahn, J. (2014). GE integrates iBeacons in new LED lighting fixtures rolling out in Walmart and other retailers. `http://9to5mac.com/2014/05/29/ge-integrates-ibeacons-in-new-led-lighting-fixtures-rolling-out-in-walmart-other-retailers/`. Last access: 23.04.2015.

Kotanen, A., Hannikainen, M., Leppakoski, H., and Hamalainen, T. D. (2003). Experiments on local positioning with bluetooth. In *Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on*, pages 297–303. IEEE.

Liu, H., Darabi, H., Banerjee, P., and Liu, J. (2007). Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(6):1067–1080.

Oksar, I. (2014). A bluetooth signal strength based indoor localization method. In *Systems, Signals and Image Processing (IWSSIP), 2014 International Conference on*, pages 251–254. IEEE.

Pham, A. (2012). Understanding core motion. Video recording of talk at Apple's WWDC 2012. Session 524.

Pham, A. and Chow, S. (2014). Motion tracking with core motion framework. Video recording of talk at Apple's WWDC 2014. Session 612.

Shanklin, T. A., Loulier, B., and Matson, E. T. (2011). Embedded sensors for indoor positioning. In *Sensors Applications Symposium (SAS), 2011 IEEE*, pages 149–154. IEEE.

Siddiqi, S., Sukhatme, G. S., and Howard, A. (2003). Experiments in monte-carlo localization using wifi signal strength. In *The 11th International conference on advanced robotics, Coimbra, Portugal*.

Siddiqui, M. H. and Khalid, M. R. (2012). A novel approach for tracking of a mobile node based on particle filter and trilateration. 6(4):788 – 793.

Straub, J. (2010). Pedestrian indoor localization and tracking using a particle filter combined with a learning accessibility map. *Bachelor Thesis*.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*. MIT Press, Cambridge, Mass.; London.

Ulanoff, L. (2015). Near Me is dead, long live iBeacons — at SXSW. `http://mashable.com/2015/03/13/ibeacons-sxsw/`. Last access: 23.04.2015.

Wang, H., Lenz, H., Szabo, A., Bamberger, J., and Hanebeck, U. D. (2007). Wlan-based pedestrian tracking using particle filters and low-cost mems sensors. In *Positioning, Navigation and Communication, 2007. WPNC'07. 4th Workshop on*, pages 1–7. IEEE.

Wang, Y., Yang, X., Zhao, Y., Liu, Y., and Cuthbert, L. (2013). Bluetooth positioning using rssi and triangulation methods. In *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*, pages 837–842.