CLOUD-BASED INDOOR POSITIONING - BLUETOOTH CLIENT AND PARTICLE FILTER

by
PATRICK HODEL

Supervisor
PROFESSOR DR. TORSTEN BRAUN

Institute of Computer Science
Communication and Distributed Systems
Universität Bern

Januar 31, 2019

Patrick Hodel: *Cloud-Based Indoor Positioning - Bluetooth Client and Particle Filter ,* Januar 31, 2019

ABSTRACT

The rise of smartphones and Wifi offers a new potential for Indoor Localization technology. Using Radio Signal Strength Indications (RSSIs) of either WiFi access points or Bluetooth Low Energy (BLE) beacons captured with a smartphone or an integrated platfrom the distance to the emitting device can be estimated. Additionally, Inertial Measurement Unit (IMU) data can be captured for step estimation. Using a particle filter we can fuse this data to reduce the localization error.

This thesis presents and demonstrates a particle filter implementation on a cloud server accessible with thin clients over the WebSocket protocol. This server also allows administration of a location, recording of collected data and replaying stored and live computed positions on a floorplan. Further, an iOS client collecting Bluetooth packets is implemented and evaluated. The iOS implementation based on iBeacons, however, did not deliever expected results as the ranging data was too unstable and inaccurate to be useful for localization purposes. The particle filter implementation with data from WiFi ranging improved localization when compared to Pedestrian Dead Reckoning (PDR) localization.

Soli deo gloria.

ACKNOWLEDGMENTS

I would like to thank Zhongliang Zhao and José Luis Carrera for their supervision of the thesis and their continous support. Furthermore I would like to thank Stefan Serena and Lucien Madl for the fruitful collaboration.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

RSSI Radio Signal Strength Indication

SDK Software Developement Kit

CDF Cumulative Distribution Function

IMU Inertial Measurement Unit

BLE Bluetooth Low Energy

ISM industrial, scientific and medical

LOS line of sight

SUS Stochastic Universal Sampling

API Application Programming Interface

GPS Global Positioning System

PDR Pedestrian Dead Reckoning

AR Augmented Reality

UI user interface

NEFF effective sample size

1 INTRODUCTION

1.1 BACKGROUND

The Global Positioning System (GPS), originally developed for the US Military and declared fully operational in April 27, 1995, is now ubiquituous in our daily life. Countless applications in aviation, on roads or marine and in tasks such as surveying and mapping underline the utility of the system [28]. GPS fundamentally relies on the reception of signals transmitted by satellites. Conventional consumer electronics will often fail to determine an accurate position in indoor environments. New research in indoor positioning yields solutions to this challenge.

This thesis is following up on recent work[4]. The proposed solution makes use of WiFi RSSI values as captured by an Android mobile phone to compute an approximate position. Due to Software Developement Kit (SDK) limitations imposed by Apple it is currently impossible to gather RSSI values of individual WiFi access points. Therefore, WiFi cannot be used in indoor positioning systems with iOS. In 2013 Apple introduced iBeacon, a protocol for indoor localisation[10]. Using specialized hardware (iBeacons) that continuously emits a BLE signal instead of the WiFi access points might allow the same approach as suggested by [4] to be used with iOS.

iBeacon as a replacement for WiFi based positioning

The computation of a position using a particle filter is resource intensive. We, therefore, propose a server responsible for computation and clients solely responsible for data capturing (herin called thin clients). Server-based processing brings the advantage of greater processing capabilities, thus, allowing for faster and more accurate computation. An additional advantage of using a server for computation is that only one implementation must be maintained vs. three individual implementations on each client. In this thesis an iOS client for data capture and a server implementation for computation is developed.

The client simultaneously captures RSSIs of iBeacons and IMU data, then continously sends this data to the server. In parallel, two other clients by Lucien Madl [22] and Stefan Serena [33] have been implemented, which both capture WiFi data and IMU data and continously send this data to the server. One client has been implemented on an Android smartphone, while the other has been implemented on a ESP32 microcontroller. All clients are expected to be implemented against the same server.

On the server, three modules are implemented. The rotations module receives IMU data, computes the device heading and determines

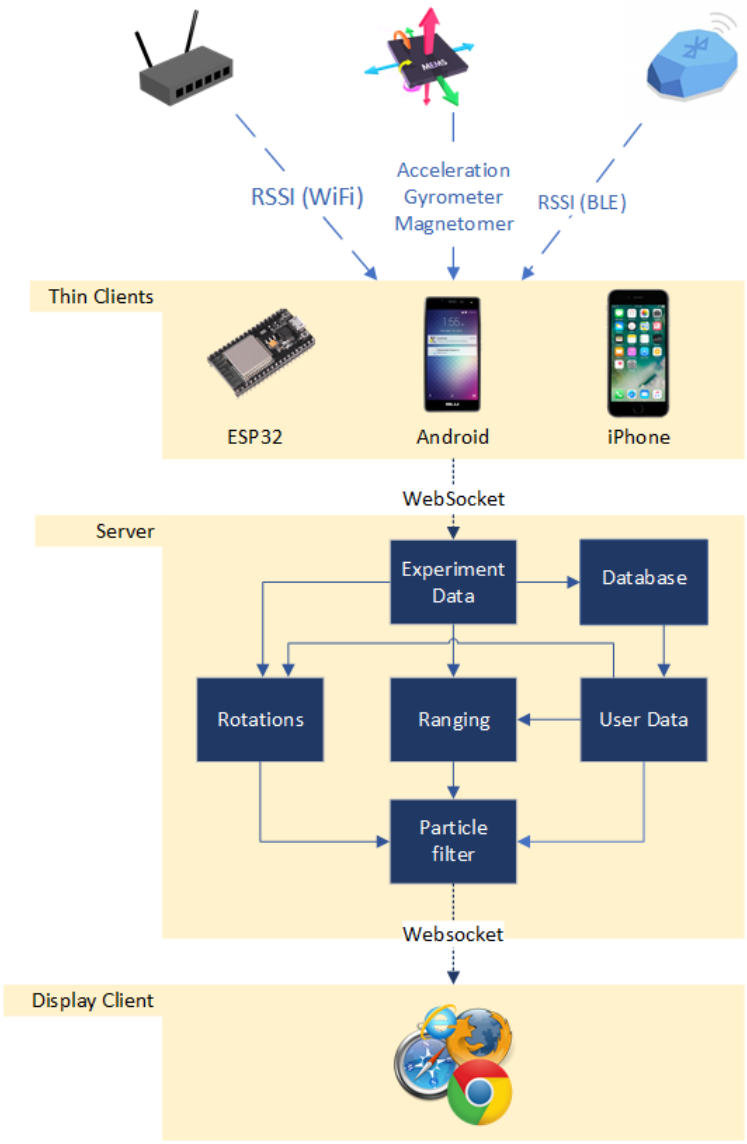

Figure 1.1: Conceptual overview of the system

a step vector. The ranging module computes the distance to access points based on RSSI values. The results of these two modules are consumed by the particle filter to compute a position. The entire track can then be drawn on a map accessible through a browser. Additionally, an admin panel to configure locations, devices, recorded tracks and their ground truth was implemented.

1.2 CONTRIBUTIONS

This work presents a localization system consisting of a) thin clients responsible for IMU and ranging data collection and b) a localization server providing an interface for data gathering clients, an implemen-

tation of a positioning algorithm, a graphical user interface for the administration of a location and replaying of recorded and live tracks.

This thesis is part of a joint effort with Stefan Serena [33] and Lucien Madl [22]. The goal is to provide a cloud-based centralized indoor localization solution for multiple clients. The contributions of this thesis are the following:

1. We implement and evaluate an iOS client that periodically sends IMU data (accelerometer, gyroscope and magnetometer) and RSSI values from iBeacons to the server for further processing.
2. We present and evaluate a particle filter implementation running on a server to fuse the RSSI data and movement vectors.
3. We provide an implementation of a browser-based map to replay stored and live tracks with their computed path, ground truth path, trilaterated path and PDR path.

1.3 OVERVIEW

This thesis is structured as follows: Chapter 2 gives an introduction to the theoretical background of iBeacons and the usage of particle filter's for indoor localization and briefly discusses some related work. Chapter 3 highlights some implementation details, chapter 4 evaluates the performance and chapter 5 contains the conclusion.

2

THEORETICAL BACKGROUND AND RELATED WORK

This chapter presents the theoretical background and introduces some related work.

2.1 BLE IBEACON

2.1.1 *Bluetooth Low Energy*

BLE has been introduced in the Bluetooth specification version 4.0 in 2010. It operates on the unlicensed 2.4 Ghz industrial, scientific and medical (ISM) band. BLE distinguishes between advertisers and scanners. Devices that transmit advertising packets are called advertisers. Devices that receive advertising but do not have the intention to connect to the advertisers are called scanner[2].

The iBeacon technology has been introduced by Apple with iOS7 in 2013 [40]. The technology uses BLE with a specific advertisement packet. Devices with iBeacon technology (iBeacons) send an advertisement consisting of a field with a size of 16 bytes called UUID and two fields with a size of 2 bytes called Major and Minor. UUID is typically the same for all iBeacons of a deployment use case. The Major/Minor numbers can be used to define regions depending on the use case. Further, it typically contains the calibrated signal strengths at a 1 meter distance to estimate distances[13].

iOS provides Application Programming Interfaces (APIs) to receive a notification when a device enters a region defined by one or multiple beacons. For ranging Apple provides APIs to determine an estimated proximity to a beacon indicated with four proximity states:

IMMEDIATE indicates a high level of confidence that a device is physically very close to an iBeacon.

NEAR indicates a proximity of approximately 1-3 meters given line of sight (LOS).

FAR indicates that a beacon was detected but the level of confidence is too low to determine the states Near or Immediate.

UNKNOWN indicates that the proximity cannot be determined (e.g. because there are insufficient measurements).

Additionally, for each measurment there is an accuracy property indicating an estimate of the distance in meters and an RSSI property,

which holds the average of the received RSSI values for a beacon since the last reading.

Typically iBeacon hardware batteries last up to 6 months or more. iBeacons are readily available by a number of manufacturers and can be deployed easily.

2.1.2 *BLE-based indoor localization systems*

Martin et al.[23] have successfully demonstrated localisation with iBeacons using an external server and particle filtering. In the test setup three beacons were set up in a room where each beacon has a distance of approximately 7 meters to the next beacon. On a path within the triangle spanned by these three beacons, an average error of 0.53 meters has been achieved. Li et al.[20] have employed a Kalman filter for indoor positioning with iBeacons. They deployed 36 beacons and tested positioning at 15 fixed points over an area of approximately 600 square meters. For each coordinate, tests have been repeated three times, resulting in an average error of 2.9 meters over all points. Further Zhong et al.[42] used iBeacons for more precise localization in the context of indoor Augmented Reality (AR). They observed a relatively good performance at distances of 0-3 meters. In their setup in an empty room of 6x6 m with 9 equally distributed beacons, resulting in one beacon for each 3 meter circle, an error between 0.16-0.23 meters was observed at five measured positions.

2.2 PARTICLE FILTERS

2.2.1 *Basics of Particle Filters*

The Particle Filter is a filtering algorithm used for state estimation. In this work it uses floorplan information, WiFi and IMU signals. We initialize it using the tuple P as parameter.

$$P = (\sigma, \mu, p, f, N, \alpha, \beta, l, s)$$

The meaning of each parameter is the following:

σ is the parameter describing the variance of the normal distribution used to assign probabilities to the particles.

p is a set of n particles where each particle consists of a vector and a weight noted as $p.position$ and $p.weight$.

f represents the floorplan as a set of vectors which represent all inaccessible positions (e.g. because of walls).

N is the effective sample size (NEFF) used to trigger resampling when the particles have degenrated execessively. Degeneration describes the situation, where, after a few iterations, all but one particle have negligible weight[1].

α represents the average angular offset to be used to generate movement vectors of the particles given step vectors.

β represents the variance of the angle to be used to generate movement vectors of the particles given step vectors.

l represents the average length offset of the step length to be used to generate movement vectors of the particles given a step vector.

s represents the variance of the step length to be used to generate movement vectors of the particles given a step vector.

The algorithm consists of several stages which are hereafter explained.

INITIALISATION A set of particles p is randomly distributed across the floorplan while inaccessible positions are avoided. If the starting position is known, the particles are distributed around the starting position. The weight assigned to each particle is $\frac{1}{n}$.

UPDATE When new step vectors or ranging results are available, an update is triggered. For each particle the step vector v_s is multiplied by s' with $s' \sim \mathcal{N}(l, s)$ and rotated with α' with $\alpha' \sim \mathcal{N}(\alpha, \beta)$. The resulting vector shall be called v'_s. The new particle's position $p.position'$ is then

$$p.position' = p.position + v'_s$$

After moving the particles, their weights are recalculated and systematic resampling takes place. If a new ranging result from one access point is passed, it is cached. Weight recalculation and resampling is triggered once new ranging results of all access points are available.

Using the floorplan information of f we can disallow particles to reach an invalid position. This case is formally described as

$$p.position' \in f$$

If this is the case we want to set the particle back to the last possible valid position along its projected movement path. Formally, we have to find $i \in [0, |v'_s|]$ such that

$$p.position + i \cdot v'_s \in f$$

It may still be possible that a particle crosses a wall. If this is disallowed, it may be possible that all particles are trapped in a room if no particle exits the room (through an exit). We still want particles

that are not affected by walls to be favoured. Therefore, we introduce a weight factor w' that is multiplied with the current particle weight. It is defined as follows

$$w' = \left(\frac{1}{|v'_s|}\right)^2 \cdot b_1 \cdot b_2$$

where b_1 is the distance the particle filter could not travel because its new position would have been on a wall and b_2 is the distance the particle travels on a wall.

WEIGHT RECALCULATION The recalculation consists of several steps. First, an access node likelihood a_{ij} for each particle p_i and access node q_j is computed using the following formula:

$$a_{ij} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x_j - \left|p_i - q_j\right|}{2\sigma^2}},$$

where x_j is the measured ranging distance to access point j and $|\cdot|$ is the euclidian metric. In this case it computes the distance from particle p_i and access point q_j. These values are weighted differently such that values corresponding to an access node that is closer contribute more towards the final result. For this the variable u_j corresponding to access point j is introduced:

$$u_j = \frac{x_j^{-1}}{\sum_{0 \le i \le m} x_i^{-1}},$$

where m is the number of access nodes. The weight w_i or particle p_i is then computed as follows:

$$w_i = \prod_{0 \le j \le m} a_{ij}^{u_j},$$

These weights are then normalized such that the sum of all weights equals 1.

$$w'_i = \frac{w_i}{\sum_{0 \le j \le n} w_j},$$

SYSTEMATIC RESAMPLING Systematic resampling describes the process of eliminating weak positions and replacing them with stronger ones. All particles with a weight lower than the given parameter are collected in a set. Formally

$$N' = \frac{\frac{1}{n}}{N}$$
$$r = \{p_1, p_2, \ldots, p_{n'}\} = \{p : p.weight < N' \wedge p \in P\}$$

The new positions are then choosen among the remaining particles $v = p \backslash r$ using a version of Stochastic Universal Sampling (SUS).

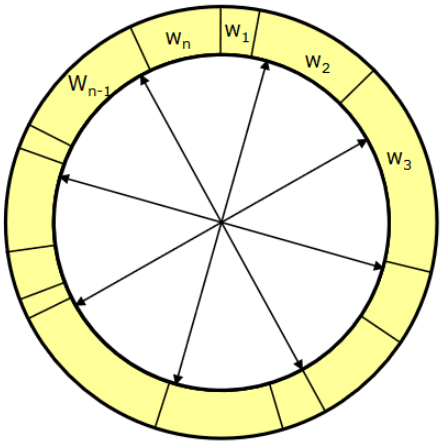

Figure 2.1: $w_1, w_2, \ldots, w_n$ represent particles weight. In this example 8 particles are chosen. The angle of the arrows represent the random number R.[1]

The selection algorithm is schematically illustrated in Figure 2.1 and can be described by the pseudocode in listing 2.1[34].

The positions of the returned particles are now applied to the particles in r. The process ends with normalizing the particle weights again and is repeated as long as

$$\frac{1}{\sum_{0 \leq i \leq n} p_i.weight^2} < N'$$

holds.

2.2.2 *Particle Filter-based Indoor Localization Systems*

Zhao et al. [41] have presented an indoor localization system using a particle filter implementation with an average tracking error of 1.7 meters. This work applies the same concepts as presented in [41] to a cloud-based localization system with an iOS smartphone and an integrated platform as clients. In [8] a similar architecture with the extension that a particle filter is used to calibrate the IMU was used.

Others have used a hidden markov model for localization using RSSI and accelerometers[19].

Sung, Lee, and Kim [35] have achieved a localization error below 10cm using only a smartphone (iPhone 5s) for computation. They have employed machine learning algorithms, Kalman filters and support vector machines with RSSI measurements of iBeacons and WiFi access points. While the best accuracy was achieved with particle filters, they considered alternative methods more suitable due to significantly faster computation. In this work we have instead used cloud computing to circumvent computational constraints on smartphones and

Listing 2.1: Pseudocode of stochastic universal sampling

```
SUS(Particles, N)
        F := total weight of particles
        N := number of particles to resample
        P := distance between the weights (F/N)
        Start := random number between 0 and P
        Pointers := [Start + i*P | i in [0..(N-1)]]
        return RWS(Particles,Pointers)

RWS(Particles, Points)
        Chosen = []
        for P in Particles
                i := 0
                while weight sum of Particles[0..i] < P
                        i++
                add Population[i] to Chosen
        return Chosen
```

did not use machine learning for computation. The work of Lo et al. [21] focused on multi-floor building incorporating elevators and stairs. They propose a robust particle filter weighting mechanism and achieve a good floor detection accuracy. Their problem statement is slightly different from the problem statement of this work, as this work does not include floor detection. A novel, multidimensional particle filter has been introduced by Pei et al. [29]. Additionally, their heading estimation incorporates floorplan information which allows computation of path constraints for the vector computed by PDR. Using their multidimensional particle filter, they have been able to reduce the 67th error percentile by 4cm from 58cm to 54cm and the 95th error percentile by 35cm from 124cm to 89cm, when compared to an ordinary particle filter as used in thiw work.

Rajeshirke and Dhage [31] have implemented a similar approach to this work with a particle filter to fuse detected step vectors and detected RSSI distances. In their evaluation they show the effect of people in a room and tested with different smartphones.

3 DESIGN AND IMPLEMENTATION

3.1 SERVER ARCHITECTURE

In this thesis, thin clients are employed for data collection. The idea of a thin client is that it is optimized for establishing a remote connection and does only minimal computation on the device itself. The computation is done in realtime on a server. Computed positions can optionally be sent to a browser based client and displayed on an interactive map. Figure 3.1 shows a conceptual overview of the system architecture. Three types of clients collect IMU data and RSSI data of either iBeacons or wireless access points. This data is sent to the server via WebSocket for storing and computation. Using experiment data and user information, such as device specific ranging models and floorplan information, a position is computed. The rotations module computes a step vector, the ranging module computes access point distances and the particle filter computes the final position by fusing this data. A position update is then sent via WebSocket to a browser. A more in depth description can be found in [33]. As part of this work, the architecture has been extended to allow recording of tracks. Additionally, recorded tracks can be annotated with ground truth positions reached. These known ground truth positions reached at a known time are herein called *waypoints*.

3.2 BLUETOOTH CLIENT

While the work of Zhao et al. [41] employed Android clients to collect RSSI of WiFi access points, this thesis seeks to extend the approach to iOS clients. A search on stackoverflow reveals, that Apple will not allow to access this data[6, 14–17, 26, 27]. Apple had apps such as WiFi scanners using this functionality in 2010. Using private APIs it may still be possible to implement this functionality, but any application making use of it would not be allowed in the AppStore[12]. For this reason this client only implements collection of RSSI of iBeacons. This is the approach Apple recommends for localization tasks using iOS.

3.2.1 *Available iBeacons*

The term iBeacon stands for the BLE-based protocol developed by Apple as well as hardware compatible with the iBeacon protocol. Apple does not produce iBeacons themselves, but instead instead allows certified manufactures to build and sell their own iBeacons.

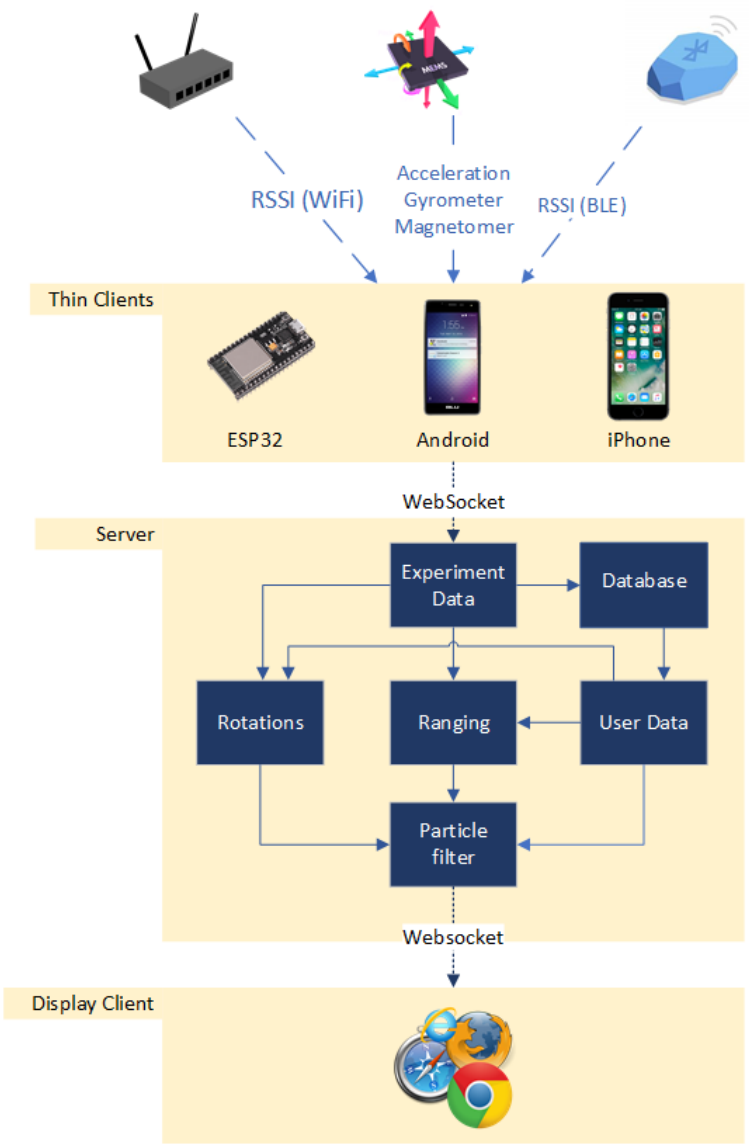

Figure 3.1: Conceptual overview of the system architecture.

Figure 3.2: A selection of available iBeacons[24]

Currently there are a number of different iBeacons with different properties available. As shown in Figure 3.2, they come in various sizes, shapes and colors. While most iBeacons feature relatively small internal batteries, some vendors produce iBeacons that are powered by USB or power outlet. Typically the firmware allows control over transmission power and advertising interval. Most manufacturers use either a Gimbal, Nordic, Bluegiga or TI CC254x chipset. For this thesis we chose to work with Estimote iBeacons. Besides an iPhone app to edit the parameters, the Estimote iBeacons feature a web admin panel and supports a number of other BLE-based protocols such as Google's Eddystone.

3.2.2 *iOS Client Development Stack*

The iOS client depends on the following software and libraries:

SWIFT The language used to program the iOS application. It runs on all current versions of iOS and is the successor of ObjectiveC. It is also used to program Apple Watch applications, iPad appli-

cations and Apple TV applications. Swift features a completely new syntax.

IOS The operating system used on the iPhone. Specifically, we were developing for iOS 11.3. Later versions, such as the current iOS 12 have not been tested. In principle the app should be compatible with iPads but its design has not been optimized for it.

PODS Pods are used as the package manager for iOS applications. Specifically, we have used it to include current versions of the Estimote SDK as well as the Dropbox api and WebSocket API.

3.2.3 *Goal/Design of the client*

The iOS client fullfills the following purposes:

1. At early stages of developement we had to gather IMU and RSSI data for manual analysis on a computer. To faciliate this purpose, a specific user interface (UI) and functionality for data recording and uploading was implemented. It allowed recording IMU and RSSI data to a file that is subsequently uploaded to a Dropbox. The UI featured a start/stop button, an input field for the filename and an additional button to add a reference at the current time in the recorded file. During the recording, the currently recorded values were displayed in textfields.

2. To verify that all iBeacons as well as the application are functioning properly, a sepearate page in the app was added to display a list of iBeacons in the vincinity and their measured distance.

3. Originally, it was intended to show the position of the device on a map on the iPhone. Some developement went into the UI for a map, but later it was decided to use a browser instead.

4. The client had to upload its measured values in real time via websocket. Another page with a "connect" button to set up a connection with the server and a "start" button to start uploading captured values was added. An additional button allowed to add a reference in the current recording.

3.3 PARTICLE FILTER FOR POSITION COMPUTATION

The particle filter receives data from the rotation and ranging components. Specifically, a step vector is passed whenever a step is detected. The ranging component forwards all computed distances to the individual access points. In this section some of the concepts are explained.

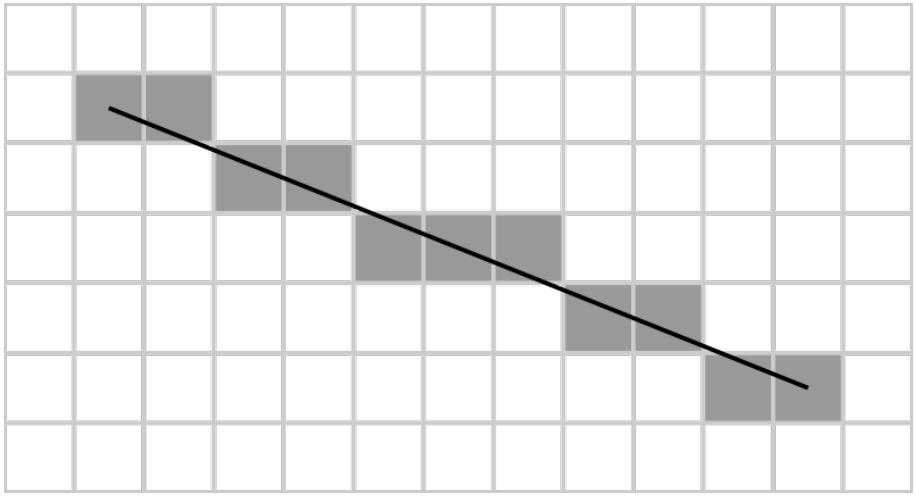

Figure 3.3: The Bresenham line rendering algorithm[3].

3.3.1 *Step Vector Computation*

The step vector component detects steps based on IMU data. An in-depth description of the used module to detect and generate a step vector is available in [22]. Once a step has been detected, the particle filter receives the computed step vector. The vector is implemented as an object with its coordinates saved as a NumPy array[18]. It overrides functions and algebraic functions, such as $+$, $-$, $*$, / with scalars or vectors. This simplified programming of vector manipulations such as adding a vector to another or stretching a vector by a scalar.

Walls are stored in a 2D grid. In order to avoid that the computed position is on a wall and therefore invalid, we had to check, whether any part of the step vector touches a wall. For this computation the projected step vector is decomposed into multiple smaller vectors. For this we used the Bresenham line rendering algorithm shown in Algorithm 3.1[3]. Figure 3.3 shows all grid cells touched by a step vector as computed with the Bresenham line rendering algorithm.

The input vector is first transformed to the octant 1 (Figure 3.4). The functions `_input_vector_transformation(octant)`and `_output_vector_transformation(octant)`transform the vector to the octant 1 and transform it back to the original octant. The function then iterates over the x coordinates of the input vector. If it determines that increasing the y coordinate results in a vector closer along the input vector it will increase the y coordinate.

Listing 3.1: Bresenham line rasterization variation

```
def _simple_bresenham(self, vector: Vector) -> List[Vector]:
    """Returns all positions of the boxes if vector is
        rendered with Bresenham
    Args:
```

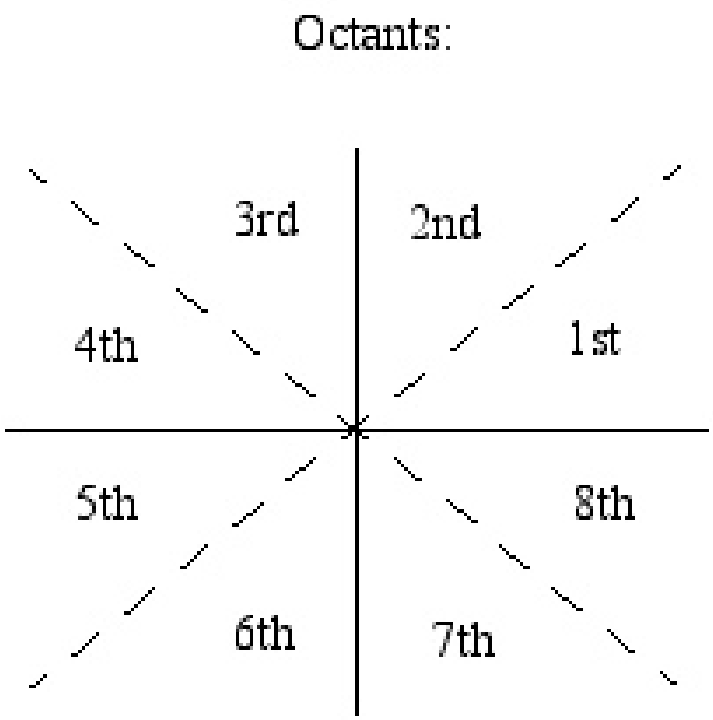

Figure 3.4: The octants of a plane[30].

```
        vector (Vector): The vector from which to compute
            the positions
    Returns: The positions of each box on the path of the
        input vector.
    """
    octant = self._determine_octant(vector)
    output_transfromation = self._output_vector_
        transformation(octant)
    vector = self._input_vector_transformation(octant)(vector
        )
    positions = []  # type: List[Vector]
    d = 2 * vector.y - vector.x  # type: int
    y = 0  # type: int
    for x in range(0, int(vector.x) + 1):
            positions.append(output_transfromation(Vector(x,
                y)))
                    if d > 0:
                            y += 1
                            d -= 2 * vector.x
            d += 2 * vector.y
    return positions
```

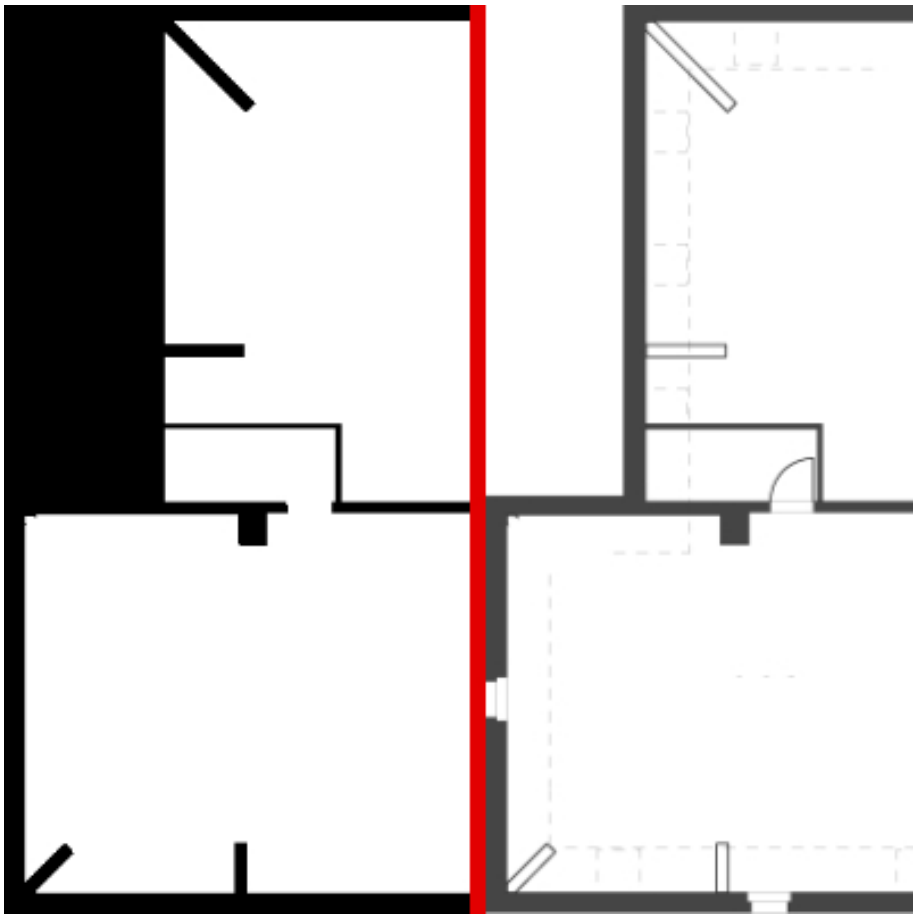

Figure 3.5: Floorplan on the left is used in the algorithm, floorplan on the right is used for display.

3.3.2 *Floorplan Representation*

The floorplan of a map is uploaded to the admin panel in two versions. One version is used to display on the map on the frontend and may contain all helpful features for orientation typically available on a floorplan such as door marks. A second version is a black and white representation of exactly the same dimensions. In this version all positions that are impossible positions for the client such as walls and positions outside the building are black while everything else is white. Figure 3.5 shows a comparison of both versions of the floorplan. This image is then computed to a boolean array which is used to check the validity of a particle.

3.4 DISPLAY CLIENT MAP

In order to show the computed positions of a recorded track or the current position of a device, a map was implemented. This allows to evaluate the particle filter's performance and detect potential prob-

lems. In this section the implementation and features of the client are explained.

3.4.1 *Display client implementation*

There have been a number of technologies and libraries used. These will be presented in this section. In general the map is built for the browser environment.

TYPESCRIPT Typescript is programming language used in the browser to display and animate the map. It is a typed superset of JavaScript that compiles to plain JavaScript[37]. The language is developed by Microsoft and is open source. The disadvantage of the additional compile step required for execution in the browser is more than compensated by potential bugs that can be avoided because of available type information[9].

FABRIC.JS Fabric.js is an HTML5 canvas library. It makes it simple to draw paths, particles, access points and circles on the floorplan[7].

WEBSOCKET The WebSocket protocol[38] was used for transmission of the location and ranging data. It allows for a two-way communication, which allows the server to directly initiate a stream of data without the client having to poll the server, which would be the case with AJAX requests. This reduces the latency and enables near realtime replay of a position[36].

NPM This is short for Node package manager which is used to install the dependencies[25].

WEBPACK Webpack compiles and bundles JavaScript modules. It is mainly used to compile Typescript[39].

CLASS-TRANSFORMER This is a simple library to convert JavaScript plain objects (JSON) to class objects[5].

3.4.2 *WebSocket Communication between Server and Display Client*

Figure 3.6 shows the communication of the map with the server. The process is initialized when the user opens the webpage on *http://<server>:8100/track-id/<id>* with *<server>* being the ip or domain of the deployment server and *<id>* the id of the track that is to be displayed. The server will then deliever the relevant HTML file after which the client renders the website and loads all required dependencies such as javascript files and images. After the browser finished loading, a GET request is sent to the WebSocket server. Note that both servers run on the same server instance, they simply have different

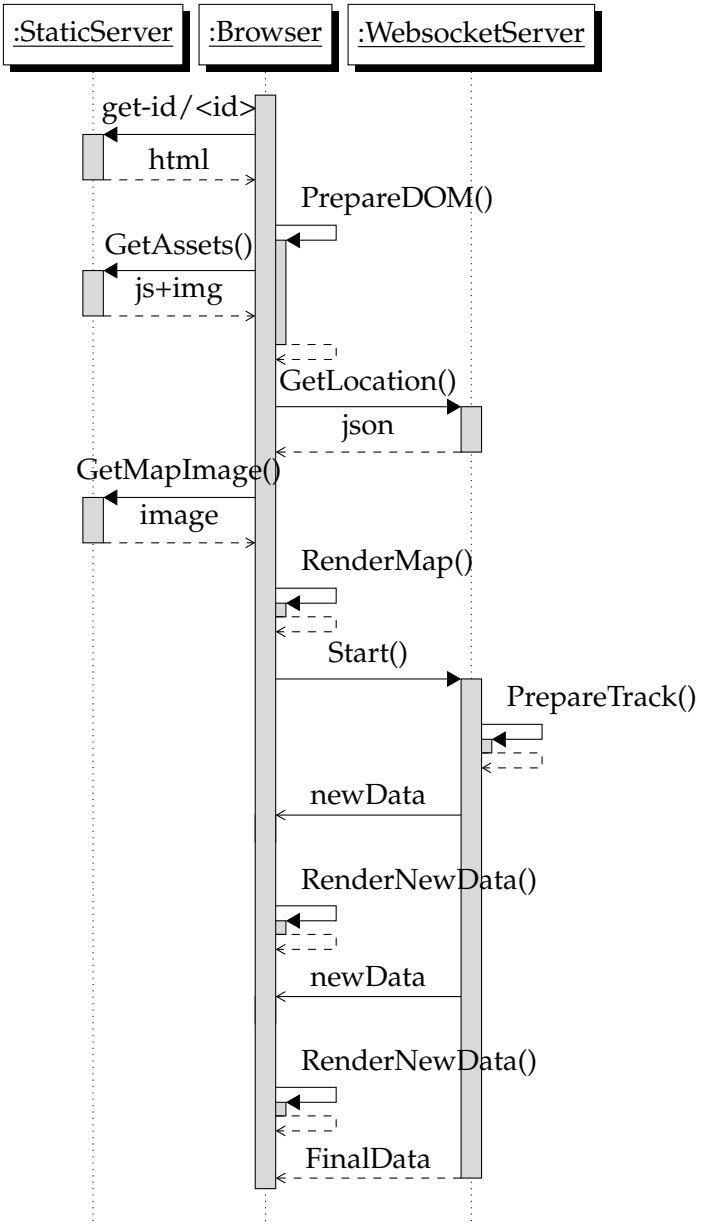

Figure 3.6: WebSocket communication

handlers. The server returns a JSON object with the following four keys:

LOCATION contains an object with information about the floorplan such as the image to be loaded, the map dimensions and scale, gps coordinates as well as the name.

RANGING_DEVICES is an array of all access points on the floor including their identifier and position.

RANGING_LOCATIONS: This array contains the positions used to generate the ranging model.

TRACKING_DEVICES is an array with the devices that are of interest and their initial position. This may be used to render the position of multiple devices but for the current track rendering mechanism only one element is used.

After successful rendering of the inital map a request to the server is made to start the reception of data. The server now sends a continuous stream of JSON data packages containing the following keys:

TIMESTAMP The epoch timestamp of the original measurement.

POSITIONS: An object containing positions to be rendered on the map. The following keys may be provided:

RSSI_TRILATERATED A vector with the position with RSSI trilateration only.

PARTICLE_FILTER is the position computed by the particle filter.

PARTICLES is an array of positions with all particles.

PDR are the vectors computed by the step recognition component.

RANGING An object containing identifiers of access points and their measured distance in pixel and meter.

3.4.3 *Features of the Display Client*

The map can dispaly the following featues of either a recorded track or a live position:

ACCESS POINT A colored dot representing a WiFi access point.

RANGING CIRCLES describe the distance computed by RSSI values of an access point. Radius is animated over time. Color of the stroke corresponds to the color of the access point.

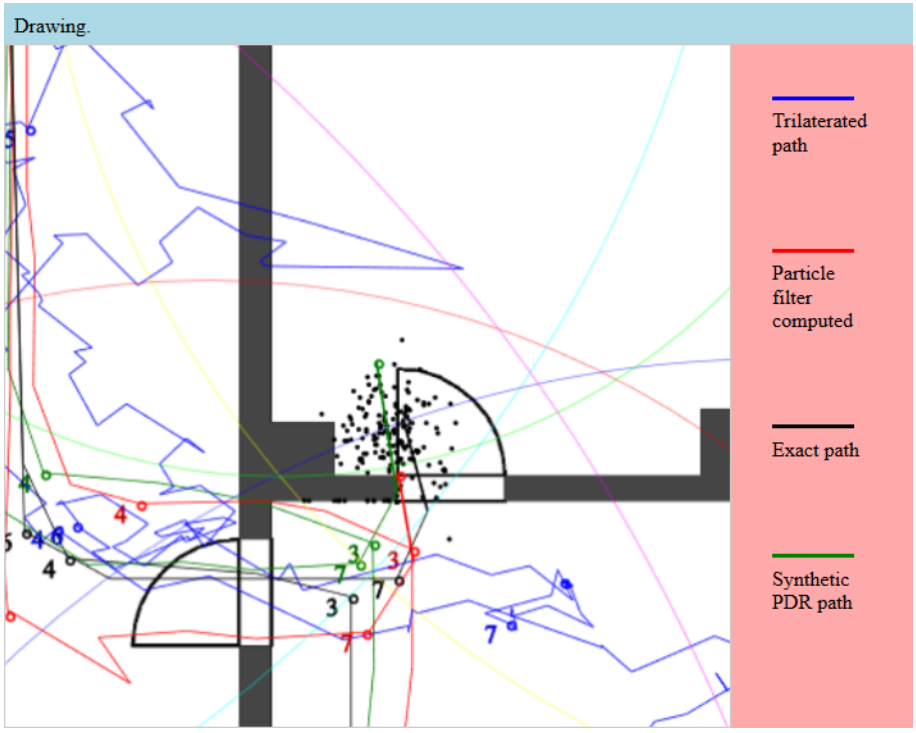

Figure 3.7: Map live drawing showing waypoints, ranging circles and paths (as described in section 3.4.3)

EXACT PATH When a track with waypoints and corresponding numbers was recorded, the path that represents the ground truth is drawn over time.

FLOORPLAN The floorplan as an image on the background.

PARTICLES Particles with the current position and a color intensity corresponding to its weight.

COMPUTED PATH is the path computed with the particle filter.

WALKED PATH is the PDR path as computed with the accelerometer, gyroscope and magnetometer which serve as input to the particle filter.

TRILATERATED PATH: Each time a new ranging value is available a position is computed using trilateration.

WAYPOINTS After a waypoint is passed, a marker with the associated waypoint number is positioned on the map. This allows the user to see and trace the offset of previous positions.

Figure 3.7 shows a path being replayed. The latest path update is fully colored while later sections of the path are more transparent in order to keep the current position visible.

4 EVALUATION

4.1 IOS CLIENT

In this section the performance of iBeacons sampling rate and signal strength is evaluated. For comparison, an ESP32 measuring WiFi signals was used. Details for the ESP32 can be found in [33].

4.1.1 *Experimental setup*

The experiments were conducted in the seminar room, 3rd floor of the Institute of Computer Science at the University of Bern. Six iBeacons were positioned in the room with a maximum distance of 4 meters between them. 15 positions in the room were choosen to create a regression model using the ranging admin panel [33, Section 3.1.2]. The procedure required to take measurements at each position while facing north, easth, south and west direction for 15 seconds. The results were sent to a server running on a laptop using WebSocket. Figure 4.2 shows the iBeacon setup. The iBeacons were set to transmit a paket every 250ms at 0dBm. Thus the iPhone should receive four pakets per iBeacon each second. Apple's SDK limits the number of RSSI measurements to one per second. If more pakets are received more frequently, the RSSI measurements are aggregated into a single measurment. Therefore, it is theoretically possible to see 60 measurements for each iBeacon.

4.1.2 *Presentation of the results*

We analysed our results and summarized them in three graphs. In the graphs each point represents one 60 second sample for one position and one beacon. The x-axis shows the distance between the measuring position and the beacon position. In other experiments similar data was collected using WiFi access points [33]. In their experiemnts data was captured at 48 locations with an ESP32 using the same procedure described previously (Figure 4.1). We include these results for comparison.

- Figure 4.3 shows the number of captured measurements during the 60 seconds capture interval.
- Figure 4.4 shows the average signal strength of the iBeacons
- Figure 4.5 shows the standard deviation of the iBeacon RSSI

Figure 4.1: Floorplan shows the ranging experiment setup with WiFi access points.

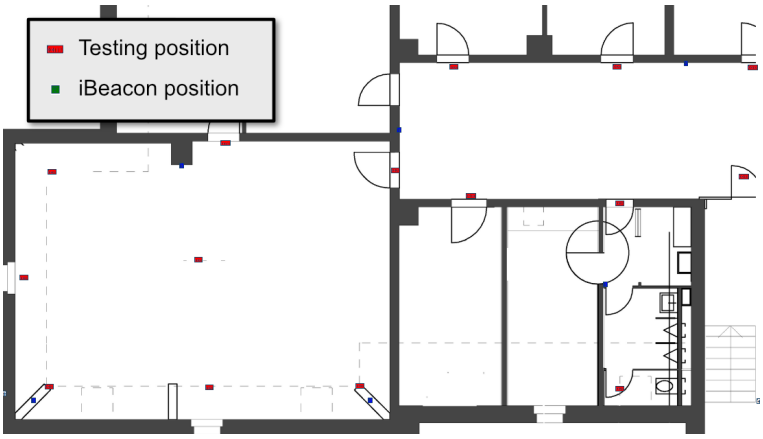

Figure 4.2: Floorplan shows the ranging experiment setup with iBeacon.

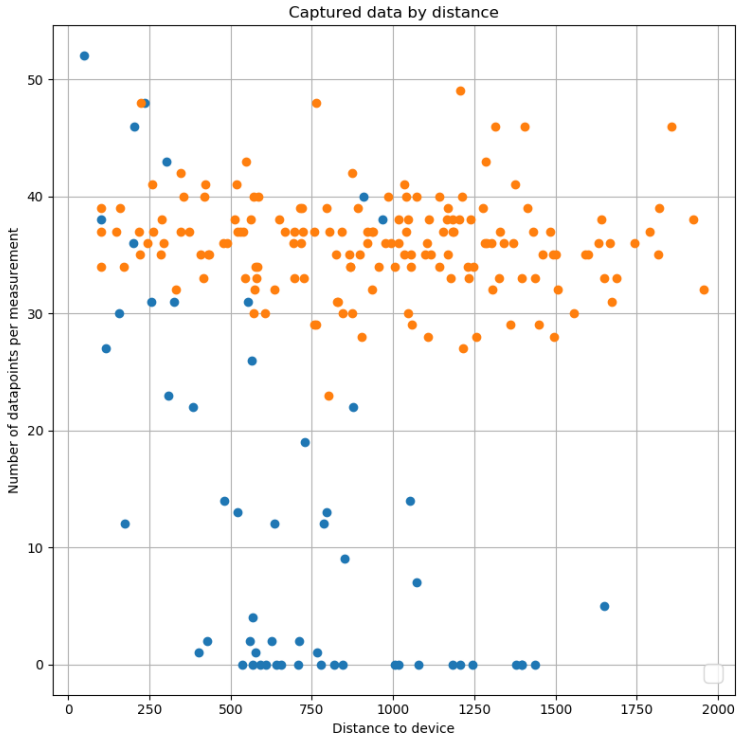

Figure 4.3: Number of datapoints per measurmenet by distance to device (cm). Blue dots show the iBeacon measurements, the orange dots show the ranging results of ESP32 tests.

4.1.3 *Discussion of the results*

On the first figure (Figure 4.3) we notice that many measurements led to zero or a very low number of captured datapoints. There appears to be a weak, negative correlation between distance and number of captured datapoints. By comparison, in the experiments with the ESP32 and WiFi access points the number of captured datapoins were consistently around 30 to 40, even when the distance increased.

When evaluating the signal strength (Figure 4.4), we notice a number of results with 0 dBM. This is due to the Apple SDK which returns a RSSI value of 0 when no pakets are received. These values are inconsistent with the results of the ESP32 experiments, where a clear, negative correlation between RSSI and distance is recognized.

4.2 PATRICLE FILTER

In this section the evaluation of the particle filter algorithm with an ESP32 client is presented.

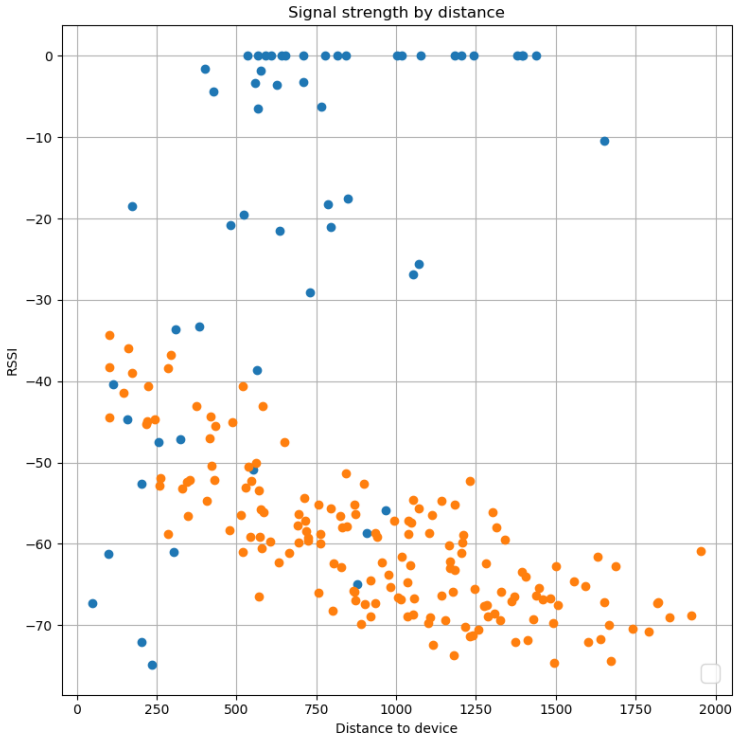

Figure 4.4: Average iBeacon signal strength by distance (cm). Blue dots show the iBeacon measurements, orange dots show the results of ranging tests with an ESP32.

4.2.1 *Experimental setup*

The following sections describe the detailed procedure of the experiments.

4.2.1.1 *Path recording*

The experiments were conducted in an office area at the Institute of Computer Science at the University of Bern. The projected path spans over 7 rooms. A total of 6 WiFi access points were used for testing. The testing device was an ESP32 client that has been previously used and trained at the same location. The test setup involved a laptop on which the WebSocket server was running and a smartphone that provided a WiFi hotspot for the two devices to communicate. Once the server was running, one person walked a defined path holding the ESP32 slightly above waist height. A second person carrying the laptop followed the person to ensure maximum connectivity between the ESP32 and the server. This experiment was repeated three times. The path consists of 15 waypoints. Figure 4.6 shows the path and the 6 WiFi access points as colored dots. Each time a waypoint was passed, a physical button on the ESP32 was pressed. This incremented a variable corresponding to the last passed waypoint, which is included in the data sent to the

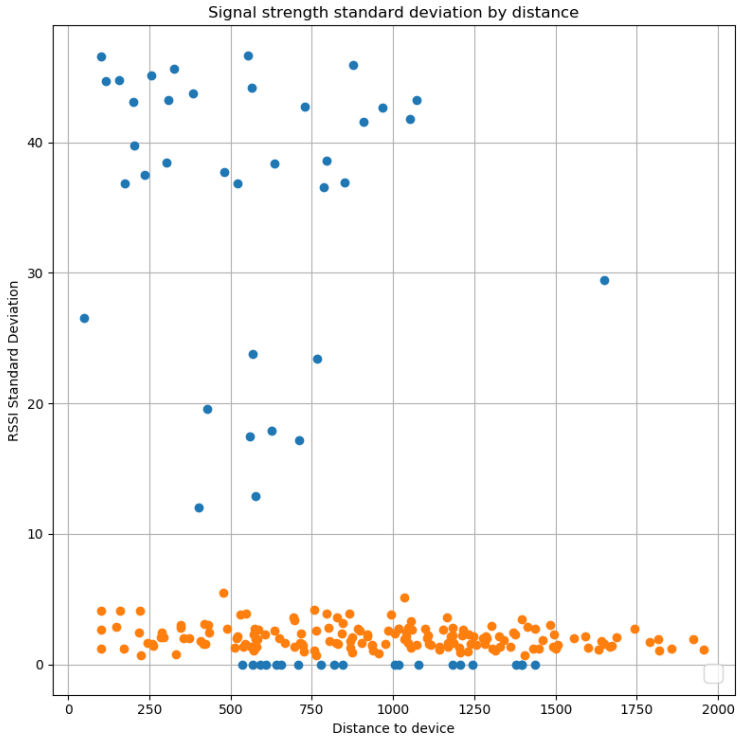

Figure 4.5: iBeacon signal strength standard deviation by distance (cm). Blue dots show the iBeacon measurements, orange dots show the results of ranging tests with an ESP32. Best viewed in color.

server. Later a change of the counter variable is used to recognize that the ESP32 has passed a waypoint.

4.2.1.2 *Data annotation*

Through the admin interface the positions of the waypoints were added. With the recording of the last passed waypoint (through pressing a button), the ground truth is known for all waypoints. Figure 4.7 shows the admin interface, where waypoints (ground truth positions) can be added. The flag can be dragged and dropped onto a position on the map. The number on the dropped marker corresponds to the recorded waypoints.

4.2.1.3 *Particle Filter Parameters*

The used ranging model has an average error of 1.07m and a standard deviation of 1.33m. Initially, it was intended to compute the Pedestrian Dead Recognitioning (PDR) path using contributions of [22]. However, it appeared to be impossible to use this module for with data recorded with the ESP32 instead of the Android phone it was tested with. Developing our own PDR module was out of scope for this thesis. Therefore we decided to synthetically generate the PDR path. By using synthetically generated paths it is possible to examine the effect of the

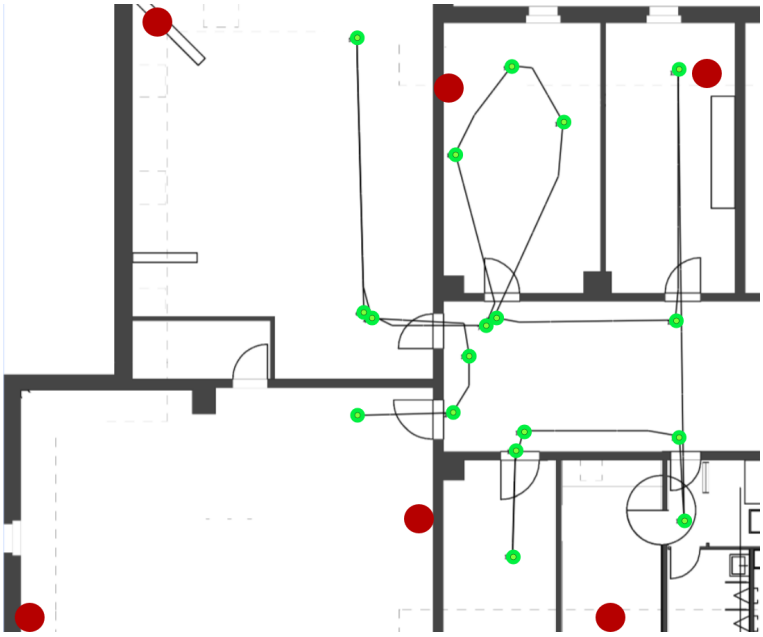

Figure 4.6: Base path

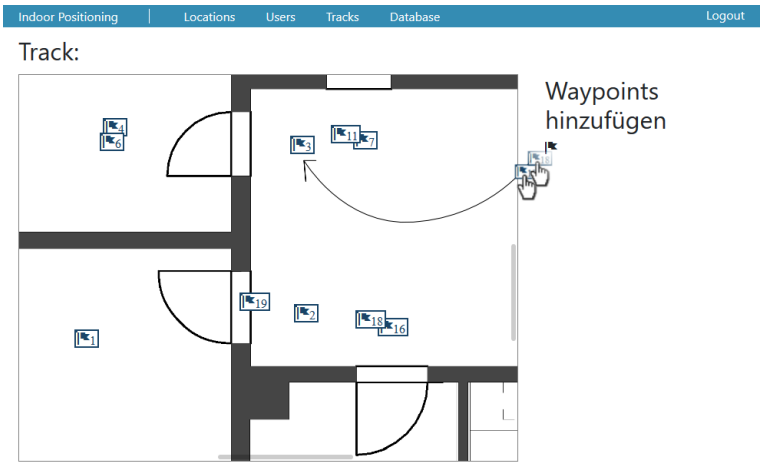

Figure 4.7: Waypoints interface

PDR path error on the localization performance of the algorithm. To generate the synthetic path we used the following procedure:

- We split the ground truth into 0.7m sections which are the estimated step the person holding the ESP32 did.
- We rotated each estimated step vector with $r \sim N(\alpha, \beta)$
- We stretched each estimated step vector by multiplying the vector with $s \sim N(l, s)$

Table 4.1 shows the properties of the tested PDR paths. These PDR paths have been generated for all three recorded paths (containing RSSI measurements) resulting in a total of 18 paths available for particle filter evaluation.

Properties	Track 1	2	3	4	5	6
Step length factor offset l	1	1	1	1	1	1
Step length factor variance s	0.2	0.2	0.2	0.2	0.2	0.2
Step rotation offset α	1°	2°	3°	4°	5°	6°
Step rotation variance β	5°	10°	15°	20°	25°	30°

Table 4.1: Tested PDR paths

Figure 4.8 shows the generated paths on the floorplan. The error is computed by summing the Euler distance of each waypoint In this figure we can see that the total PDR increases the longer the path is. Since the particle filter is not deterministic due to randomness in the systematic resampling step (see 2.2.1), 5 particle filter paths have been generated.

4.2.2 *Presentation of the results*

Figure 4.9 shows the average error of the particle filter and the PDR path by number of particle. Each point represents the average error over 15 tracks.

Figure 4.10 shows the average error for the last waypoint only.

Figure 4.11 shows the average error at each waypoint.

4.2.3 *Particle Filter evaluation results*

We can not observe any improvement of the error by increasing the number of particles. Fluctations are attributed to the randomness in the particle filter algorithm. PDR paths with a higher deviation tend to perform worse than the PDR paths with a lower deviation. The total error of the particle filter paths becomes higher than the PDR path error once the PDR becomes more exact.

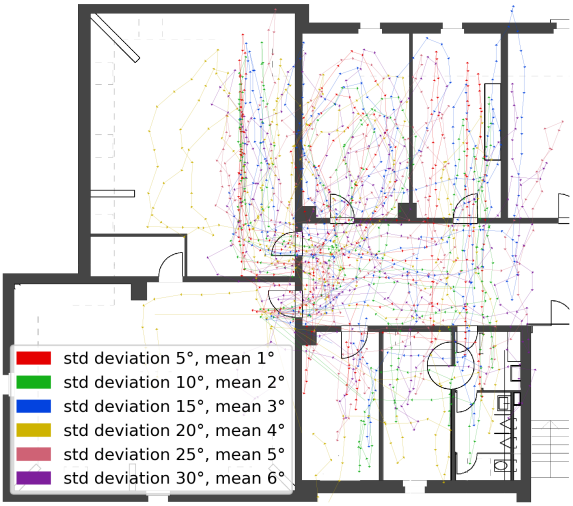

Figure 4.8: Generated PDR paths. Best viewed in color.

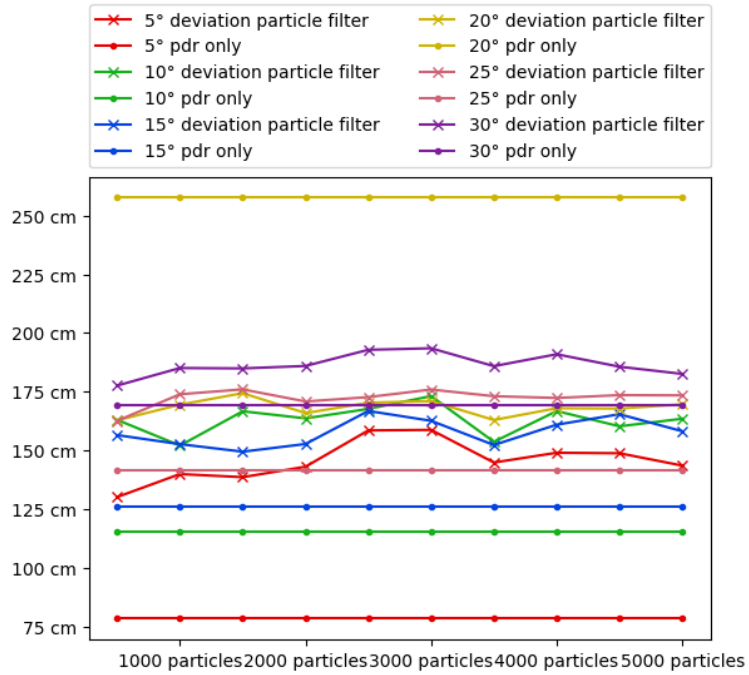

Figure 4.9: Total average error by number of particles. Best viewed in color.

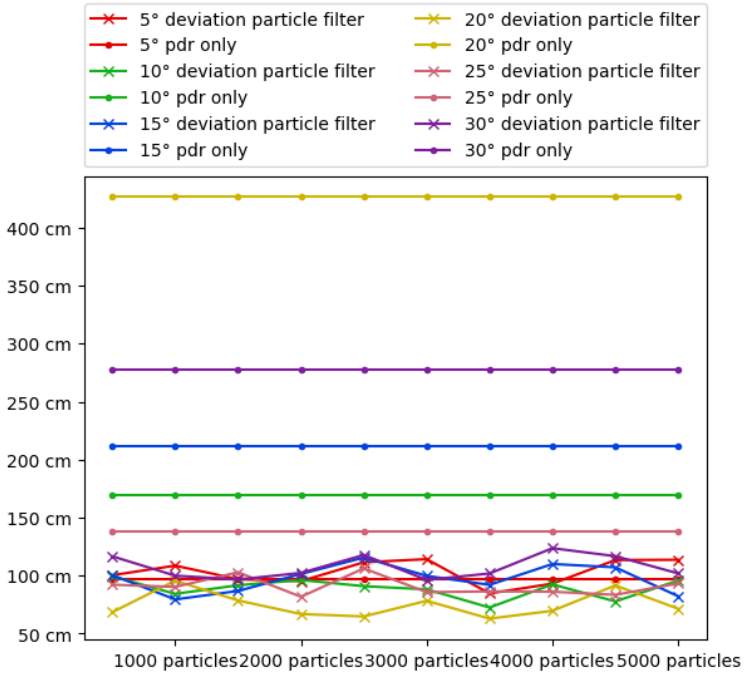

Figure 4.10: Average error at last waypoint by number of particles. Best viewed in color.

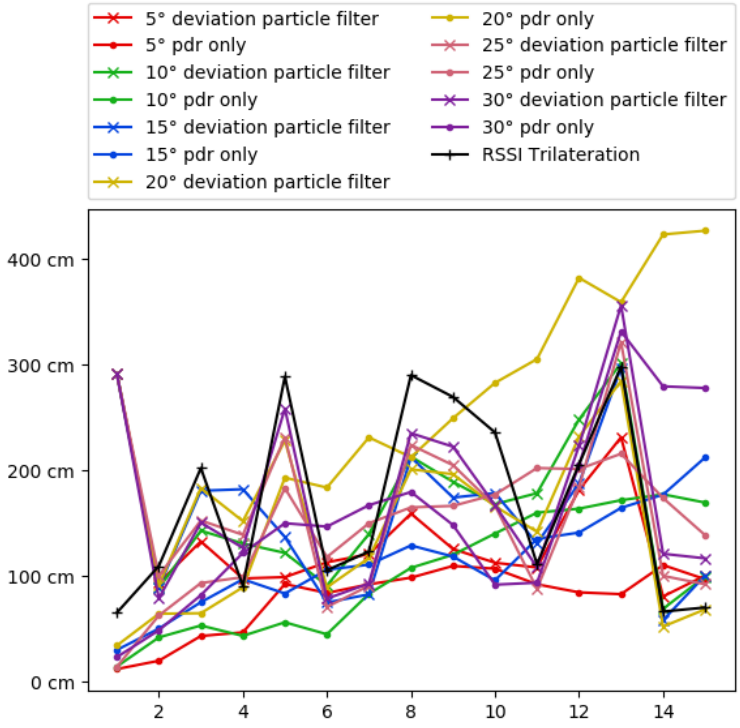

Figure 4.11: Average error by waypoint. Best viewed in color.

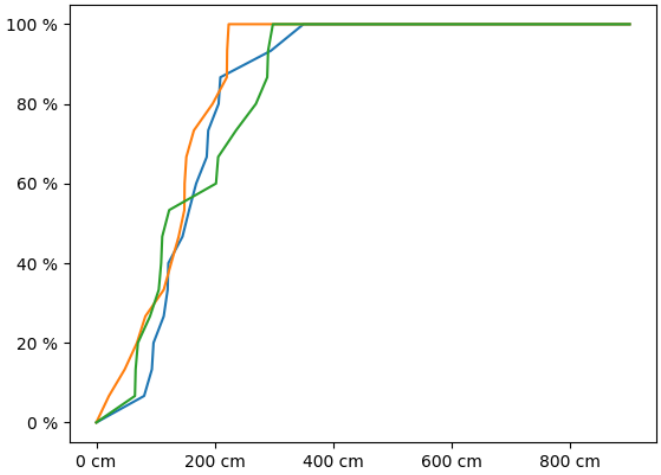

Figure 4.12: Localization CDF. The green line shows the particle filter, blue line shows the triangulated, the orange path shows the PDR perfomance. The average of all tracks is used. Best viewed in color.

For the average error of the last waypoint only the particle filter error is - apart from the exactest PDR path - consistently lower than the corresponding PDR path. In this case we can also observe that increasing the number of particles does not show any effect.

In Figure 4.11 on page 31 we observe that the PDR error is more precise in the beginning of the path and tends to decrease as the track progresses. The particle filter track is, in most cases, performing better than RSSI trilateration. The accuracy of the of the PDR path has only little impact on the particle filter error.

5 CONCLUSION

5.1 SUMMARY

In this work, we presented and evaluated three components of a client-server based localization system.

We implemented an iOS client for iBeacon signal capture and sending data to a server via a Websocket connection. We evaluated the performance of the client. We found that the performance in the testing environment was inadequate for localization tasks.

Further, we implemented a particle filter implementation to fuse ranging data and PDR path for an improved localization. The particle filter improves the positioning when compared to PDR . When compared to RSSI trilateration the localization can be slightly improved in cases where the trilateration with RSSI shows large fluctuation in a short time frame.

Additionally, we provide an implementation for a floorplan map. It is capable of showing a live representation of the particle filter state, PDR data and ranging data. With our extension to the admin interface we provide a simple way to annotate experiment data with ground truth.

5.2 FUTURE WORK

In future work other BLE protocols for proximity beacons such as Google Eddystone[11] may be implemented. Further tests with iBeacon may be carried out using other devices such as Android phones or integrated platforms. A number of promising work with iBeacon's has already been conducted [20, 23, 35].

Further, the multi dimensional particle filter approach of 2018 could be implemented into this system.

BIBLIOGRAPHY

[1] Sanjeev Arulampalam and Branko Risti. "Comparison of the particle filter with range-parameterized and modified polar EKFs for angle-only tracking." In: *Proc. SPIE* 4048 (July 2000), pp. 288–299. DOI: 10.1117/12.391985 (cit. on p. 7).

[2] *Bluetooth Specification version 4.0*. 30. The Bluetooth Special Interest Group. June 2010. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/archived-specifications (cit. on p. 5).

[3] *Bresenham's Line Algorithm*. In: *Wikipedia*. Page Version ID: 864354932. Oct. 16, 2018. URL: https://en.wikipedia.org/w/index.php?title=Bresenham%27s_line_algorithm&oldid=864354932 (visited on 01/20/2019) (cit. on p. 15).

[4] José Luis Carrera, Zan Li, Zhongliang Zhao, Torsten Braun, and Augusto Neto. "A Real-time Indoor Tracking System in Smartphones." In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems - MSWiM '16*. ACM Press, 2016. DOI: 10.1145/2988287.2989142 (cit. on p. 1).

[5] *Class-Transformer: Proper Decorator-Based Transformation / Serialization / Deserialization of Plain Javascript Objects to Class Constructors*. typestack, Aug. 11, 2018. URL: https://github.com/typestack/class-transformer (visited on 08/13/2018) (cit. on p. 18).

[6] *Cocoa Touch - Accessing iPhone WiFi Information via SDK*. URL: https://stackoverflow.com/questions/351954/accessing-iphone-wifi-information-via-sdk (visited on 10/17/2018) (cit. on p. 11).

[7] *Fabric.Js Javascript Canvas Library*. URL: http://fabricjs.com/ (visited on 08/13/2018) (cit. on p. 18).

[8] Yuan Gao, Qingxuan Yang, Guanfeng Li, Edward Y. Chang, Dong Wang, Chengu Wang, Hang Qu, Pei Dong, and Faen Zhang. "XINS." In: *Proceedings of the 1st international workshop on Mobile location-based service - MLBS '11*. ACM Press, 2011. DOI: 10.1145/2025876.2025884 (cit. on p. 9).

[9] Zheng Gao, Christian Bird, and Earl T. Barr. "To Type or Not to Type: Quantifying Detectable Bugs in JavaScript." In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, May 2017. DOI: 10.1109/icse.2017.75 (cit. on p. 18).

[10] "Getting Started with iBeacon." In: (2014), p. 11. URL: https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf (cit. on p. 1).

[11] Google. *Specification for Eddystone, an open beacon format from Google*. Jan. 2019. URL: https://github.com/google/eddystone (cit. on p. 33).

[12] *Google Code Archive - Long-Term Storage for Google Code Project Hosting*. URL: https://code.google.com/archive/p/iphone-wireless/ (visited on 10/17/2018) (cit. on p. 11).

[13] *iBeacon - Apple Developer*. URL: https://developer.apple.com/ibeacon/ (visited on 10/18/2018) (cit. on p. 5).

[14] *Ios5 - iOS Wifi Scan, Signal Strength*. URL: https://stackoverflow.com/questions/12054818/ios-wifi-scan-signal-strength (visited on 10/17/2018) (cit. on p. 11).

[15] *Iphone - Accessing & Using the MobileWiFi.Framework*. URL: https://stackoverflow.com/questions/2018110/accessing-using-the-mobilewifi-framework (visited on 10/17/2018) (cit. on p. 11).

[16] *Iphone - How to Use iOS CNCopyCurrentNetworkInfo to Read WIFI Signal Strength (RSSI)?* URL: https://stackoverflow.com/questions/28999064/how-to-use-ios-cncopycurrentnetworkinfo-to-read-wifi-signal-strength-rssi?s=7%7C131.2349 (visited on 10/17/2018) (cit. on p. 11).

[17] *iPhone Signal Strength*. URL: https://stackoverflow.com/questions/2959567/iphone-signal-strength (visited on 10/17/2018) (cit. on p. 11).

[18] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed <today>]. 2001–. URL: http://www.scipy.org/ (cit. on p. 15).

[19] Reda A. El-Khoribi, Haitham S. Hamza, and M. A. Hammad. "Indoor localization and tracking using posterior state distribution of hidden markov model." In: *2013 8th International Conference on Communications and Networking in China (CHINACOM)*. IEEE, Aug. 2013. DOI: 10.1109/chinacom.2013.6694657 (cit. on p. 9).

[20] Xiangjie Li, Dan Xu, Xuzhi Wang, and Rizwan Muhammad. "Design and implementation of indoor positioning system based on iBeacon." In: *2016 International Conference on Audio, Language and Image Processing (ICALIP)*. IEEE, July 2016. DOI: 10.1109/icalip.2016.7846648 (cit. on pp. 6, 33).

[21] Chi-Chung Lo, Ting-Hui Chiang, Tsu-Kuang Lee, Ling-Jyh Chen, and Yu-Chee Tseng. "Wireless location tracking by a sensor-assisted particle filter and floor plans in a 2.5-D space." In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, Apr. 2018. DOI: 10.1109/wcnc.2018.8377214 (cit. on p. 10).

[22] Lucien Madl. "Attitude Heading Reference System and Step Recognition for Cloud-Based Indoor Positioning – Android Client." Cand. thesis. University of Bern, Sept. 2018 (cit. on pp. 1, 3, 15, 27).

[23] Paul Martin, Bo-Jhang Ho, Nicholas Grupen, Samuel Muñoz, and Mani Srivastava. "An iBeacon primer for indoor localization." In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings - BuildSys '14*. ACM Press, 2014. DOI: 10.1145/2674061.2675028 (cit. on pp. 6, 33).

[24] Nick. *The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs (2015)*. URL: https://www.aislelabs.com/reports/beacon-guide/ (visited on 10/18/2018) (cit. on p. 13).

[25] *Npm*. URL: https://www.npmjs.com/ (visited on 08/13/2018) (cit. on p. 18).

[26] *Objective c - How to Get WIFI Signal Strength in dBm in iOS Programmatically*. URL: https://stackoverflow.com/questions/42061307/how-to-get-wifi-signal-strength-in-dbm-in-ios-programmatically?s=6%7C131.2381 (visited on 10/17/2018) (cit. on p. 11).

[27] *Objective c - Is It Possible to Get Wifi Signal Strength in Ios 9*. URL: https://stackoverflow.com/questions/32970711/is-it-possible-to-get-wifi-signal-strength-in-ios-9?s=2%7C154.6053 (visited on 10/17/2018) (cit. on p. 11).

[28] *Part 1: The Origins of GPS, and the Pioneers Who Launched the System*. May 2, 2010. URL: http://gpsworld.com/origins-gps-part-1/ (visited on 08/05/2018) (cit. on p. 1).

[29] Ling Pei, Donghui Liu, Danping Zou, Ronald Lee Fook Choy, Yuwei Chen, and Zhe He. "Optimal Heading Estimation Based Multidimensional Particle Filter for Pedestrian Indoor Positioning." In: *IEEE Access* 6 (2018), pp. 49705–49720. DOI: 10.1109/access.2018.2868792 (cit. on pp. 10, 33).

[30] *python - Determining which octant*. URL: https://codereview.stackexchange.com/questions/95550/determining-which-octant-has-a-specific-point (cit. on p. 16).

[31] Ritambhara P. Rajeshirke and Manisha R. Dhage. "Inertial Sensor Based Localization Using Wi-Fi In Complex Indoor Environment." In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. IEEE, Sept. 2017. DOI: 10.1109/ctceec.2017.8455074 (cit. on p. 10).

[32] Cyrill Stachniss. *Robot Mapping - Short Introduction to Particle Filters and Monte Carlo Localization*. 2013. URL: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam11-particle-filter.pdf.

[33] Stefan Serena. "Cloud-Based Indoor Positioning - ESP32 Client." Cand. thesis. University of Bern, Sept. 2018 (cit. on pp. 1, 3, 11, 23).

[34] *Stochastic Universal Sampling*. In: *Wikipedia*. Page Version ID: 858218481. Sept. 5, 2018. URL: https://en.wikipedia.org/w/index.php?title=Stochastic_universal_sampling&oldid=858218481 (visited on 10/09/2018) (cit. on p. 9).

[35] Kwangjae Sung, Dong Kyu 'Roy' Lee, and Hwangnam Kim. "Indoor Pedestrian Localization Using iBeacon and Improved Kalman Filter." In: *Sensors* 18.6 (2018). DOI: 10.3390/s18061722. URL: http://www.mdpi.com/1424-8220/18/6/1722 (cit. on pp. 9, 33).

[36] *The WebSocket API*. URL: https://www.w3.org/TR/websockets/ (visited on 08/13/2018) (cit. on p. 18).

[37] *TypeScript - JavaScript That Scales*. URL: http://www.typescriptlang.org/ (visited on 08/13/2018) (cit. on p. 18).

[38] W3C. *HTML Standard*. URL: https://html.spec.whatwg.org/multipage/web-sockets.html (cit. on p. 18).

[39] *Webpack*. URL: https://webpack.js.org/ (visited on 08/13/2018) (cit. on p. 18).

[40] *What's New in Core Location - WWDC 2013 - Videos*. URL: developer.apple.com/videos/play/wwdc2013/307/ (visited on 08/05/2018) (cit. on p. 5).

[41] Zhongliang Zhao, Stephane Kuendig, Jose Carrera, Blaise Carron, Torsten Braun, and Jose Rolim. "Indoor Location for Smart Environments with Wireless Sensor and Actuator Networks." In: *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. IEEE, Oct. 2017. DOI: 10.1109/lcn.2017.65 (cit. on pp. 9, 11).

[42] Xingfu Zhong, Wenming Wang, and Quanyu Wang. "An indoor AR registration technique based on iBeacons." In: *2017 IEEE International Conference on Information and Automation (ICIA)*. IEEE, July 2017. DOI: 10.1109/icinfa.2017.8079065 (cit. on p. 6).