
# Particle Filtering with Geospatial Analysis for Indoor Positioning

by

Dorian Harder

Thesis for the degree of
Master of Science
(Geodesy and Geoinformatics)
HafenCity University Hamburg
01.09.2021

Examiners:

Prof. Dr.-Ing. Harald Sternberg,
Msc. Hossein Shoushtari

Dorian Harder
dorian.harder@hcu-hamburg.de
Student ID:  6067534

# DEDICATION

Many people have supported me in numerous ways throughout my time at the HafenCity University and especially while I was writing this thesis. First of all I want to thank my fellow students for their company especially through the events of the last year. Further I want to thank all my friends and family for supporting me. I am especially thankful for Niko Khaled, who provided much inspiration and inside into programming languages and all things regarding information technology and who was allways available for deep and interesting conversations that I always enjoy. Further I want to thank the examiners Prof. Dr.-Ing. Sternberg and Hossein Shoushtari for the possibility to write this thesis and for assessing it. Special thanks go to Hossein, who motivated me to work in the topic of this thesis and who always supports me in doing my best.

Hamburg, 31.08.2021
Dorian Harder

HafenCity Universität Hamburg

# EIDESSTATTLICHE ERKLÄRUNG

**Diese Erklärung ist der Thesis beizufügen!**

| | |
|---|---|
| **Name:** | Harder |
| **Vorname:** | Dorian |
| **Matrikelnummer:** | 6067534 |
| **Studienprogramm:** | Geodäsie und Geoinformatik, Master |

Ich versichere, dass ich die vorliegende Thesis mit dem Titel

Particle Filtering with Geospatial Analysis for Indoor Positioning

selbstständig und ohne unzulässige fremde Hilfe erbracht habe.

Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Im Falle einer Gruppenarbeit bezieht sich die Erklärung auf den von mir erarbeiteten Teil der Thesis.

Hamburg, 01.09.2021

Ort und Datum

Unterschrift der/des Studierenden

**VOM PRÜFUNGSAMT AUSZUFÜLLEN**

Die o.g. Thesis wurde abgegeben am

**Eingangsstempel Infothek**
**Studierendenverwaltung | Prüfungsamt**

# TABLE OF CONTENTS

# LIST OF FIGURES

FIGURE

# LIST OF ALGORITHMS

# LIST OF ACRONYMS

**5G** Fifth-Generation of Mobile Telecommunications Technology

**CAD** Computer Aided Design software

**CDF** Cumulative Distribution Function

***cl*** check for line of sight

***cm*** check for room

***cr*** check for routing

**CSV** Comma Separated Values

**GIS** Geographic Information System

**GNSS** Global Navigation Satellite System

**HCU** HafenCity University

**IMU** Inertial Measurement Unit

**KF** Kalman Filter

**LIDAR** Light Detection and Ranging

**LOS** Line of Sight

**MARG** Magnetic, Angular Rate and Gravity

**PDF** Probability Density Function

**PDR** Pedestrian Dead Reckoning

**PF** Particle Filter

**RF** Radio Frequency

**RSSI** Received Signal Strength Indication

**UWB** Ultra Wide Band

**WLAN** Wireless Local Area Networks

***wm*** weighting by rooms

***wl*** weighting by line of sight

***wr*** weighting by routing

***ww*** weighting by walls

# ABSTRACT

Modern smartphones enable a wide range of people the use of location based services, such as personal navigation, in everyday life. The localisation in outdoor scenarios with smartphones is usually based on Global Navigation Satellite System (GNSS). However, the positioning with GNSS fails to provide sufficient and accurate measurements in an indoor environment. To enable the usage of location based services, for example the navigation at exhibition or airport areas or large university buildings, other indoor positioning approaches have been researched. Most of those can't provide the needed accuracy or are expensive to install and maintain. A possible alternative is the usage of the Inertial Measurement Unit (IMU) of the smartphone using a pedestrian dead reckoning system Pedestrian Dead Reckoning (PDR). To correct the drift of the position estimate that results from noisy measurements, corrections through additional information is necessary. For this reason, two Particle Filter (PF) methods, a simple bootstrap PF and a backtracking PF, have been developed in this thesis. They both use building information from floor plans and routing edges, to improve the position estimate. The PF uses weighted particles to represent the probability density of the position estimate. The backtracking PF further stores the propagation history that can be used for further improvements of the position estimate. One novelty of the developed PF methods is the usage of geospatial analysis tools to derive information about the spatial relation between the particles and the geometries from the building information. The implementation of geospatial analysis tools further enables the use of geodata, which can be derived from building plans in Computer Aided Design software (CAD) format. The developed PF methods have been tested and compared with different methods for the weighting of particles regarding their positioning accuracy on two different test paths through the HafenCity University (HCU)-building in Hamburg. It was possible to achieve a position error of less than 3 m for a path through narrow corridors and less than 5.5 m for a path including wider spaces, such as hallways, 90 % of the time for some of the tested methods.

# KURZFASSUNG

Moderne Smartphones ermöglichen einem breiten Personenkreis die Nutzung ortsbezogener Dienste, wie zum Beispiel die persönliche Navigation, im Alltag. Die Lokalisierung mit Smartphones im Außenbereich basiert in der Regel auf Global Navigation Satellite System (GNSS). Ausreichend präzise Messungen in Innenräumen sind mit GNSS jedoch in der regel nicht möglich. Um die Nutzung ortsbasierter Dienste in Innenräumen zu ermöglichen, beispielsweise die Navigation auf Messe- oder Flughafengeländen oder in großen Universitätsgebäuden, wurden andere Indoor-Positionierungsansätze erforscht. Die meisten von ihnen können nicht die erforderliche Genauigkeit bieten oder erfordern einen hohen Aufwand in der Installation und Wartung. Eine mögliche Alternative ist die Nutzung der Inertial Measurement Unit (IMU) des Smartphones mittels eines Pedestrian-Dead-Reckoning-Systems Pedestrian Dead Reckoning (PDR). Um die Drift der Positionsschätzung, die aus verrauschten Messungen resultiert, zu korrigieren, sind Korrekturen durch zusätzliche Informationen notwendig. Aus diesem Grund wurden in dieser Arbeit zwei Particle Filter (PF)-Methoden entwickelt, ein einfacher Bootstrap-PF und ein Backtracking-PF. Beide verwenden Gebäudeinformationen aus Grundrissen und Routing-Kanten, um die Positionsschätzung zu verbessern. Der PF verwendet gewichtete Partikel, um die Wahrscheinlichkeitsdichte der Positionsschätzung zu repräsentieren. Der Backtracking-PF speichert außerdem den bisherigen Verlauf der Fortbewegung, welcher für weitere Verbesserungen der Positionsschätzung verwendet werden kann. Eine Neuheit der entwickelten PF-Methoden ist der Einsatz von Geodatenanalysen, um Informationen über die räumliche Beziehung zwischen den Partikeln und den Gebäudedaten abzuleiten. Die implementierung der Geodatenanalysen ermöglicht darüber hinaus die Nutzung von Geodaten, die sich aus Bauplänen im Computer Aided Design software (CAD)-Format ableiten lassen. Die entwickelten PF-Methoden wurden auf zwei verschiedenen Testpfaden durch das HafenCity University (HCU)-Gebäude in Hamburg getestet und mit verschiedenen Methoden zur Gewichtung der Partikel hinsichtlich ihrer Positionierungsgenauigkeit verglichen. Bei einigen der getesteten Verfahren konnte in 90 % der Fälle ein Positionsfehler von weniger als 3 m für einen Weg durch enge Korridore und von weniger als 5,5 m für einen Weg mit breiteren Fluren erreicht werden.

# 1 Introduction

Nowadays the usage of smartphones for navigation, based on the global navigation satellite system Global Navigation Satellite System (GNSS), is common in everyday life, especially for the personal navigation. GNSS is the dominant technology for outdoor localization, but due to heavy attenuation and reflection of the GNSS signals by building structures, a sufficient availability and accuracy can not be provided indoors. Location based services, including positioning and navigation, can be useful in indoor scenarios, too. They can for example be used for the navigation at an exhibition area, in shopping malls, universities, office buildings, airports or train stations. To enable the mentioned applications of indoor positioning, the development and research of GNSS independent indoor navigation systems is neccessary.

Other GNSS independent localization approaches have already been explored, including ultra wideband Ultra Wide Band (UWB), wireless local area networks Wireless Local Area Networks (WLAN) [1], [2], magnetometer [3], vision [4] or ultrasound [5], among others. But most of them can't provide a needed meter-level accuracy, that is required to determine for example the current room, that a user is in. Furthermore, infrastructure assisted approaches, especially WLAN and UWB usually lead to high maintenance and implementation costs and efforts, which must be expended by the individual facility management [6].

Many approaches use the inertial sensors, also called Inertial Measurement Unit (IMU), to track a user's position by continuously estimating the displacement from a known location. This pedestrian dead reckoning PDR [7], [8], [9], [10] is not dependent on infrastructure assistance. However, the IMU's measurements are noisy, resulting in an accumulating positional drift over time. Therefore it can only provide reliable position estimation for a limited time, making additional support necessary. Several approaches exist, such as a visual support approach using vision features [6], [11]. Other approaches use point clouds [12], [9] from cameras as well as Light Detection and Ranging (LIDAR) sensors, that can be combined with the odometer information to correct the drift and to provide more accurate localization estimations. These features are not always available since vision sensors can't be used all the time. State-of-the-art odometry based learning approaches [8], [13], are an alternative, but they rely on label data that is laborious to attain.

PDR-based localization with map matching support poses an alternative to the mentioned methods. If building information in the form of floor plans and/or routing graphs are available, the information about the environment can be combined with the IMU measurements to achieve a more accurate position estimation. For this task, Particle Filter (PF), which belong to the Monte

Carlo algorithms, can be implemented. PF are well suited for processing the noisy data of the smartphone sensors and the information derived from the building's geometry. PF algorithms enable the consideration of inaccuracies in the measurements for the position, by using weighted particles as representation of the probability density distribution of possible position estimates [14]. Different PF algorithms exist, from a simple bootstrap PF to a backtracking PF, that enables the correction and re-estimation of implausible paths. For this reason, two PF algorithms, a bootstrap PF and a PF with backtracking, have been developed for this thesis. Both PF algorithms are developed on the base of existing approaches and have been modified to use geospatial analysis to deal with spatial information. The usage of geospatial analysis tools enables the use of building information in GeoJSON format. It can comparatively easily be derived from digital plans, typically produced in Computer Aided Design software (CAD) environments. Furthermore, the usage of geodata packages to query for spatial information (such as query for intersections) make the implementation more robust and easier.

The bootstrap PF and the PF with backtracking functionality are tested with pre-calculated step length and step heading values from two paths through the HafenCity University (HCU) building in Hamburg, Germany. The two PF approaches have been compared regarding their accuracy,. For both PF algorithms different weighting methods for the particles have been tested and compared. The goal is to find the best approach for a PF algorithm for indoor navigation with map matching, providing an comparatively easy handling of the needed data as well as a sufficient accuracy, precision and robustness.

The remainder of this thesis is structured as follows: related literature is reviewed in the next chapter; chapter three describes the underlying methodology with examples from related works; in chapter four the implementation is explained and the used data set is presented; section five presents and discusses the results of the two PF-based map matching algorithms and finally, chapter six provides the conclusions and an outlook for future development.

# 2 Literature Review

Different methods for positioning in indoor environments with the use of smartphones have been researched and developed, which can provide different levels of localisation precision and accuracy. The available indoor navigation systems can be classified into infrastructure supported, autonomous approaches (based on integrated sensors) and hybrid methods, which combine integrated sensors and infrastructure support.

Positioning systems that are based on infrastructure, typically rely on signal information from wireless technologies, where the distance to reference points is determined to calculate the users position. In [1], a Radio Frequency (RF) based system has been developed. Signal strength information of several base stations with known location is recorded and processed. With the use of signal propagation models the position of the user can than be determined, resulting in a median resolution of 2 - 3 m. Methods based on UWB can achieve a position accuracy of 1 to 5 m [15]. In [2] WLAN signal strengths are measured and compared to signal strength maps, that have been created for the given area to determine a users position. This way a position error of less than 1 m can be achieved in some cases. Various other methods for the positioning based wireless networks, such as WLAN and Bluetooth exist. These include time measurements (time of arrival), relative time measurements (time difference of arrival), cell based positioning (cell of origin) as well as Received Signal Strength Indication (RSSI) approaches [16]. However, these methods have the disadvantage of being costly and time-consuming to install and often fail to reach a meter-scale accuracy. Infrastructure based systems that use laser light can reach sub-millimeter accuracies in certain cases, but they require line of sight connections to the reference points.

The method presented by [17] uses magnetic coils to generate magnetic fields. The distance to the coils with known position are derived through signal analyses and cross correlation. The position of the users device can than be estimated through trilateration, with an accuracy of less than a meter, for a close area (4 m distance to the coils). In [3] a different approach is presented that determines a device's position based on disturbances of the Earth's magnetic field, caused by steel elements in buildings. Large structures of magnetic sensitive material (e.g. steel) buildings warp the geomagnetic field. A reference map is created that contains information of the spatial variations of the geomagnetic field, which is compared with measurements from an array of e-compasses by the user, to determine the position. With this approach, an accuracy of up to 1 m could be demonstrated. Though the installation of a infrastructure, such as in the above mentioned methods is not necessary, it is still needed to create the magnetic field map and the device used for the positioning needs to be equipped with the necessary hardware.

Smartphones incorporate a variety of sensors to track the Magnetic, Angular Rate and Gravity (MARG) values as well as environment sensors to measure pressure, temperature, etc. that can be used for autonomous positioning. Typically a PDR serves as the basis of the localization. Essentially, a PDR is a combination of a pedometer, estimated stride length, and orientation. The system needs to estimate the step's length and direction from the IMU measurements after it detects a step [7]. The IMU measurements of smartphones are noisy and can only provide reliable location estimation in a limited amount of time, so external assistance is necessary. Approaches that use the integrated sensors with additional infrastructure support for positioning are called hybrid methods. To correct for drift and provide efficient localization, visual support can combine vision features [18, 19], point clouds from cameras [12], [9] and Light Detection and Ranging (LIDAR) sensors with odometry data. However, because of environmental circumstances or location of the smartphone, such as in a pocket or bag, such vision-based auxiliary features may not always be available. Other methods, such as state-of-the-art odometry based learning approaches [8, 13], which mainly rely on the history of motion and position sensor values to regress the velocity vector, would heavily rely on labeled data. The main problem with these methods is the lack of realistic data sets, which take into account long trajectory and unlimited human activities.

Another possibility to enhance the precision of the autonomous localization approach is to support the positioning with environmental information, such as floor plans and routing graphs of buildings. PF are well suited for processing the stochastically very different data of the smartphone sensors and the information of a building's geometry. Using PF algorithms, it is possible to consider inaccuracies in the position measurements, by using weighted particles as representation of the probability density distribution of the position estimates [14]. The weighted sum than results in the position estimate. The bootstrap PF, as used in [20, 21] is the most simple implementation of a PF algorithm. The weights of the particles are determined according to the plausibility of the position that they represent, e.g. their distance to routing edges or if the particles lie behind walls. The backtracking PF [22, 23] is a more complex approach, that enables the correction and re-estimation of implausible paths, based on the particles propagation history. This functionality can be especially useful in complex building structures, where wrong trajectories of particle clouds can be corrected, by resampling them to more plausible positions, based on the propagation history. PF can not only be used to combine the PDR approach with map material, but can also used in combination with e.g. Fifth-Generation of Mobile Telecommunications Technology (5G) signal information, as done in [24]. Here a concept has been developed to use signals from 5G antennas, installed in a building, to determine the absolute position of a device whenever possible, while using a PDR based method with PF for map matching the rest of the time.

The main drawback of the map matching approaches in general is the generation of the map material, which needs to be of good quality and the right data format. In [24] a workflow has been developed to extract the needed map material as GeoJSON-format from usually available CAD building plans. The PF algorithms in this thesis have been developed to be able to use map information in the GeoJSON-format, making the aquisition of the map material significantly easier.

# 3 Methodology

This chapter describes the underlying methodologies and concurrently presents related works. The methodologies include geospatial analysis, as well as the principles of the bootstrap Particle Filter (PF) and the PF with backtracking functionality.

## 3.1 Geospatial Analysis

Geospatial analysis describes the process of extracting, manipulating and visualizing of (usually georeferenced) spatial information [25]. One use case for geospatial analysis is the investigation of the spatial relationship between geometric objects. The most fundamental geometric objects are so-called geometric primitives. These are geometric objects, which can be described by one continuous geometry and include points, curves, surfaces and solids [26]. A point's interior consists of one point, it's boundary of no points and it's exterior of all other points. A curve has an interior of an infinitely number of points along it's length, a boundary of two end points and an exterior consisting of all other points. The interior of a surface consists of infinitely many points within, while the surface's boundary consists of one or more curves and the exterior consists of all other points.

Geospatial analysis can be done with a variety of tools, including Geographic Information System (GIS)-software such as QGIS or ArcGIS or by using programming languages, for example Python, with libraries such as GeoPandas [27]. GeoPandas is an open source project, extending the data types used by pandas to enable spatial analysis on geometric objects. These data types are *GeoSeries* and *GeoDataFrames*. The fundamental geometric objects are implemented in Shapely as (as *Point* class), *LineString* class and *Polygon* class. Several geometries of the same kind can be unified as collections, resulting in so-called *MultiPoints*, *MultiLinestrings* and *MultiPolygons* [28]. *GeoSeries* and *GeoDataFrames* can store one or several different geometric objects. *GeoDataFrames*, which are basically extensions of the *DataFrames* used in the Python library Pandas, also enable the storage of attributes of geometric objects. This spatial data structure includes spatial relationships between geometric objects, such as contains, intersects, overlaps, touches, etc. Geometric operations, such as the investigations of the mentioned spatial relationships, are performed by Shapely [28]. Queries for spatial relations will return a Boolean value (*True* or *False*). Some of the queries for spatial relationships, implemented in Shapely are shown in figure 3.1 and listed below:

- ***Intersects*** returns *True* if the boundary or interior of the object intersect in any way with those of the other.

- ***Contains*** returns *True* if no points of the other geometry lie in the exterior of this geometry and at least one point of the other geometry's interior lies in the interior of this geometry. A line's endpoints are it's boundary and are therefore not *contained*

- ***Within*** returns *True* if this object's boundary and interior intersect only with the interior of the other (not its boundary or exterior), making it the 'inverse' of *contains*.

- ***Touches*** returns *True* if the given objects' boundaries have at least one point in common, but their interiors do not intersect with any part of the other.

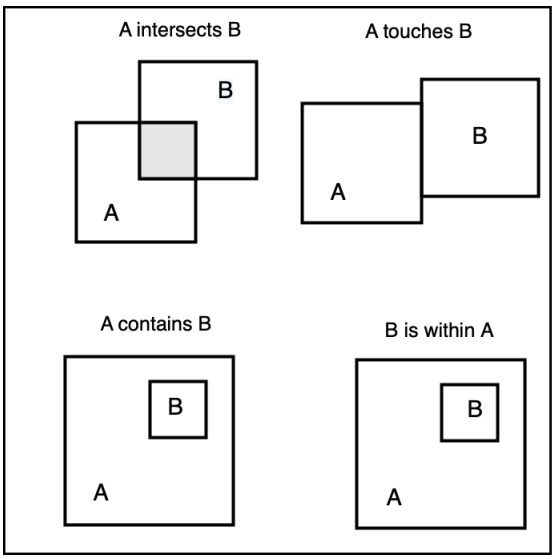



Figure 3.1: Overview of some of the spatial relations that can be queried in Shapely

The function of some of the fundamental geospatial analyses that can be applied to *GeoSeries* and *GeoDataFrames*, are listed below and are shown in figure 3.2:

- ***Distance*** returns the direct distance between geometries. It is also the orthogonal distance between a point and a line feature or edge, if the orthogonal projection of the point exists (see figure 3.2 (a). Here the distance from the corners of polygon B and the edges of the polygons A and C is simultaneously the orthogonal distance, whereas the distance between the corners of A and C is not).

- ***Spatial join*** is used to merge geometric objects based on their spatial relationships, as shown in figure 3.2 (b). Spatial relationships, that can be used as requirements for the spatial join include *intersects*, *contains*, *within* or *touches*.

- ***Query by attribute*** returns all objects of the given *GeoDataFrame* that hold the queried attribute value, as shown in figure 3.2 (c), where all objects with the *Type* value of B are selected and marked in blue.

- ***Buffer*** generates a representation of all points in a given distance of the geometry as polygon. In figure 3.2 (d), the resulting buffer-polygons around a given line and a square shaped polygon are displayed in green.

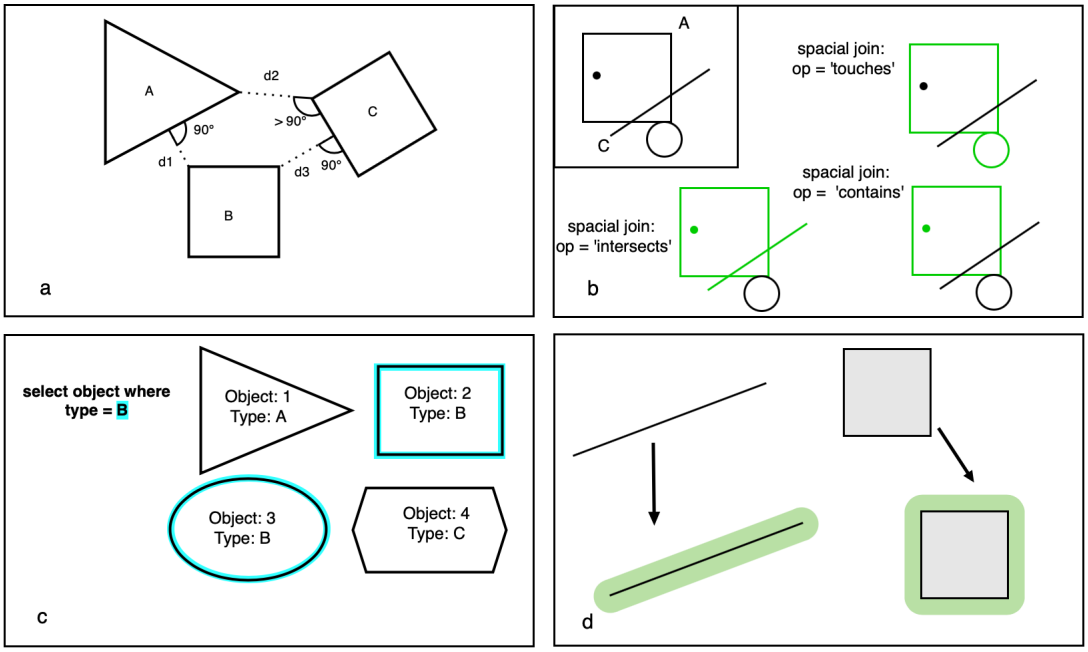



Figure 3.2: Overview of some of the used geospatial analyses in GeoPandas; with *distance* shown in (a), *spatial join* in (b), *query by attribute* in (c) and *buffer* in (d)

## 3.2 Particle Filter

PF belong to the Monte Carlo Methods, which are groups of state estimation methods to model highly non linear problems with very noisy measurements [29]. The PF, in contrast to the Kalman Filter (KF), are able to handle highly non-linear problems, where the uncertainties are represented by several estimation samples (so-called particles) [30]. The idea of a PF in the application of navigation, specifically for Pedestrian Dead Reckoning (PDR), is to approximate the Probability Density Function (PDF) of the position by a large number of weighted, independent particles [14]. The weighted sum of the particles will than result in a position estimate. In the following subsection 3.2.1, the general PF approach for indoor navigation is explained, based on the bootstrap PF. In the subsection 3.2.2 the Backtracking PF, a variation of the PF model, is explained in detail. Many

other variations of the PF exist, which mainly differ to each other by modifications of one or more of the components. For examples of other PF modifications, see [30] and [31] among others.

### 3.2.1 Bootstrap Particle Filter

The bootstrap PF is one of the simplest PF modification. For the bootstrap PF, at first a set of initial particles is created, representing the PDF of the starting position. The starting position can either be known or unknown. For the PDR, the particles are either all located at the known starting point as in [20] or they can be distributed around the starting point according to the uncertainty of the position. In other cases, for example in combination with absolute positioning methods such as Received Signal Strength Indication (RSSI), the particles could also be randomly distributed over the whole area, as done by [21]. The bootstrap PF then includes three main steps, called propagation, weighting (or estimation step) and resampling, which are explained in the following :

**Propagation:**

During the propagation step the individual particles are changed to better represent the PDF. In indoor positioning, the particles can spatially propagated, according to length and heading of the current step, which are derived from the Inertial Measurement Unit (IMU) measurements. Possible measurement errors can be taken into consideration by assigning a randomly distributed noise to step length and step heading for each particle, as done by [20]. The coordinates of the Position $x_i$ and $y_i$ for the $i^{th}$ propagated particle is then calculated from the coordinates of the last particle position $x_{last,i}$ and $y_{last,i}$ by (3.1) and (3.2):

$$x_i = x_{last,i} + (l_{step} + \epsilon_{l,i}) * cos(h + \epsilon_{h,i}) \tag{3.1}$$

$$y_i = y_{last,i} + (l_{step} + \epsilon_{l,i}) * sin(h + \epsilon_{h,i}) \tag{3.2}$$

Where $l_{step}$ is the length and $h$ is the heading of the current step and $\epsilon$ is a normal distributed noise value. This leads to a fan-like distribution of the particle in the walking direction. In [21], on the other hand, the step length for the particle propagation, is predicted from the last three observed steps. The weights of the particles remain unchanged during the propagation.

**Weighting:**

During the weighting step, the weights for the particles are updated according to the particles' plausibility, this way the position estimate can be improved with relevant information (e.g. from supported map material), while the position of the particles remains unchanged. A variety of weighting methods and criteria could be used. Following some weighting methods are described,

that are mainly based on map information:

***Weighting by routing:*** In the weighting by routing approach, as it is implemented by [20, 32], the orthogonal distance of each particle to the closest routing edge is considered, as shown in figure 3.3. The weight of each particle is hereby decreasing with increasing distance to the routing edge and can be calculated with equation 3.3:

$$wr_i = exp(-d_i^2/2) \tag{3.3}$$

Where $wr_i$ is the weight of the $i^{th}$ particle, with the orthogonal distance $d_i$ to the closest routing edge [20, 29].

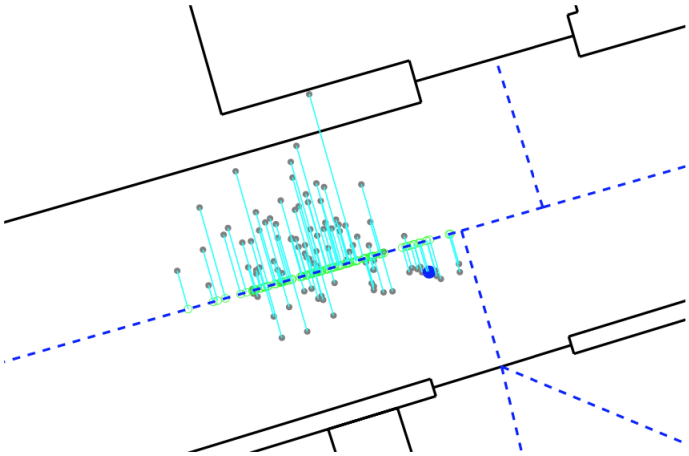


Figure 3.3: Principle of particle weighting based on routing edges(blue point: position; grey: distributed particles; cyan: orthogonal distances to selected routing edge) from [32]

***Weighting by walls:*** Another weighting approach, used by [20], determines the weight in relation to the distance of a particle to the next wall and the difference in the orientation of the particle propagation to the next wall's orientation. Figure 3.4 shows the distribution and weighting of the particles at a specific step. Red particles have the minimum weight, because they are behind walls and the other particles' weight, which increases with the distance to the wall, is displayed as shades of grey.

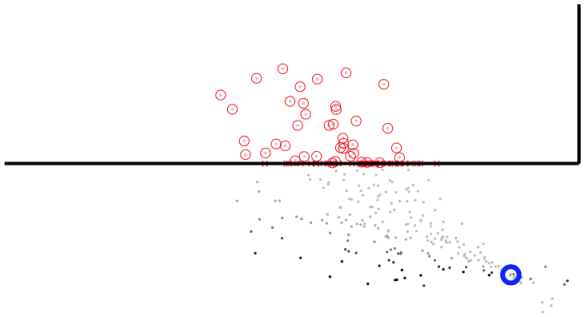


Figure 3.4: Particle distribution based on step length and heading and the weighting, dependent on wall information of the floorplan. (blue: last estimated position; red: particles behind wall with minimum weight) from [33]

***Weighting based on PDR and map material:*** Another approach for the weighting of particles is used in [21]. First, particles which are not plausible, based on the building information, receive a weight of zero, this includes particles that are outside of the building, inside walls, crossed walls or have a low distance from the wall. For this, methods of computational geometry [34, 35] have been adapted to determine if particles cross walls or are inside walls. The weight of each remaining particle $w_(k)$ depends on the difference between the *predicted* position of the particle ($x_k$, $y_k$) and the *observed* position ($zx_k$, $zy_k$), that is calculated according to the PDR from the last estimated position. The weights are calculated with the following equation, where $\sigma_z$ is the measurement noise, derived from empirical studies:

$$w_k = \frac{1}{2\pi\sigma_e^2} * exp\left(-\frac{(x_k - zx_k)^2 + (y_k - zy_k)^2}{2\sigma_z^2}\right) \tag{3.4}$$

**Resampling:**

The weighting of the particles can lead to the situation, that a small number of particles have a high weight which can lead to a diverging filter process. This can be avoided by a resampling process, which is shown in figure 3.5. The few particles with high weight are split up in several particles with weights that are in their sum equal to the original particles' weight. The goal is to assign a weight of $1/N$ for each particle, where $N$ is the number of particles. For this the particles are put into intervals according to their weights. Then a random number between 0 and 1 is generated and if this number falls in a particle's interval, than this particle will be reproduced with the weight $1/N$. Since particles with bigger weights have bigger intervals (indicated with $w_i$ for the size of the $i^{th}$ interval in figure 3.5), they are reproduced more often than particles with a lower weight [20, 29].

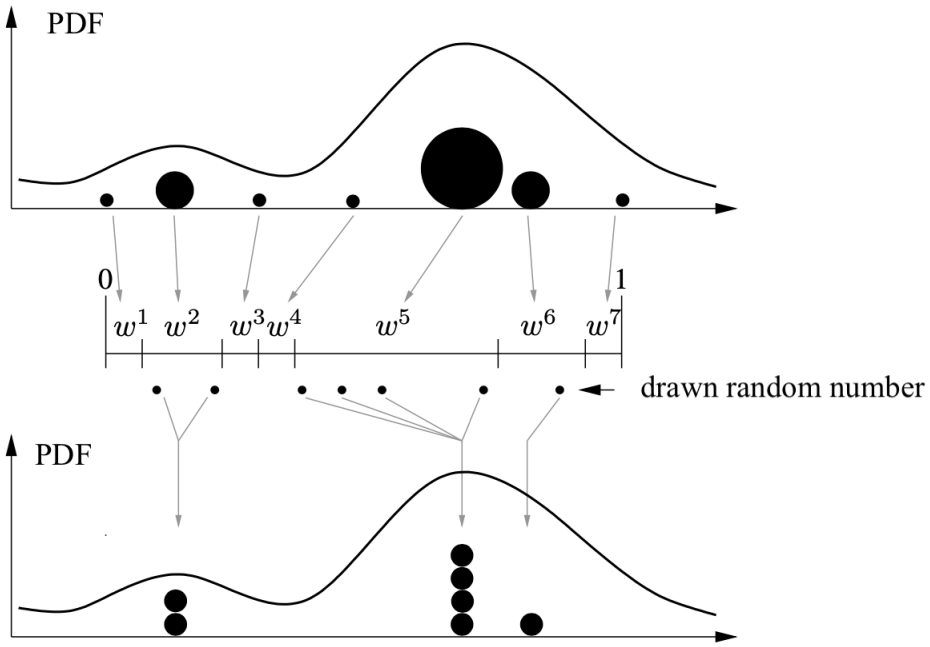



Figure 3.5: Schematic picture of the resampling process. With particles and their weights before (top) and after (bottom) the resampling (from [29])

To decide if a resampling should be done, the number of effective particles $N_{eff}$ can been used. $N_{eff}$ is defined by equation 3.5:

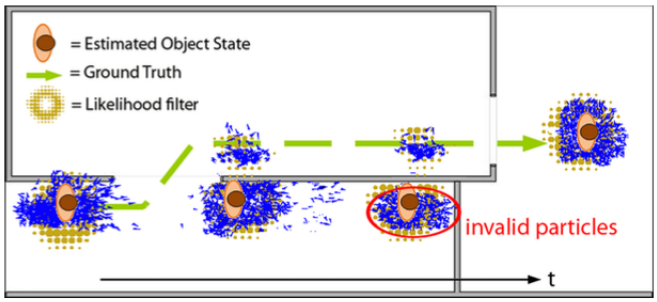
$$\frac{1}{\sum_{i=1}^{N} w_i^2} = N_{eff} \tag{3.5}$$

It is $N_{eff} = N$ if all particle have the same weight and $N_{eff} = 1$ if only one particle holds the whole weight. A threshold of $N_{eff} = 0.7$ is suggested by [29] and also used by [20].

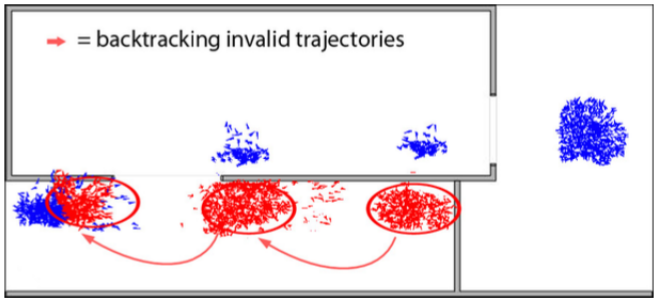
### 3.2.2 Particle Filter with Backtracking

A PF with backtracking works in its basic functionality such as the bootstrap PF. But Compared to the simple bootstrap PF, it also considers the particle trajectory's history to improve position estimates. During positioning with the map information support, it can occur that paths are not estimated correctly. For example, a change in direction occurs at a location where there are several junctions close by. It can happen that the path follows the wrong junction, due to uncertainties in the Positioning. The wrong path might lead to an aborting of the filter, due to strong corrections from the map information. If the scattering of the particles is wide enough, part of the particles might travel the correct path. In this case a backtracking functionality can recalculate the path from the stored history of the movements of the particles. This was implemented, for example

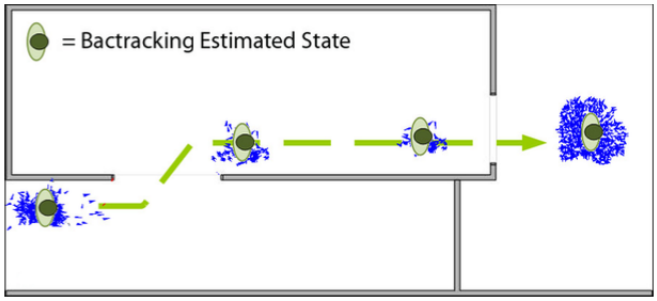
in [22], where an approach was developed to fuse RSSI fingerprinting and information from map material. In such a bactracking PF, after the propagation, the trajectories of invalid particles, for example such that cross walls or not accessible areas, are deleted during the so-called *filtering step*, in contrast to the typical case in the bootstrap PF, where the adjustment of the particles' plausibility at this point is done only by the assignment of weights. After this, the trajectories of valid particles are resampled in the *resampling step*, until a specified number of particles is reached. In this way the current as well as previous position estimations can be refined, as shown in figure 3.6.




(a) Detecting the invalid particles




(b) Backtracking the invalid trajectories



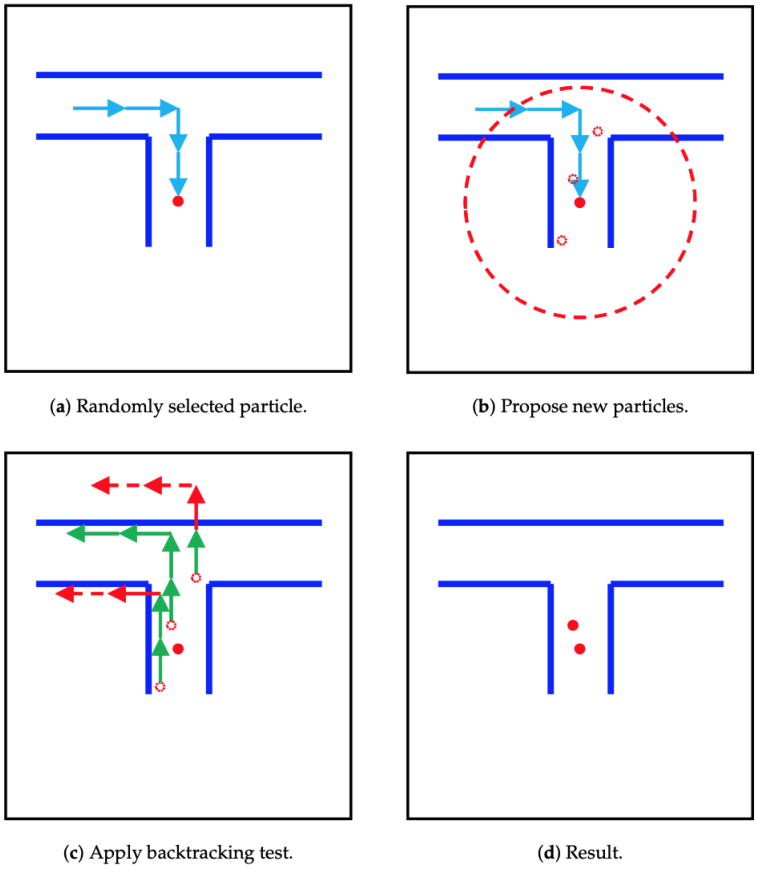

(c) Backtracking the estimated states

Figure 3.6: Backtracking PF for pattern matching localization, from [22]

In [23] a backtracking approach was used, inspired by the above mentioned one from [22]. But instead of storing the trajectory of every particle, only the history of the displacement, derived from the PDR, is stored and a scaling factor for the displacement, representing the noise of the step length, as well as a noise value for the orientation angle of the device is assigned to each

particle. During the propagation step, each particle's position is updated according to the current step's displacement multiplied with the scaling value of the particle. After the deletion of invalid particles in the filter step, the position estimate is calculated from the mean of the particle positions and the orientation of the device is updated by the mean of the products of each particle's noise value for the orientation and the measured orientation for the position.

Then, for every particle that was deleted in the filter step, it is tried to generate a new particle within a specified and random radius around a valid particle. The potential new particle's propagation history, according to the displacement of the steps, multiplied by the scaling factor of the particle, is then checked for any intersections with walls. If no intersections are detected, the particle is accepted, otherwise a particle with a different position and a different random scale for the displacement is proposed, as shown in figure 3.7.



(a) Randomly selected particle.

(b) Propose new particles.

(c) Apply backtracking test.

(d) Result.

Figure 3.7: Process of a backtracking test. Blue arrows are recent steps, red arrows are invalid paths; from [23]

# 4 Implementation

This chapter describes the implementation of the bootstrap Particle Filter (PF) and backtracking PF with geospatial analysis and the used data set. The developed algorithms are explained with the help of pseudo code sections. Both PFs have been developed in Python 3.8. The Python library GeoPandas [27], which includes Shapely [28], has been used for the implementation of geospatial analyses (as described in section 3.1). GeoPandas and Shapely have been installed using the Conda Forge environment. The Python scripts, developed in this thesis, along with the input data are handed in es a separate folder, along with this thesis. The files containing the code as well as the input and output data for the bootstrap PF are located at `/Data_and_Code/bootstrap_PF` and for the backtracking PF at `/Data_and_Code/PF_Backtracking`. In both PF approaches, the used data, described in section 4.1, has been loaded as global variables in the respective `input_data.py` file and is imported in the respective `main.py` files, which run the PF corresponding algorithm. Both PF algorithms have been structured in a modular way, to enable the relatively easy exchange and addition of further methods and functionalities.

## 4.1 Data Set

This section describes the used data sets, including the building information and the used values for step heading, step length and height. The data has been provided by [24]. The sensor data as well as the floor plans are openly accessible at https://github.com/Hossein-Shoushtari/IPIN21Data.

### Building Information

The building information of the HCU has been provided as floor plans in GeoJSON format, that was extracted from CAD plans [24] and have been minimally cleaned up manually. They contain polygons with attributes according to their type (e.g. walls, rooms, hallways, staircase, etc.). The floor plans have been provided separately for each, the ground floor, $1^{st}$ floor and $4^{th}$ floor. The complex architecture of the HCU building, including big, open hallways, small corridors as well as unconventional shaped rooms with pointy corners (see figure 4.1) results in a distinct environment to test the developed PF under various conditions.

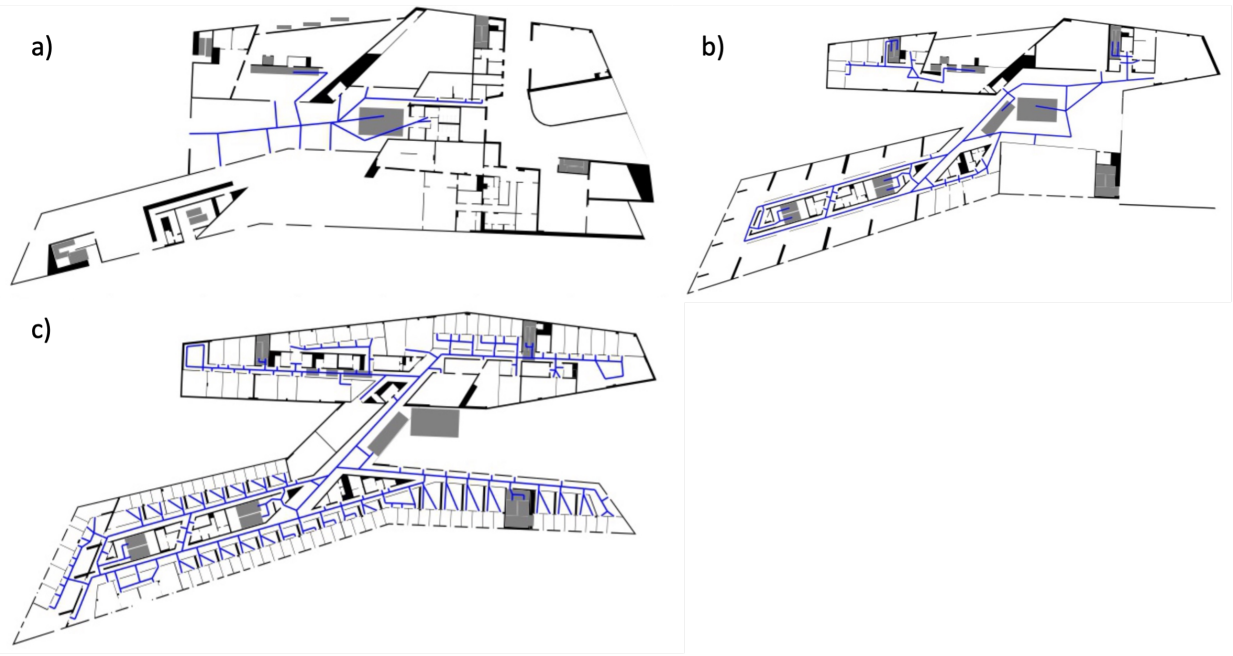



Figure 4.1: Floor plans of ground floor (a), $1^{st}$ floor(b) and $4^{th}$ floor (c), with the blue lines representing the routing edges

**Sensor Data**

The values for the step length and the step heading for the two tested trajectories have been readily provided in the Comma Separated Values (CSV) format. A handheld Samsung S10 5G smartphone has been used to record acceleration, barometer measurements and angular velocities from the gyroscope via an application named "sensor log", to calculate the step length and step heading [24]. The provided step headings are the orientation deviations in relation to the orientation at the starting point, making it necessary to add the absolute starting orientation to each heading value to derive the absolute heading to geographic north. The model for the step length calculation from the IMU values were based on a person with a different height and therefore a different step length than the persons who were generating the test data, which lead to a significant systematic error for the step length values. The height difference for each step was also provided as CSV file. To get the absolute height at any given step, the height differences until that step had to be summed up and added to the starting height. The starting height was approximated as the height of the starting floor plus 80 % of the users' height. The ground truth of the walked trajectories has been provided as CSV-file as well and contains the horizontal position, time stamp and the height difference for each point.

The height of the floors was estimated as 0 m for the ground floor, 6 m for the $1^{st}$floor and 19

m for the $4^{th}$ floor. The two trajectories are referred to as *eight* path and *zerotofour* path in the rest of this thesis. The *eight* path starts in a corridor of the $4^{th}$ floor in front of an office in the western direction and continues in the shape of an eight (see figure 4.2 for the ground truth of the *eight* path).

Figure 4.2: Ground truth points (orange) of the "zerotofour-path" in (a) to (c) and the "eight-path" in (d)

The *zerotofour* path starts in the entrance hallway on the ground floor (figure 4.3(a)) and leads over the main stairs to the $1^{st}$ floor. It than takes a left turn and proceeds to take the elevator figure (4.3(b)) to the $4^{th}$ floor. It continues along the corridor in the $4^{th}$ floor southwards, then takes a turn at the t-junction and leads into an office room in the south-eastern part of the $4^{th}$ floor (4.3 (c)).

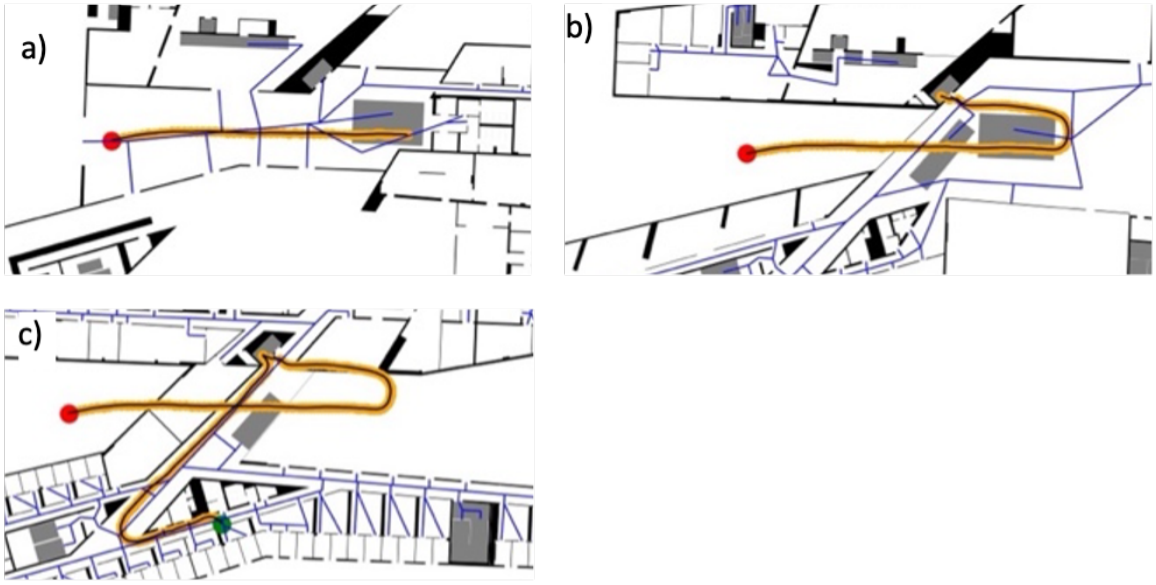



Figure 4.3: Ground truth points (orange) of the *zerotofour*-path in(a) to (c) and the *eight*-path in(d); the red dot is the starting position, the green dot is the finish

## 4.2 Bootstrap Particle Filter with Geospatial Analysis

This section explains the methods and algorithms of the bootstrap PF with geospatial analysis, starting with algorithm 4.1 that describes the overall algorithmic process of the PF. As input it requires the used weighting methods as a list of strings and the name of the walked path (*eight* or *zerotofour*) for which the data for step length, step heading, step height and start point are loaded (line 1). In line 2 to 4 the values of the needed parameters are set. For this PF, the start position is known and the initial particles are normally distributed around the starting point.

The initialization of the first particles is then done as described in subsection 4.2. The *for*-loop from line 6 on is then performing the PF algorithm for each step. For each step, first the map information for the current floor is selected. The particles' positions are than updated as described in section 4.2 and the weights for each particle are determined (line 9 to 24). The weights are calculated with each weighting method (see algorithm 4.3 to 4.6) specified in the input of the algorithm. The different weighting approaches have been tested separately as well as in several combinations in pairs of two weighting methods. If two weighting method have been used, the overall weight $w_i$ of each particle was calculated from the product of the two resulting weights $w_{1,i}$ and $w_{2,i}$ with equation 4.1:

$$w_i = w_{1,i} * w_{2,i} \tag{4.1}$$

If only one weighting method is used, the weights $w$ are equal to the weights derived from the used method. Afterwards, the calculated weights have been normalized according to 4.2:

$$\frac{w_i}{\sum_{k=1}^{N} w_k} = w_{i,norm} \tag{4.2}$$

Where $N$ is the number of particles and $w_{i,norm}$ is the normalized weight of the $i^{th}$ particle. The coordinates of the position estimate $X, Y$ were than calculated according to the weighted mean of the particles' coordinates $x_p, Y_p$ using equation 4.3 and 4.4:

$$X = \frac{\sum_{i=1}^{N} w_i * x_{p,i}}{N} \tag{4.3}$$

$$Y = \frac{\sum_{i=1}^{N} w_i * y_{p,i}}{N} \tag{4.4}$$

If all weights are invalid and have the minimum value, the values are so low, that they are treated as zero in Python. In this case the script raises an error notification and is unable to calculate the position, but doesn't abort and continues with the next step. Finally the particles are resampled with algorithm 4.7 and the *for*-loop starts again until the last step is reached. After the end of the PF algorithm, all calculated positions, as well as the standard deviation of each position estimate is returned and saved as a Comma Separated Values (CSV)-file.

**Algorithm 4.1** Bootstrap PF

**Require:** $WeightingMethods, path$
1: load $Startpoint, StepHeading, Steplength, height$ according to chosen $path$
2: $\sigma S = 0.1$
3: $\sigma H = 15 * \pi/180$
4: $ParticleNumber = 200$
5: $particles \leftarrow$ ***InitializeParticles(***$Startpoint, ParticleNumber$***)***
6: **for** $s$ in number of $Steps$ **do**
7: $walls, RoutingEdges, rooms \leftarrow$ data for current floor according to $height[s]$
8: $PropagatedParticles \leftarrow$ ***propagation(***$particles, StepHeading[s], Steplength[s], \sigma S, \sigma H$***)***
9: **if** $wl$ is in $WeightingMethods$ **then**
10: $weights \leftarrow$ ***WeightingByLOS(***$PropagatedParticles, walls$***)***
11: append $weights$ to $CalculatedWeights$
12: **end if**
13: **if** $wr$ is in $WeightingMethods$ **then**
14: $weights \leftarrow$ ***WeightingByRouting(***$PropagatedParticles, RoutingEdges$***)***
15: append $weights$ to $CalculatedWeights$
16: **end if**
17: **if** $wm$ is in $WeightingMethods$ **then**
18: $weights \leftarrow$ ***WeightingByRooms(***$PropagatedParticles, rooms$***)***
19: append $weights$ to $CalculatedWeights$
20: **end if**
21: **if** $ww$ is in $WeightingMethods$ **then**
22: $weights \leftarrow$ ***WeightingByWalls(***$PropagatedParticles, walls$***)***
23: append $weights$ to $CalculatedWeights$
24: **end if**
25: $w_{total} \leftarrow$ product of all $CalculatedWeights$
26: $PositionEstimate \leftarrow$ weighted mean of $particles$ positions
27: $particles \leftarrow$ ***Resampling(PropagatedParticles)***
28: **end for**

### Initialization

The initialization of the particles is done with algorithm 4.2 that requires the coordinates of the start position ($X_{start}$, $Y_{start}$) and the number of particles that have to be initialized as input. Each particle's position is calculated by adding a normally distributed error value to the coordinates of the start position (line 2 to 3) and a Shapely *Point*-geometry is created for this position. At the end, a *GeoDataFrame* containing all of these *Point*-geometries is returned.

**Algorithm 4.2** Create initial particles

**Require:** $X_{start}, Y_{start}, std, ParticleNumber$
1: **for** $i$ such that $0 \leq i \leq ParticleNumber$ **do**
2: $err \leftarrow$ *randnorm*
3: add *Point(*$X_{start} + err, Y_{start} + err$*)* to $ParticleList$
4: **end for**
5: $InitialParticles \leftarrow$ *GeoDataFrame(*$ParticleList$*)*
6: **return** $InitialParticles$

### Propagation

The particles' position is updated with a similar approach as [20] has used, based on the Pedestrian Dead Reckoning (PDR). The heading of the $i^{th}$ particle ($H_i$) is calculated by (4.5):

$$H_i = H_0 + H_{gy} + Rand_{norm} * \sigma H \tag{4.5}$$

Where $H_0$ is the initial heading, $H_{gy}$ is the estimated heading from the PDR, $\sigma H_i$ is the noise value for the heading and $Rand_{norm}$ is a normal distributed random number. Following the approach from [20], the step length $S_i$ for the $i^{th}$ particle is calculated by (4.6):

$$S_i = S_{acc} + Rand_{norm} * \sigma S \tag{4.6}$$

Where $S_{acc}$ is the measured step length, and $\sigma S$ is the noise value for the step length. The coordinates of the Position $X_i$ and $Y_i$ for the $i^{th}$ particle is then calculated from the coordinates of the last particle position $X_{last,I}$ and $Y_{last,i}$ by (4.7) and (4.8):

$$X_i = X_{last,i} + S_i * cosH_i \tag{4.7}$$

$$Y_i = Y_{last,i} + S_i * sinH_i \tag{4.8}$$

### Weighting methods

After the propagation of the particles, the weights of the particles are determined based on the map information, using the following methods, which are also pictured in figure 4.4:

***Weighting by line of sight:*** In the weighting by line of sight (*wl*) approach, the line representing the direct connection between a particle and the last estimated position is checked for intersections with walls for every particle, as shown in figure 4.4 (d). Algorithm 4.3 is the implementation of the *wl*. First all weights are set to one. In line 2 to 5 a *GeoDataFrame* containing all direct connections between the last estimated position and the current (propagated) particles as *LineStrings*. Then,

these connections are tested for intersections with the polygons representing the walls. For each connection, that intersects a wall, the weight of the respective particle is set to the minimal weight.

**Algorithm 4.3** Weighting by line of sight

**Require:** $particles, walls, lastPosition$
1: all $weights = 1$
2: **for** $particle$ in $particles$ **do**
3: append ***LineString***$(ParticlePosition, lastPosition)$ to $connections$
4: **end for**
5: convert $connections$ to ***GeoDataFrame***
6: $intersections \leftarrow$ spatial join on 'intersect' of $connections$ ***and*** $walls$
7: **if** number of $intersections \geq 1$ **then**
8: **for** $i \leftarrow$ ***index*** of $intersections$ **do**
9: $wieghts[i] \leftarrow MinimalWeight$
10: **end for**
11: **end if**
12: **return** $weights$

***Weighting by rooms:*** In the weighting by rooms (*wm*) method, every particle is examined if it is positioned in a polygon representing a currently valid room. All particles located in a valid room receive the weight 1, all other particles receive the minimal weight (see figure 4.4 (a)). In algorithm 4.4, which implements the *wm* method, all weights are set to the minimal value of 0.001. Then a spatial join between the two *GeoDataFrames* containing the particles and the valid rooms are done, returning all particles within those valid rooms (line 2). For each of those particles in a valid room, the weight is then set to 1 (line3 to 5). Different rooms could be used as valid rooms, in this case all staircases, lifts and corridors are used as valid rooms. Rooms that are not part of these, e.g. the small rooms in the test data set, can still be entered, if enough particles are in them, since their weight is not set to zero but to 0.001 giving them still a very small influence on the position estimate.

**Algorithm 4.4** Weighting by rooms

**Require:** $particles, ValidRooms$
1: all $weights = MinimalWeight$
2: $ParticlesWithin \leftarrow$ spatial join on 'within' of $particles$ and $ValidRooms$
3: **for** $i \leftarrow$ ***index*** of $intersections$ **do**
4: $wieghts[i] \leftarrow 1$
5: **end for**
6: **return** $weights$

***Weighting by walls:*** As in the *wm* approach, in the weighting by walls (*ww*) approach each particle is checked if it is inside a polygon. In this case the polygons represent the buildings' walls

and particles located in them are considered invalid and particles that are not inside the polygons and are considered valid (see figure 4.4 (c)). The algorithm 4.5, used to implement the weighting by walls (*ww*) method, is similar to algorithm 4.4, but the initial weights are set to 1. Now a spatial join between the two *GeoDataFrames* containing the particles and the walls are done, returning all particles within the walls (line 2). For each particle, that is located within a wall, the weight is then set to the minimal weight.

**Algorithm 4.5** Weighting by walls

**Require:** $particles, walls$
1: all $weights = 1$
2: $ParticlesinWalls \leftarrow$ spatial join on 'within' of $particles$ and $walls$
3: **for** $i \leftarrow$ *index* of $ParticlesinWalls$ **do**
4: $\quad wieghts[i] \leftarrow minimalWeight$
5: **end for**
6: **return** $weights$

***Weighting by rooting:*** The implemented weighting by routing (*wr*) approach is similar to the approach of [20, 32], as described in section 3.2.1. Here the distance of each particle to the closest routing edge is considered, as shown in figure 4.4 (b) and in line 2 of algorithm 4.6. For this, the direct distances between all routing edges and the particle are queried, by using the *distance* method of the *GeoDataFrame* containing all routing edges. The smallest of those distances is then the direct distance between the particle and the closest routing edge (called $mindist$ in algorithm 4.6. If this minimal distance is smaller than 3 m, the weight for this particle is calculated (line 4). If the minimal distance is greater than 3 m, the corresponding weight is set to the minimal weight-value of 0.001. To make the algorithm faster and simplify the computations, the shortest distance from each particle to the next routing edge has been used instead of always calculating the orthogonal distances.

**Algorithm 4.6** Weighting by routing

**Require:** $particles, RoutingEdges$
1: **for** $particle$ in $particles$ **do**
2: $\quad mindist \leftarrow$ minimal of all*distances* between $RoutingEdges$ and $particle$
3: $\quad$ **if** $mindist < 3$ **then**
4: $\quad\quad weight$ of $particle \leftarrow \exp\left(-0.5 * \frac{mindist^2}{\epsilon_{rout}}\right)$
5: $\quad$ **else**
6: $\quad\quad weight$ of $particle \leftarrow minimalWeight$
7: $\quad$ **end if**
8: **end for**
9: **return** $weights$

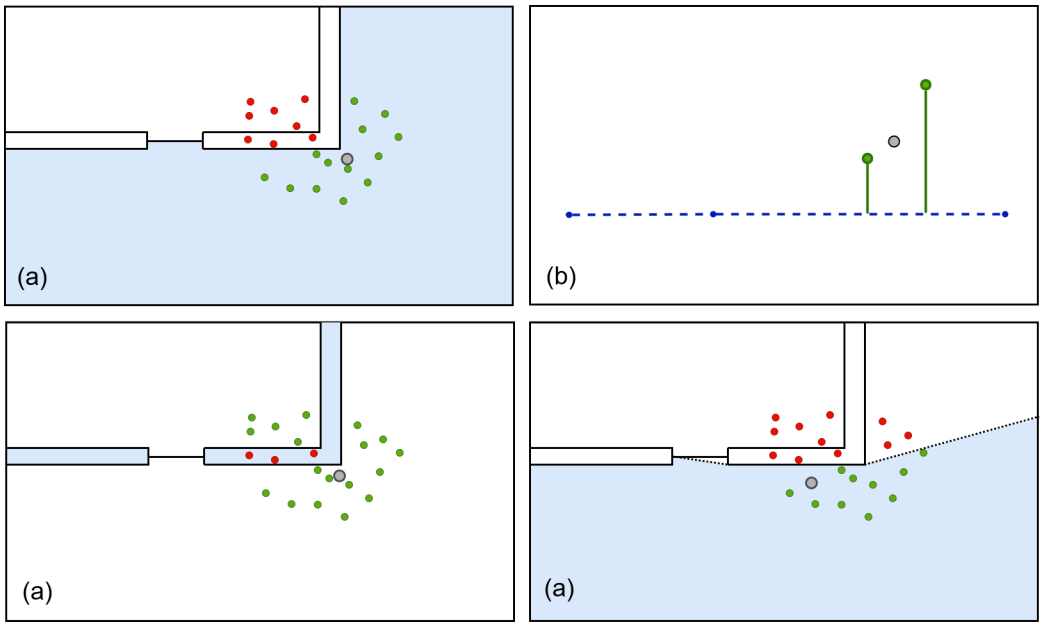



Figure 4.4: Overview of the used weighting methods, with the *wm* in (a), *wr* in (b), *ww* in (c) and *wl* in (d)

### Resampling

The algorithm for the resampling step (algorithm 4.7), is used to perform the low variance resampling, as described in section 3.2.1. The resampling process is only started, if $N_{eff}$ is smaller than 0.7 (for the calculation see equation 3.5), otherwise the original particle positions are returned. To determine the limits of the resampling intervals for the resampling, the cumulated values of the particles' weights are used (line3). In line 4 to 8, for each particle a random number between 0 and 1 is drawn and the position of every particle, which's corresponding weight interval's upper limit is smaller than the drawn random number (line 6 and 7), is added to the list of particles to resample (line 8). As explained in section 3.2.1 this results in particles with higher weight being resampled more often. The weights are not changed, since the weights will be recalculated for every particle during the next loop of the PF algorithm. The resampled particles are than returned as a *GeoDataFrame* to be used in the next propagation step.

**Algorithm 4.7** Resampling

**Require:** $particles$
1: **if** $N_{eff} =< 0.7 * ParticleNumber$ **then**
2:  **for** $particle$ in $particles$ **do**
3:  $intervals \leftarrow$ *list* of cumulated $weights$
4:  **for** each $particle$ in $particles$ **do**
5:  $rand_1 \leftarrow randomnumber$ between 0 and 1
6:  **for** $i$ in $Index$ of $particles$ **do**
7:  **if** $intervalls[i] > rand_1$ **then**
8:  append $particles[i]$ to $ResampledParticles$
9:  **end if**
10:  **end for**
11:  **end for**
12:  **end for**
13:  convert $ResampledParticles$ to *GeoDataFrame*
14:  **return** $ResampledParticles$
15: **else**
16:  **return** $Particles$
17: **end if**

## 4.3 Backtracking Particle Filter with Geospatial Analysis

This section explains the implementation of the fundamental algorithms of the Backtracking PF with geospatial analysis. Algorithm 4.8 describes the overall algorithm of this PF approach. As input it requires the name of the chosen path and the name of the chosen filtering concept as string and a boolean statement if the routing support (*wr*) should be used. Similar to the bootstrap PF with geospatial analysis, first the data for the step heading, step length and step height for the chosen path are loaded (line 1), followed by the setting of the needed parameters in line 2 to 5. In line 6 the initial particles at the start position are created (see algorithm 4.9, before the *for*-loop from line 7 on iterates over all steps while performing the PF-processes. For every step the current floor is determined and the floor data is selected accordingly (line 8 to 10). In contrast to the bootstrap PF with geospatial analysis, a step-object ($Step$) is created (line 11), which stores the length, heading, height, current floor, the current position estimate and the room it is located in, as well as all rooms where particles are allowed for the current step. In line 12 the particles' positions are then propagated and filtered regarding their plausibility according to the chosen filter concept (for detailed explanation see algorithms 4.10 to 4.12). If routing support is active, the particles' weights according to the *wr* method are determined afterwards, using algorithm 4.6, otherwise all weights are set to one. The weighted mean of the particles position then, provides the position estimate. Following the estimation of the position, the backtracking algorithm (see algorithm 4.13,

is executed to create new particles (corresponding to the *resampling* of the bootstrap particle filter) and add them to the *GeoDataFrame* containing the propagated, valid particles. These are then used in the next loop of the backtracking PF algorithm. After the PF is performed on the last step, all calculated positions, as well as the standard deviation of each position estimate is returned and saved as a CSV-file.

**Algorithm 4.8** Backtracking PF

**Require:** $path$, $FilterConcept$, $RoutingSupport = True/False$
load $Startpoint, StepHeading, StepLength, StepHeight$ according to chosen $path$
$maxParticleNumber = 200$
$\sigma_R = 1.5^2$
$\sigma H = 15 * np.pi/180$
$\sigma S = 0.1$
$particles \leftarrow$ ***InitializeParticles(***$Start, maxParticleNumber, \sigma H, \sigma S$***)***
**for** $i$ in $StepNumber$ **do**
  $CurrentFloor \leftarrow$ ***CheckFloor(***$Height$***)***
  $FloorChanged \leftarrow True/False$
  set $FloorData$ according to the $CurrentFloor$
  $Step \leftarrow$ ***Step(***$StepLength, StepHeading, Height, CurrentFloor, CurrentRoom, ValidRooms, CurrentPosition, StepScale$***)***
  $PropagatedParticles \leftarrow$ ***CheckParticleSteps(***$particles, Step, FloorData, FloorChanged, ransition$***)***
  **if** $RoutingSupport$ is $True$ **then**
    $weights \leftarrow$ ***WeightingByRouting(***$PropagatedParticles, RoutingEdges$***)***
  **else**
    set all $weights$ to 1
  **end if**
  $PositionEstimate \leftarrow$ weighted mean of $particles$ positions
  append $PositionEstimate$ to $Trajectory$
  $NumberOfNewParticles \leftarrow maxParticleNumber -$ number of $PropagatedParticles$
  $BacktrackingStepNumber \leftarrow$ number of $Steps$, but max 32
  $BackTrackingSteps \leftarrow$ walked Steps in reverse order until $BacktrackingStepNumber$ or a floor change is reached
  $particles \leftarrow$ ***BackTracking(***$NumberOfNewParticles, PropagatedParticles, BackTrackingSteps, FloorData, maxTries, \sigma H, \sigma S$***)***
  **return** $PositionEstimate$
**end for**
**return** $Trajectory$

### Particle Initialization

In algortihm 4.9, the initialization of particles is shown. As for the bootstrap PF with geospatial analysis, the start position is considered to be known and each particle position is calculated in by adding a normally distributed noise value to the coordinates of the start position. Here, for each particle an individual noise value for step heading and step length is generated by multiplying the noise values for step heading ($\sigma H$) and step length ($\sigma S$) with a normally distributed random number (line 3 and 4). The particles position, represented by *Point*-geometries, as well as the noise values are then stored in a *GeoDataFrame*, that is returned.

**Algorithm 4.9** Create initial particles

**Require:** $X_{start}, Y_{start}, MaxParticleNumber$
1: **for** $i$ such that $0 \leq i \leq MaxParticleNumber$ **do**
2: append *Point(*$X_{start} + err_{randnorm}, Y_{start} + err_{randnorm}$*)* to $geometries$
3: append $\sigma S + err$ to $LengthNoise$
4: append $\sigma H + err$ to $AngleNoise$
5: **end for**
6: $InitialParticles \leftarrow$ *GeoDataFrame(*$geometries, LengthNoise, AngleNoise$*)*
7: **return** $InitialParticles$

### Propagation and particle check

In the backtracking PF with geospatial analysis, the propagation of the particles is integrated in the filter process, in which the particles are checked for their plausibility by either the check for line of sight (*cl*), check for room (*cm*) or check for routing (*cr*) concept. The new position of each particle $X_i, Y_i$ is calculated in the propagation with equations 4.9 and 4.9:

$$X_i = X_{last,i} + (S + \epsilon_{l,i}) * cos(H_0 + H_{gy} + \epsilon_{h,i}) \tag{4.9}$$

$$Y_i = Y_{last,i} + (S + \epsilon_{l,i}) * sin(H_0 + H_{gy} + \epsilon_{h,i}) \tag{4.10}$$

Where $H_0$ is the initial heading, $H_{gy}$ is the estimated heading from the PDR, $S_{acc}$ is the measured step length and $\epsilon_{l,i}$ and $\epsilon_{h,i}$ are the noise values for step length and step heading for each $i_{th}$ particle. The *cl* (see algorithm 4.10, requires the current particles, the current step object and the statement if the floor changed in this step. It further requires the *GeoDataFrame* containing the polygons representing the relevant stairs and lifts (called $CurrentTransition$ in algorithm 4.10,4.11 and 4.12). In line 1 to 3, particles that are not located in the polygon contained in $CurrentTransition$ are deleted. The particles are than propagated as mentioned above. The step heading and step length are obtained from the *Step* object. Next, a GeoDataFrame is created containing all the connections between each propagated partice and it's previous position (line 5 to 7). Every particle,

whose corresponding connection line intersects a wall is delete, every other particle is kept in the *GeoDataFrame* (line 9 to 17). Finally the *GeoDataFrame* with the remaining particles is returned.

**Algorithm 4.10** Check particles for intersections

**Require:** $particles, Step, walls, FloorChanged, CurrentTransitions$
1: **if** $FloorChanged$ is $True$ **then**
2: $particles \leftarrow particles$ in $CurrentTransition$
3: **end if**
4: $PropagatedParticles \leftarrow$ ***propagation(***$particles, StepHeading, StepLength$***)***
5: **for** each $p1$ in $particles$ and $p2$ in $PropagatedParticles$ **do**
6: append ***LineString(***$p1$,$p2$***)*** to $connections$
7: **end for**
8: $intersections \leftarrow$ spatial join on 'intersect' between $connections$ and $walls$
9: **if** number of $intersections \geq 1$ **then**
10: **for** $p$ in $PropagatedParticles$ **do**
11: **if** $p$ is not endpoint of any line in $intersections$ **then**
12: keep $p$
13: **else**
14: delete $p$
15: **end if**
16: **end for**
17: **end if**
18: **return** $PropagatedParticles$

Algorithm 4.11, for the *cm* concept is similar structured as algorithm 4.10. But here not the walls are required as map information, but the rooms considered as valid for the particles ($ValidRooms$) and doors. If a floor change took place, the rooms currently contained in the *GeoDtaFrame* $ValidRooms$ are replaced with the *polygon*-geometry representing the current transition (line 1 to 3). For each propagated particle, it is checked if it is either within one of the valid rooms or if the *LineString*-geometry from it's new and it's previous position intersects either with any door-*polygon* or any polygon contained in $Transitions$. If that is the case, it is appended to the valid particles (line 5 to 9), which are finally returned as a *GeoDataFrame*.

**Algorithm 4.11** Check particles for containing room

**Require:** $particles, Step, FloorChanged, Transition, ValidRooms, doors)$
1: **if** $FloorChanged$ is $True$ **then**
2: $particles \leftarrow particles$ in $CurrentTransition$
3: **end if**
4: $PropagatedParticles \leftarrow$ ***propagation(***$particles, StepHeading, StepLength$***)***
5: **for** each $p1$ in $particles$ and $p2$ in $PropagatedParticles$ **do**
6: **if** $p2$ is in $ValidRooms$ or LineString($p1$,$p2$) intersects $Doors$ or $Transitions$ **then**
7: append $p2$ to $ValidParticles$
8: **end if**
9: **end for**
10: **return** $ValidParticle$

The algorithm 4.12, for the *cr* concept, is similar to 4.6. All particles are propagated in the same way as in algorithm 4.10 and 4.11. For each propagated particle the smallest distance to the routing edges is determined (as in algorithm 4.6). Only if the smallest distance to the routing edges for a particle is smaller than 2 m, it is appended to the list of valid particles (line 3 to 4). The valid particles are then, as for the other concepts, returned as a *GeoDataFrame*.

**Algorithm 4.12** Check particles for distance to routing edges

**Require:** $particles, Step, RoutingEdges$
1: $PropagatedParticles \leftarrow$ ***propagation(***$particles, StepHeading, StepLength$***)***
2: **for** each $p$ in $PropagatedParticles$ **do**
3: **if** distance between $p$ and $RoutingEdges < 2$ **then**
4: append $p$ to $ValidParticles$
5: **end if**
6: **end for**
7: **return** $ValidParticles$

### Backtracking

Following, the algorithms for the backtracking procedure are explained. Algorithm 4.13 overall procedure, which basically tries to generate new particles until the maximal number of particles is reached again. First, a number particles, according to the difference between the number of valid particles and the maximum number of particles, is randomly chosen from the currently existing particles (line 2). For each sample particle, a new particle at a random position inside radius (line 1) is proposed (line 5). For this proposed particle, a backtracking test is performed and if it passes the test, it is appended to the existing particles (line 6 to 8). If the particle doesn't pass the backtracking test, the procedure is repeated until either the maximum number of particles is reached or until a maximum number of tries (in this case eight tries) has been reached. The limited

number of tries will prevent an unnecessary slowing down of the algorithm, due to a very high number of tries for each particle. The *GeoDataFrame*, to which the accepted particles are added, is finally returned.

**Algorithm 4.13** Backtracking

**Require:** $NumberOfNewParticles, particles, BackTrackingSteps, walls, MaxTries, \sigma H, \sigma S$

1: $Radius = 1$
2: $Samples \leftarrow$ ***randomChoice(***$particles, NumberOfNewParticles$***)***
3: **for** $p$ in Samples **do**
4:  **while** $Try > 0$ **do**
5:  $NewParticle \leftarrow$ ***ProposeParticle(***$p, Radius$***)***
6:  **if** ***BackTrackingTest(***$NewParticle$***)*** is passed **then**
7:  append $NewParticle$ to $particles$
8:  **end if**
9:  $Try = Try - 1$
10:  **end while**
11: **end for**
12: **return** $particles$

For each concept of the checking of particles, the corresponding concept for the backtracking test is used. The backtracking test that is used with the *cl* concept is shown in algorithm 4.14. For each backtracking step $i-1$ from the current step $i$ the new position $X_{p,i-1}, Y_{p,i-1}$ of each particle $p$ is calculated from the current position of the particle $X_{p,i}, Y_{p,i}$ with equation 4.11 and 4.12:

$$X_{p,i-1} = X_{p,i} - (S_{i-1} + \epsilon_{l,p}) * cos(H_0 + H_{gy,i-1} + \epsilon_{h,p}) \tag{4.11}$$

$$Y_{p,i-1} = Y_{p,i} - (S_{i-1} + \epsilon_{l,p}) * sin(H_0 + H_{gy,i-1} + \epsilon_{h,p}) \tag{4.12}$$

Where $S_{i-1}$ is the step length of the last step and $H_{gy,i-1}$ is the heading of the last step. Each new particle position is added to the backtracking trajectory (line 1 to 3). The backtracking trajectory is then checked for intersections with walls. If any intersections are found, the backtracking test is not passed and it returns *False*, otherwise it is considered as passed and the algorithm returns *True* (line 5 to 8). It is not checked for intersection at every step, but for all steps simultaneously, because the query for intersections is the most time consuming step and the usage of *GeoDataFrames* makes it more efficient to query for intersections of the *LineString*-geometry representing the whole trajectory, than to query for intersections at each step, if multiple steps are considered.

**Algorithm 4.14** Backtracking test intersections

**Require:** $particle, Steps, walls$
1: **for** each $s$ in $BacktrackingSteps$ **do**
2: *propagate*$(particle, s)$
3: append new position of $particle$ to $trajectory$
4: **end for**
5: **if** $trajectory$ intersects $walls$ **then**
6: **return** $False$
7: **end if**
8: **return** $True$

The algorithm 4.15 is used for the backtracking test, that is performed in the *cr* concept. For each propagation according to the backtracking steps (see equation 4.11 and 4.12), it is checked if any of the polygons in the *GeoDataFrame* of the valid rooms contains the particle at the new position (line 3 to 7). If it is contained in a valid room, the test continues, otherwise it returns *False*. If the *for*-loop can proceed for all backtracking steps, the backtracking test is considered as passed and it returns True.

**Algorithm 4.15** Backtracking test rooms

**Require:** $particle, Steps, ValidRooms$
1: **for** each $s$ in $Steps$ **do**
2: *propagate*$(particle, s)$
3: **if** $ValidRooms$ contains $particle$ **then**
4: $continue$ with test
5: **else**
6: **return** $False$
7: **end if**
8: **end for**
9: **return** $True$

For the *cr*, the algorithm 4.16 is used for the backtracking test. It is similar to algorithm 4.15. The only difference is, that here for every backtracking step the shortest distance between the particles' new position and the closest routing edge is determined. If it is less then 2 m the *for*-loop continues, if it is 2 m or more the backtracking test stops and returns *False*.

**Algorithm 4.16** Backtracking test routing

**Require:** $particle, Steps, RoutingEdges$

1: **for** each $s$ in $Steps$ **do**
2: *propagate*$(particle, s)$
3: **if** distance between $particle$ and $RoutingEdges < 2$ **then**
4: $continue$ with test
5: **else**
6: **return** $False$
7: **end if**
8: **end for**
9: **return** $True$

# 5 Results

In this chapter, the results of two test trajectories are presented and discussed. Both, the bootstrap Particle Filter (PF) and the backtracking PF with geospatial analysis have been tested on two trajectories through the HafenCity University (HCU) building, the *eight* path and the *zerotofour* path, as described in section 4.1. First, different weighting methods for the bootstrap PF are tested and compared (section 4.2), followed by the comparison of the test results of the different concepts of the backtracking PF (section 4.3). For both PF approaches, different values to correct the systematic error of the step lengths (see section 4.1) have been tested. The comparison of different corrections of the step length enabled interpretations regarding the robustness of the PF algorithms in regards to systematic error of the step length estimation. Finally, the results of the backtracking PF and the bootstrap PF are compared to each other (section 5.3). To compare the accuracy of the different PF algorithms, the Cumulative Distribution Function (CDF) of the positioning error has been calculated. The CDF has been calculated for discrete error values, without fitting of a normal distribution function to the cumulative probabilities of the error values.

## 5.1 Bootstrap Particle Filter

The bootstrap PF has been tested with the following weighting methods (described in section 4.2): weighting by rooms (*wm*), weighting by line of sight (*wl*), weighting by routing (*wr*) and weighting by walls (*ww*). Further, the *wm*, *wl* and *ww* concept have been tested in conjunction with the *wr* concept. Figure 5.1 shows the CDF of the the position error for the different weighting concepts for the *eight* path with a step correction of 0.1 m. The *wl* method stands out by it's bad performance, an error of 5 m or less could only be achieved for slightly more than 40 % of the steps. In 90 % a position error of nearly 9 m or less could be achieved. The *wm* concept, with and without additional use of the *wr* concept as well as the *wl* in combination with the *wr* concept provided the best results, with an error of 2 - 3 m in 90 % of the time.

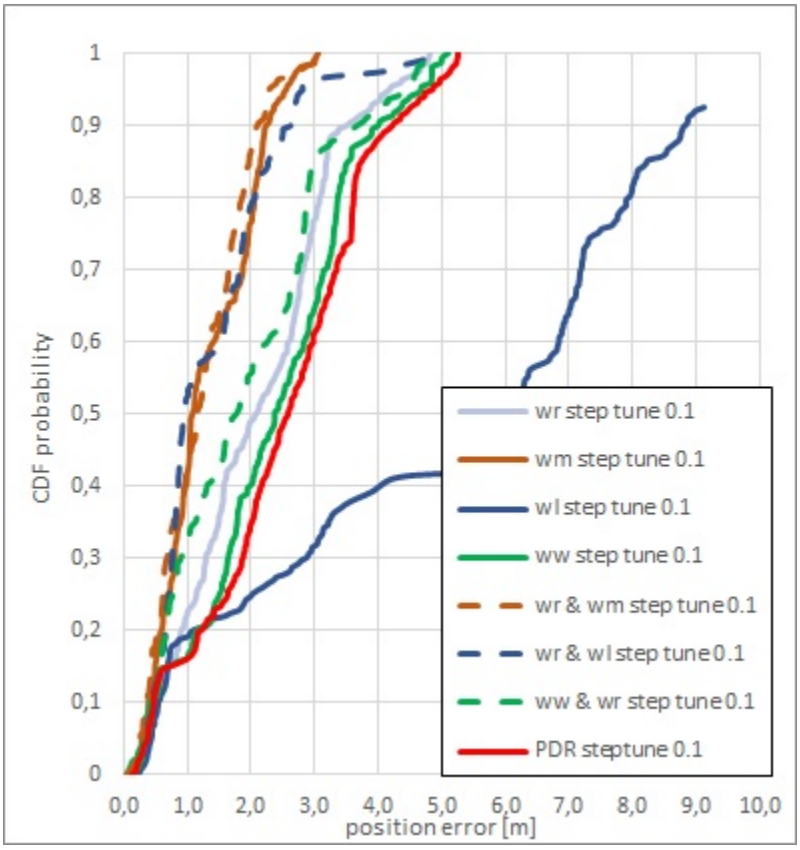



Figure 5.1: CDF of the positioning error of the *eight* path with step length correction of 0.1 m for the bootstrap PF and the PDR without a step correction.

The combination of the *wr* and the *wl* method reduced the positioning error significantly, which can also be observed on the respective trajectories, displayed in figure 5.2. When only the *wl* method is used, the trajectory falsely deviates into neighbouring rooms (see figure 5.2 (a)) because the doors are not considered in the *wl* method. Once the estimated position is in a wrong room, all particles outside of this wrong room are falsely considered invalid because they are located behind walls, viewed from the last estimated position. When additionally using the *wl* method, particles close to the routing edges receive a higher weight. This decreases the influence of the particles further away in the wrong rooms on the position estimate and therefore helps in preventing the position to be calculated in a wrong room, resulting in the trajectory in figure 5.2 (b). Despite the improved accuracy, the trajectory from the combination of *wl* and *wr* missed the last turn. For the *wm* method, the additional use of the *wr* method had a negligible effect on the error of the position estimate and for the *ww* method, the effect was small as well.

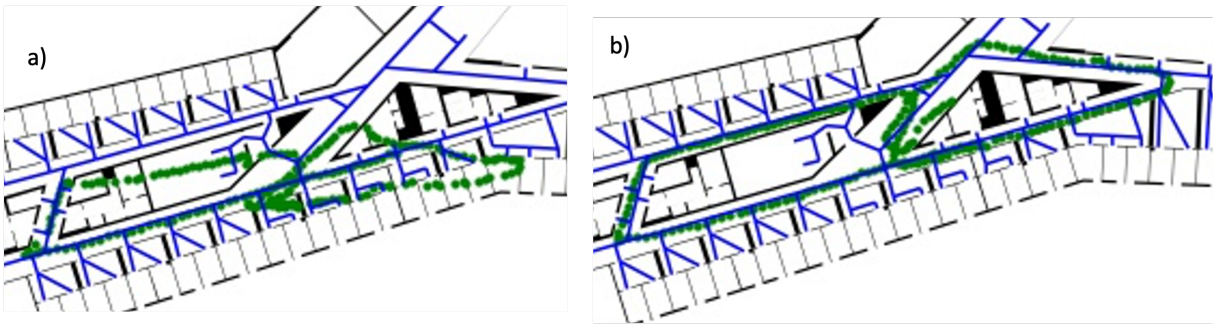



Figure 5.2: Trajectory from the *wl* method (a)) and the combination of *wl* and *wr* method (b)) for *eight* path with step length correction of 0.1 m

Figure 5.3 shows the error CDF for the *zerotofour* path with a step correction of 0.2 m, that was added to each step's length. The position errors for the *zerotofour* path are generally larger than for the *eight* path, while the accuracy of the individual weighting methods are less distinguishable than for the *eight* path. This is mainly a result from the situation that this path lead through more wide areas, such as the big entry hall at the ground floor, that provide only very little opportunities to correct wrongly estimated paths of the particles. Since less information is available, the trajectories are more similar to each other, as well as to the trajectory, which would be derived solemnly from the PDR.

The *wr* as well as all other methods in combination with the *wr* method provided slightly better results for the accuracy (with a probability of 90 % for an error of less than 6 m) than the *ww*, *wm* and *wl* method by themselves, which could achieve a accuracy between 6 and 7 m or better at 90 % of the time.

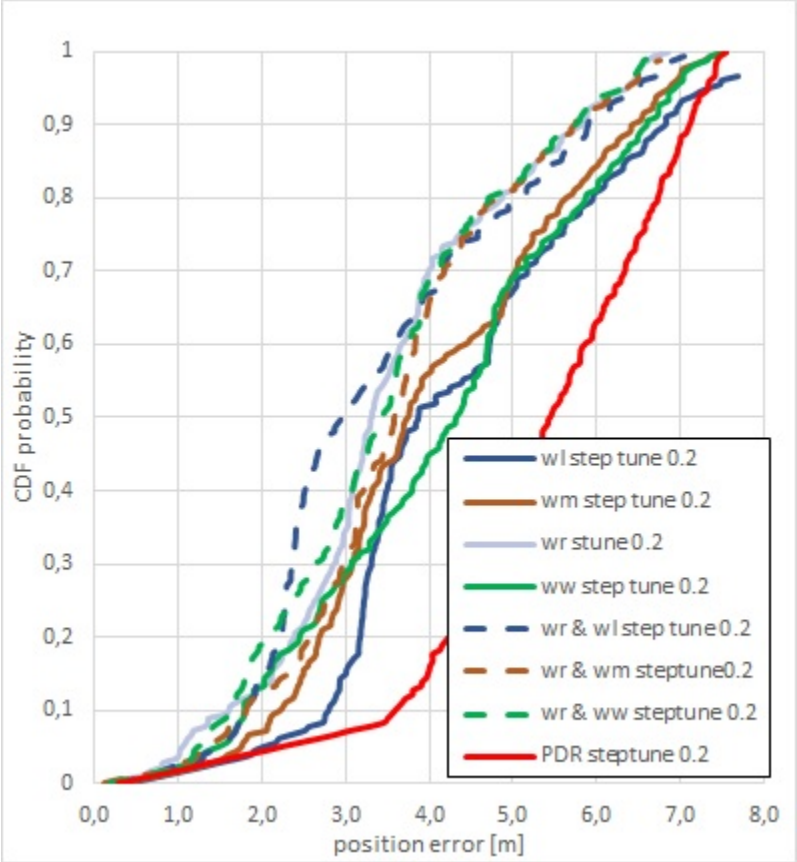



Figure 5.3: CDF of the positioning error of the *zerotofour* path with step length correction of 0.2 m for the bootstrap PF and and the PDR with a step correction of 0.2 m

Figure 5.4 shows the trajectories of the *wl*, *ww* and *wm* methods in combination with the *wr* for the *zerotofour* path. Of all the tested methods only the trajectories of the *ww* with *wr* and the *wm* method were able to enter the elevator during the *zerotofour* path, while the other trajectories missed it closely. The last room was only reached by the trajectories from the *ww* in combination with the *wr* as well as the *wm* method with and without the additional support through the *wr* method. The combination of *wl* and *wr* even missed the last turn in the $4^{th}$ floor, where some positions could not be estimated because all particles were invalid.

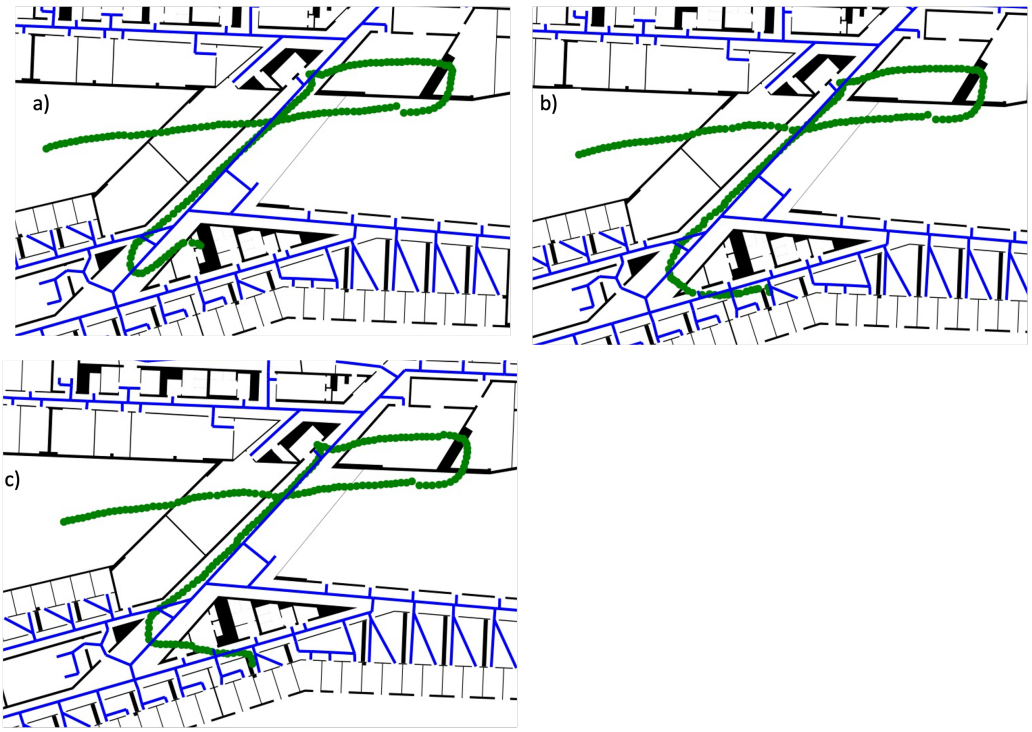



Figure 5.4: Trajectories of the *wl* (a)), *wm* (b)) and *ww* (c)) methods in combination with the *wr* for the *zerotofour* path with step correction of 0.2 m; green dots: estimated positions

The results for the *eight* path without any step length correction is displayed in figure 5.5. The performance of the *wl* method with and without the *wr* method worsens significantly. The accuracy of the *wm* method is similar to the test with a step correction of 0.1 m, with a positioning error of 2 to 3 m 90 % of the time. The *ww* and the *wr* methods performed similar to the test with a step correction with 0.1 m as well.

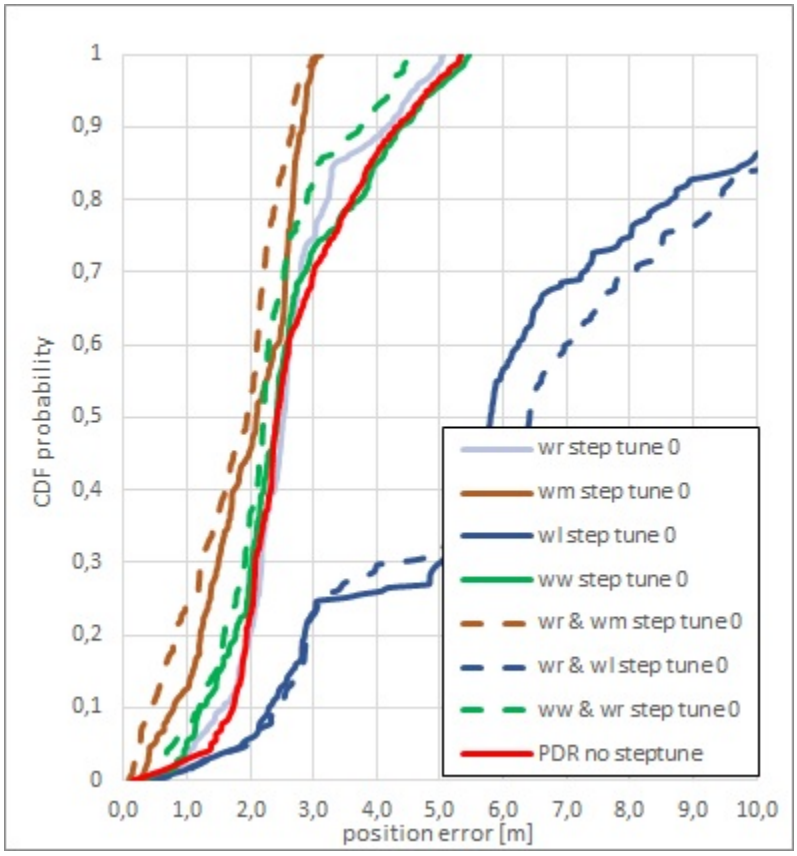



Figure 5.5: CDF of the positioning error of the *eight* path without step length correction for the bootstrap PF and the PDR without a step correction.

The *wl* method is strongly effected by the smaller step correction, because shorter steps result in earlier turns at corners, placing many particles behind corners or walls, blocking the Line of Sight (LOS) to the last position. This leads to a back turn of the trajectory (as seen in figure 5.6 (a), leading it through a door into the neighbouring room, while the *wm* method only cuts the corner a bit too early (see figure 5.6), as the particles on the other side are in the same room (the corridor) and therefor valid.

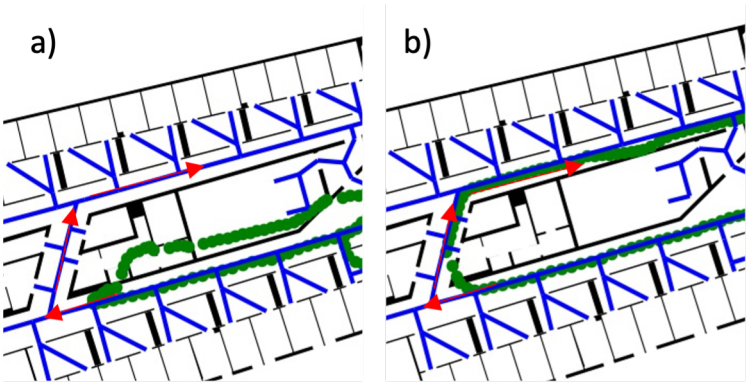



Figure 5.6: Trajectories from the *wl* method (a)), turning back and the *wm* method (b)) cutting the corner; green dots: estimated positions, red arrows: walked path

For the *zerotofour* path, a decrease of the the step correction value to 0.15 m resulted in higher positioning errors, shown in figure 5.7. The *wl*, *ww* and *wm* method, all in combination with the *wr* method, reached a position accuracy of 7.5 m or better in 90 % of the steps. The *wm* and *ww* methods with *wr* support achieved an overall better accuracy than the *wr* and *wl* combination. The combination of the *wl* and *wr* method could enter the elevator, but failed to take last turn at the $4^{th}$ floor. As in the previous test for the *zerotofour* path, at some positions at the missed turn, all particles were invalid. The *wm* and *ww* methods with routing support on the other hand missed the elevator slightly, but reached the last room.

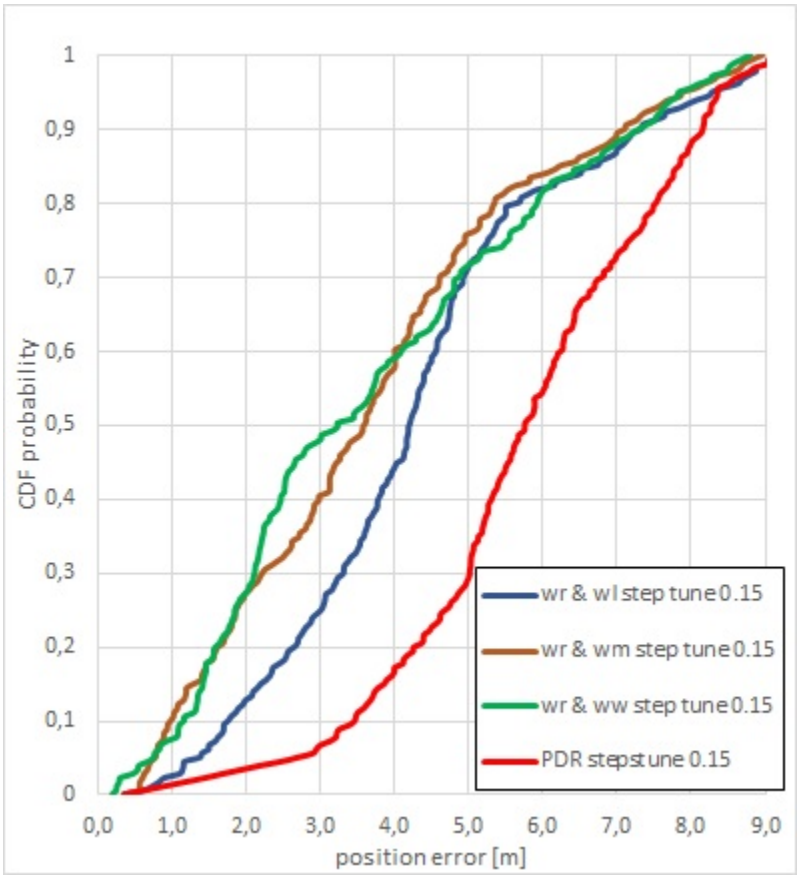



Figure 5.7: CDF of the positioning error of the *zerotofour* path with step length correction of 0.15 m for the bootstrap PF and the PDR with step length correction of 0.15 m.

As it can be seen from the test results, a good step length estimation is important to reach a high accuracy with the bootstrap PF with geospatial analysis. In contrast to the random error of the step length and heading, which is the result of noisy measurements of the IMU, the systematic error of the step length can not be mitigated as well by the PF algorithm. This is especially true for the *zerotofour* path. The *eight* path mainly leads through narrow corridors with many turns, which gives more opportunities to correct for wrongly estimated positions. The *zerotofour* path on the other hand passes the entrance hall in the ground floor as well as in the first floor, providing less opportunities for corrections of the particles' positions, especially in the first half of the trajectory. For this reason, all weighting methods work less well, even with a sufficient correction for the systematic step length error, for the *zerotofour* path. For both paths, the *wm* method in combination with the *wr* method provided the best results overall, regarding the accuracy as well as the determination of the correct room, closely followed by the combination of the *ww* and *wr* method.

## 5.2 Particle Filter with Backtracking

The backtracking PF has been tested with the three different concepts check for line of sight (*cl*)), check for room (*cm*), check for routing (*cr*) for particle checks (as described in section 4.2). The CDF for the position estimate error for the different concepts used for the *eight* path, with a step correction of 0.1 m, is shown in figure 5.8. The *cm* and *cr* method reached an overall similar accuracy, with the *cm* method reaching a accuracy of less than 4 m and the *cr* method reaching a accuracy of just less than 3 m in 90 % of the steps.

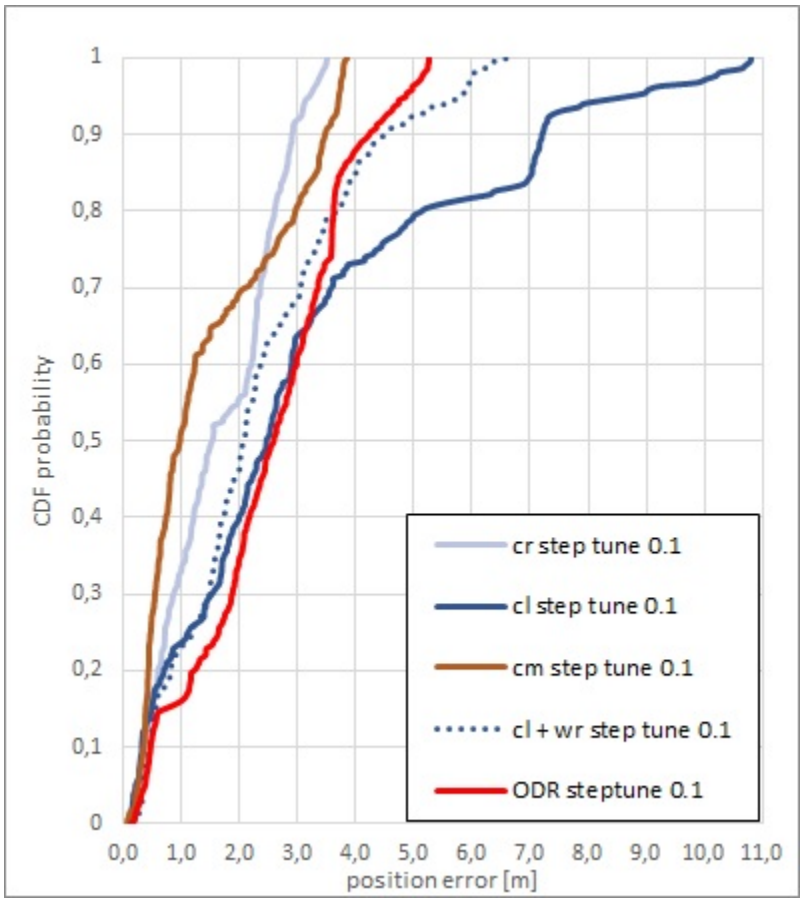



Figure 5.8: CDF of the positioning error of the *eight* path with step length correction of 0.1 m for the backtracking PF and the PDR with step length correction of 0.1 m.

The *cl* method performed significantly worse, this is mainly due to the situation that at a step correction of 0.1 m, enough particles are overshooting at one of the turns, ending up in the ”gallery” that the positions for the next steps is estimated to be on the wrong side of the wall (see figure 5.9.


Figure 5.9: Part of the trajectory of the *eight* path, that wrongly proceeds in the ”gallery”, when using the *wl* method, black arrows indicating the walking direction, the green points representing the position estimates for each step.

At this point, the backtracking functionality can be observed (see figure 5.10). At the end of the ”gallery”, with every subsequent step more particles get deleted because they would cross a wall, resulting in the backtracking algorithm to resample the invalid particles at more plausible corrections (as explained in section 4.3) and by this, changing the trajectory to the correct path.

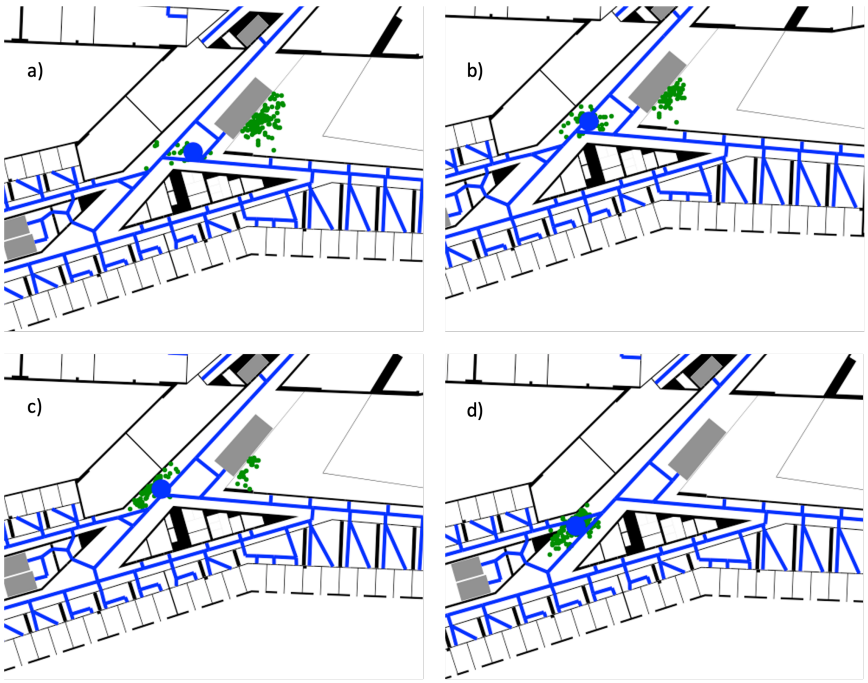



Figure 5.10: Example of the trajectory correction (from (a) to (b)) through the backtracking functionality, the green dots represent the valid, propagated particles, the blue dot is the resulting position estimate

This was also an exception, where the *wr* method helped to significantly increase the accuracy

to less than 4.5 m in 90 % of steps, by correcting the trajectory closer to the next routing edge (see figure 5.11). Since the ”gallery” does not contain any routing edges, the trajectory was directed back into the corridor. The effect of the *wr* method was negligible for the other methods.

Figure 5.11: Effect of the support through the *wr* support on the deviated trajectory.

Figure 5.12 shows the CDF of the position estimate error for the *zerotofour* path with a step correction of 0.2 m. The accuracy for all methods is between 5 and 6 m in 90 % of the steps. The *cm* method can provide an position error of less than 4 m more often than the two other methods. From the three methods, only the *cm* method lead the trajectory into the elevator, while only the *cr* method reached the final room. The *cm* and the *cl* method both lead the trajectory in the neighbouring room, west of the final room.

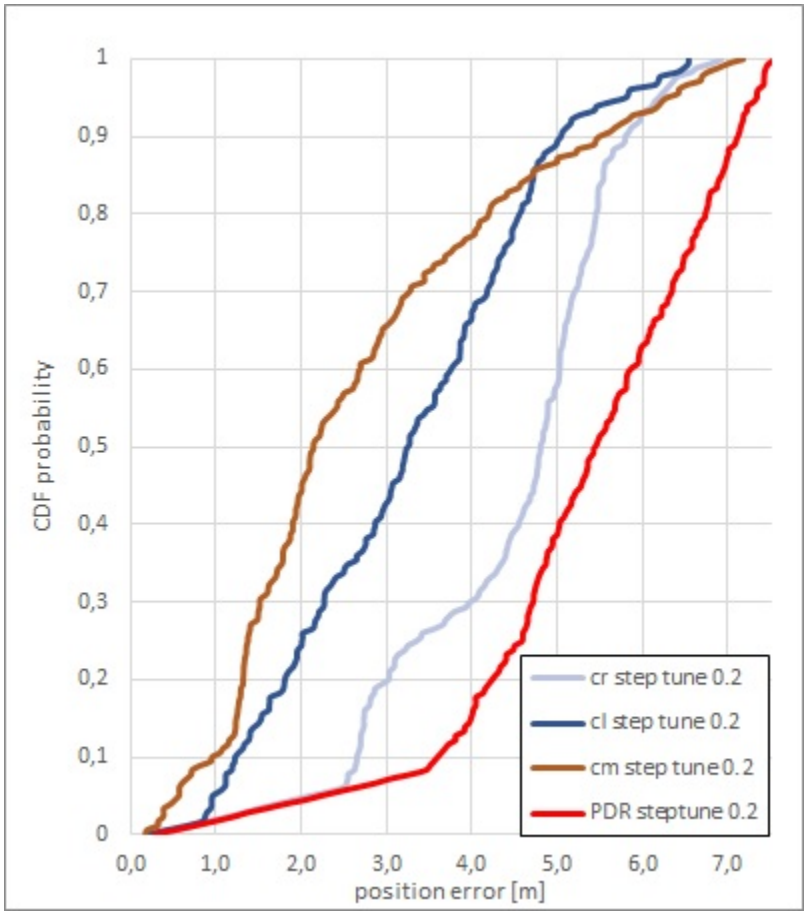



Figure 5.12: CDF of the positioning error of the *zerotofour* path with step length correction of 0.2 m for the backtracking PF and the PDR with step length correction of 0.2 m.

For the *eight* path without step correction, a accuracy better than 3 m can be achieved with the *cm* and *cl* method, while the overall performance of the *cr* method falls behind the two other methods, as shown in figure 5.13. The use of additional routing support (*wr*) even worsens the achievable accuracy of the *cr* method. For the *cm* and *cl* method, the *cr* has only a small influence on the error of position estimate. The *cl* method perform significantly better (also see figure 5.14), mainly, because less particle overshoot at the turn at the eastern corner. This way the trajectory doesn't deviate into the "gallery".

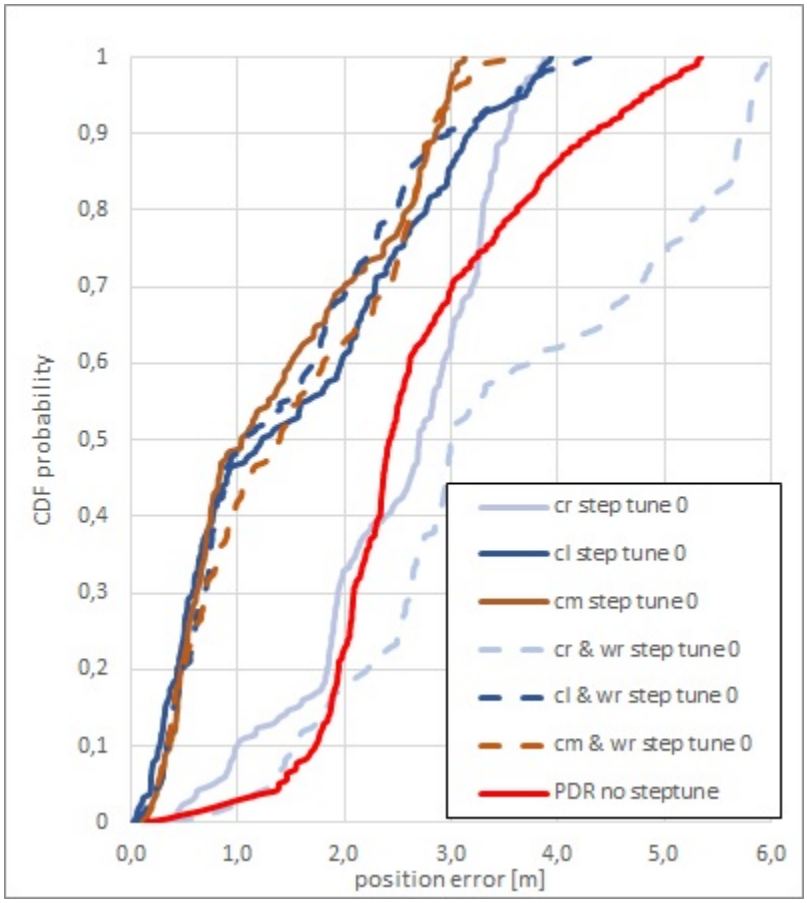



Figure 5.13: CDF of the positioning error of the *eight* path without step length correction for the backtracking PF and the PDR without step length correction.

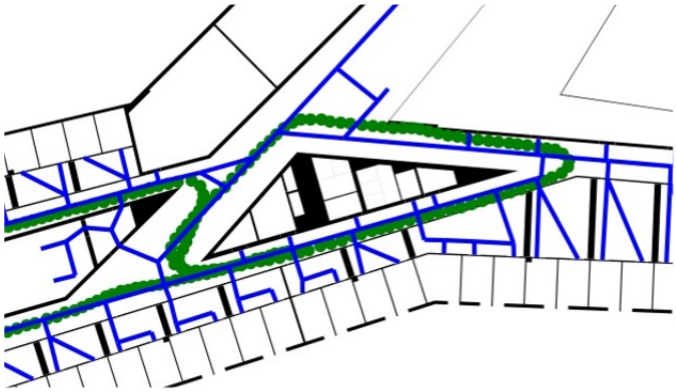


Figure 5.14: Trajectory (green dots) from the *cl* method for the *eight* path without step length correction for the backtracking PF

Figure 5.15 shows the different method's CDF of the positioning error for the *zerotofour* path with a step length correction of 0.15 m. All methods, with and without the additional use of the

*wr* method achieve a positioning error of less than 7 m in 90 % of the steps, while the *cm* and *cl* methods achieve a significantly better accuracy for the positioning than the *cr* method. The *wr* method only has a minor influence on the accuracy of the used methods, slightly improving the *cl* method and slightly decreasing the accuracy of the *cm* method. The elevator could be reached by the *cl* method as well as the *cl* and *cm* methods with routing support (*wr*). It was closely missed by the *cm* method. The *cr* method, was the only one, which reached the destination room, all other trajectories ended in the same neighbouring room.

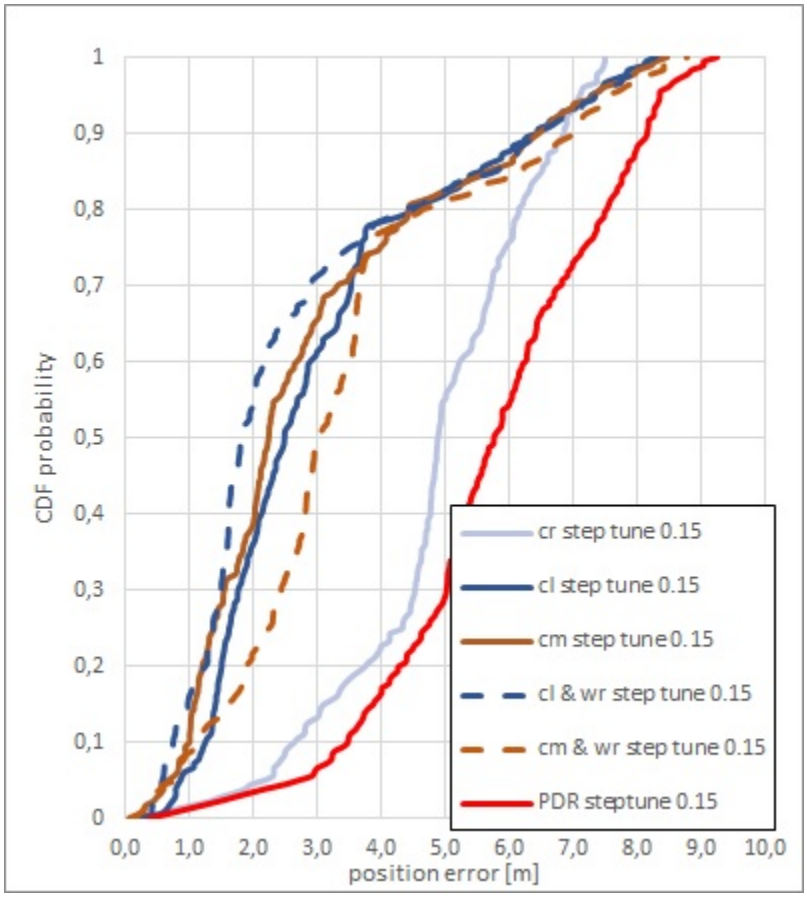



Figure 5.15: CDF of the positioning error of the *zerotofour* path with step length correction of 0.15 m for the backtracking PF and the PDR with step length correction of 0.15 m.

In most cases the *cm* and the *cl* method had the best accuracy. The *eight* path with a step correction of 0.1 m is the exception, here the *cl* method reached a significantly worse accuracy, due to the above mentioned situation, that the trajectory gets ”trapped” in the gallery for a part of the trajectory.

## 5.3 Comparison of Bootstrap and Backtracking Particle Filter

In this section the results for the test of the bootstrap PF and the backtracking PF with geospatial analysis are compared with each other. Since the *wr* method didn't reduce the run speed of the weighting methods in the bootstrap PF, but increased the accuracy of the position estimation in most cases, all weighting methods for the bootstrap PF are considered in combination with the *wr* method for this comparison. Figure 5.16 shows the error CDF for the different methods of the backtracking and the bootstrap PF algorithm for the *eight* path with a step correction of 0.1 m. The bootstrap PF with the *wm* and the *wl* methods achieve the best accuracy, followed closely by the backtracking PF with the *cm* method. Though the *wl* method of the bootstrap PF achieved an overall good accuracy, it missed the last corner (see figure 5.2. The trajectories of the bootstrap PF with the *wm* method and the backtracking PF with the *cm* method are shown in figure 5.16.

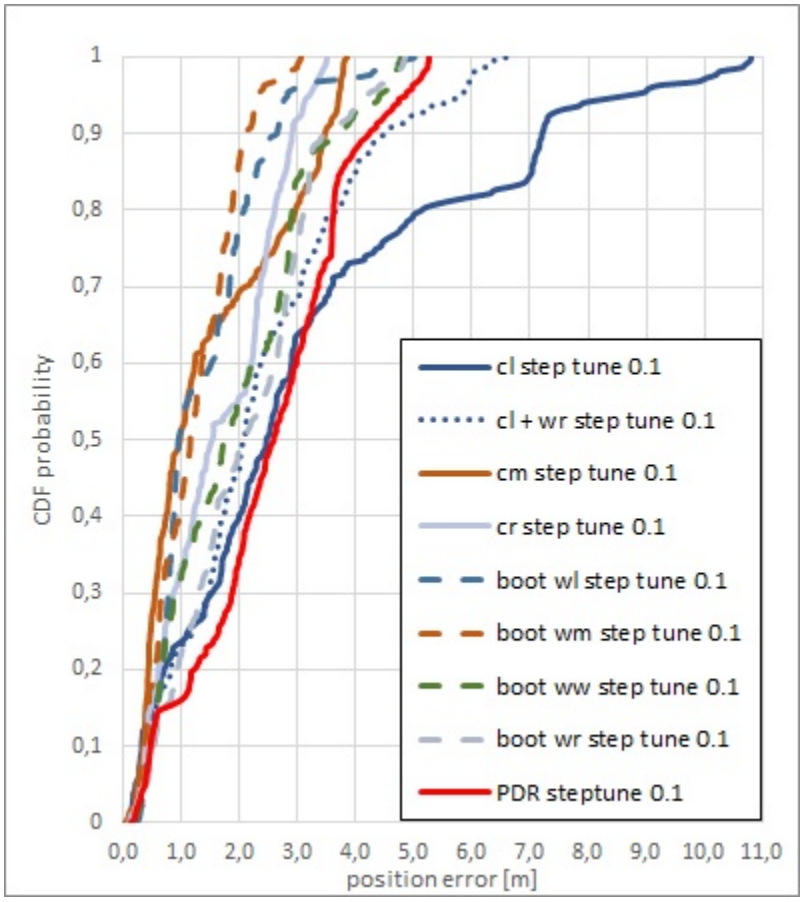



Figure 5.16: Comparison between the CDF of the positioning error of the *eight* path with a step length correction of 0.1 m of the backtracking PF and the bootstrap PF

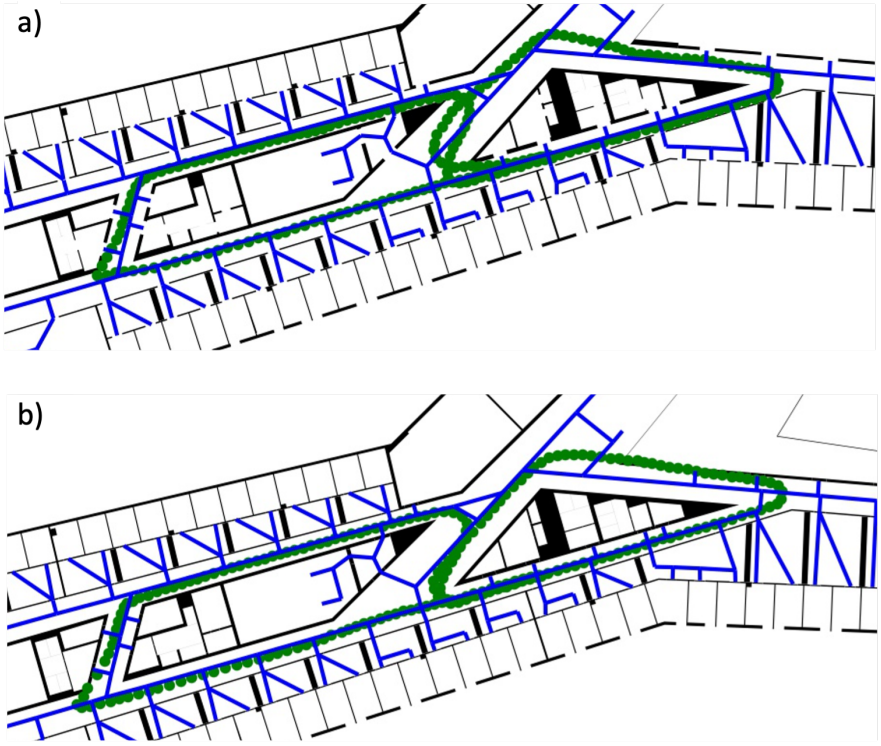



Figure 5.17: Trajectories of the *wm* and *wr* method of the bootstrap PF (a)) and the *cm* method of the backtracking PF (b)) for the *eight* path with a step length correction of 0.1 m

The comparison of the error CDF of the two PF approaches for the *zerotofour* path with 0.2 m step correction is shown in figure 5.18. The *cm* method of the backtracking PF achieves the best accuracy most of the time. It is closely followed by the *cl* method of the backtracking PF and the bootstrap PF methods, which provide overall similar accuracy most of the time. As mentioned in section 3.2.1, the bootstrap PF with *wl* misses the last turn and doesn't come as close to the destination room as the other approaches. Even though the bootstrap PF with *wm* manages to reach the last room, in contrast to the backtracking PF methods, it generally performs less well, including the cutting of the last corner.

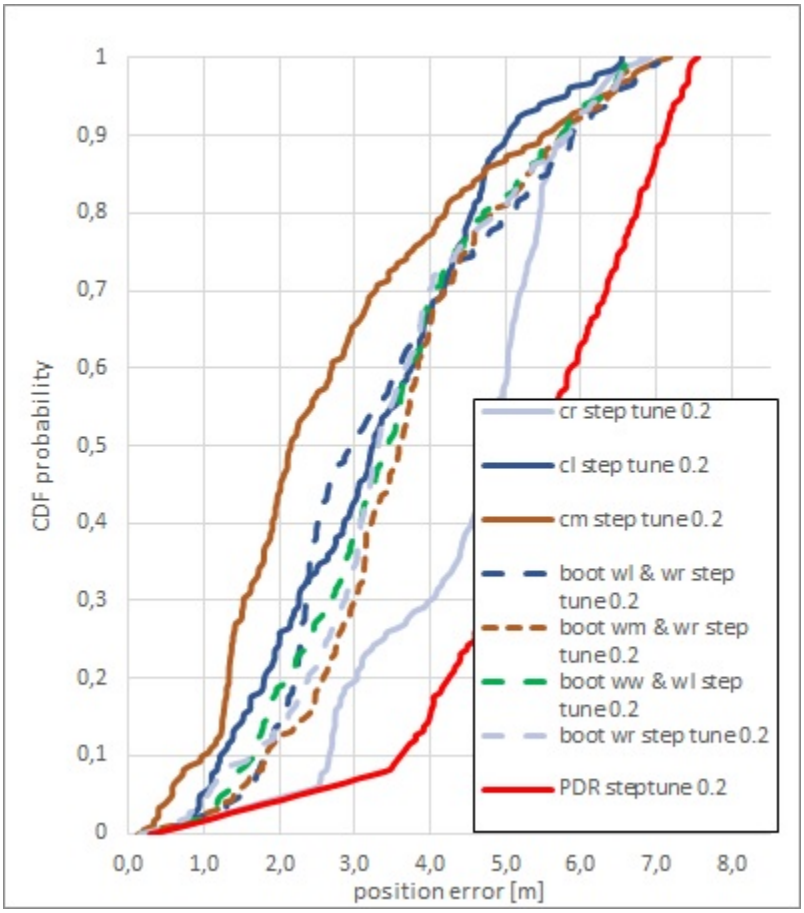



Figure 5.18: Comparison between the CDF of the positioning error of the *zerotofour* path with a step length correction of 0.2 m of the backtracking PF and the bootstrap PF

For the *eight* path without step correction, the backtracking PF with the *cm* method achieved the overall best accuracy, followed by the *cl* method and than the bootstrap PF with the *wm* and *wr* method, as can be seen in figure 5.19. The *wm* method of the bootstrap PF could achieve a accuracy of less than 2 m less often than the two mentioned backtracking PF methods, but errors of more than 2.5 m occurred less often.

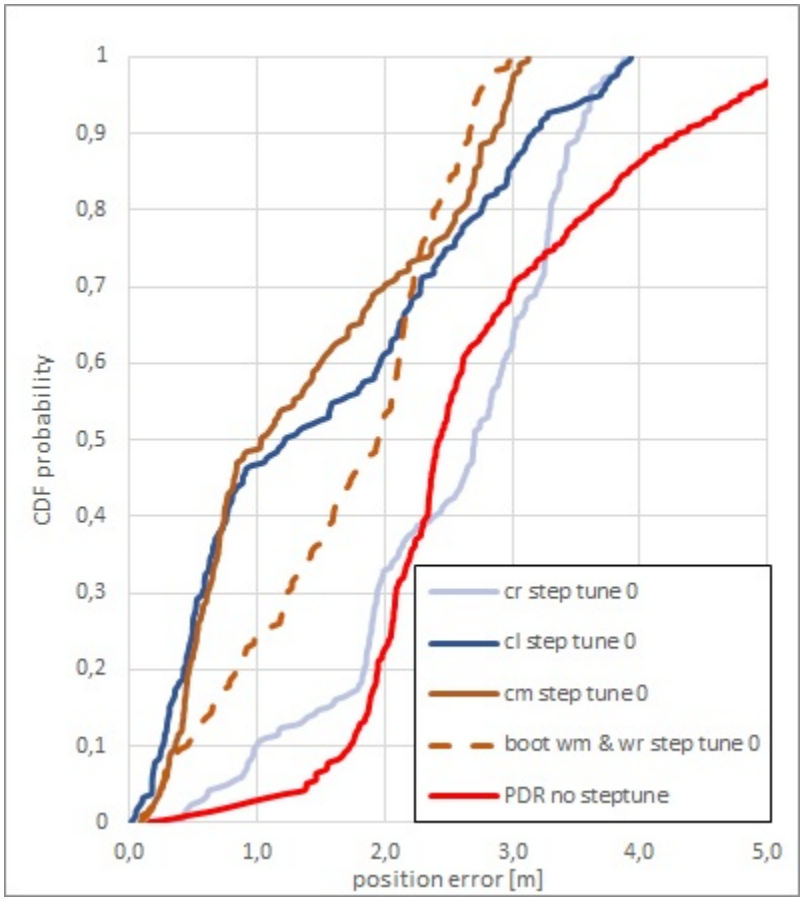



Figure 5.19: Comparison between the CDF of the positioning error of the *eight* path without step length correction of the backtracking PF and the bootstrap PF

For the *zerotofour* path with a step length correction of 0.15 m, the *cl* and the *cm* method of the backtracking PF could provide the best accuracy, with only minor differences to each other. While the difference between these two backtracking PF methods and the bootstrap PF methods *wl* and *wm* (plus *wr*) is pronounced for the occurrence of errors less than 4.5 m, the difference in the number of occurrences of higher error values is considerably smaller.

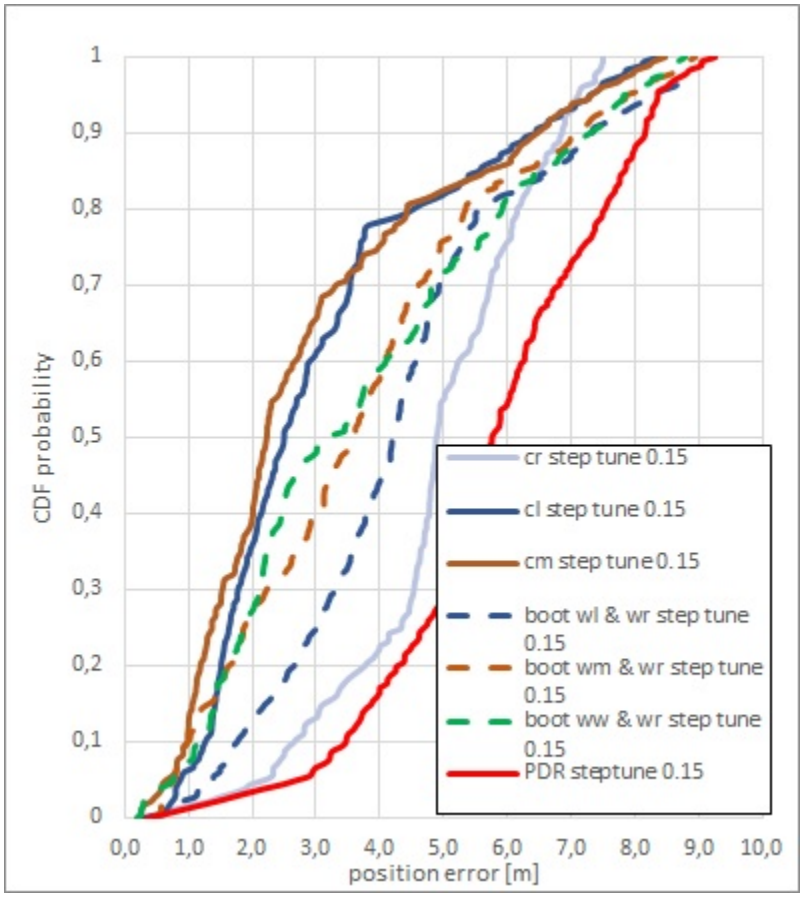



Figure 5.20: Comparison between the CDF of the positioning error of the *zerotofour* path with a step length correction of 0.15 m of the backtracking PF and the bootstrap PF

Altogether, the backtracking PF with the *cm* and *cl* method provides slightly better results than the bootstrap PF methods, with the *cm* method being a little bit more robust, though the differences are often small. But even in the cases, where the bootstrap PF algorithms show very similar or even a slightly better accuracy, the backtracking PF methods tend to detect the position in specific rooms slightly better, such as the lift or the final room in the *zerotofour* path, as seen in 5.21 where the trajectories of the *zerotofour* path of the bootstrap PF with the *wm* method (a)) and the backtracking PF with the *cm* method (b)) are shown. In the backtracking PF, at least one position is located inside the elevator and the last steps resemble more the actual walked path, even though it enters one room too early. The jumps in the trajectory of the backtracking PF are the result from the deletion of many particles when a floor change is detected and only particles in elevators and stairs are considered valid. If many particles have been deleted, the next position estimate can deviate more than the estimated step length from the previous one.

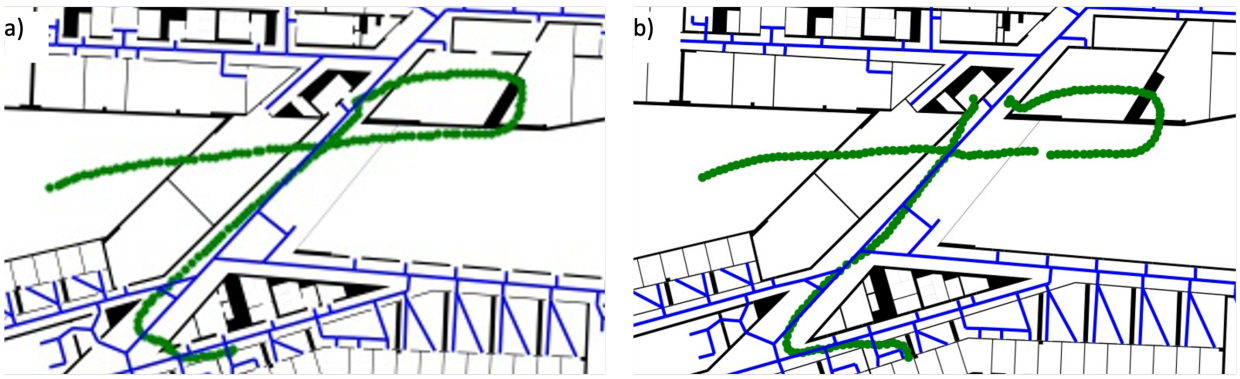



Figure 5.21: Comparison between the trajectories of the bootstrap PF with the *wm* method (a)) and the backtracking PF with *cm* method (b)). Green dots represent the estimated positions at each step

Even though the *cl* method of the backtracking PF is temporary deviating from the walked route for the *eight* path, the effect of the backtracking algorithm, correcting the wrong path, shows the potential robustness of the backtracking PF.

# 6 Conclusion and Outlook

In this thesis, a bootstrap and a backtracking PF algorithm have been developed that use geospatial analysis to determine the plausibility of the particles based on building information. The goal was to develop a PF algorithm that provides a meter scale position accuracy for indoor navigation without the use of GPS or infrastructure assistance and allows the use of building information in the GeoJSON format.

For this, first the underlying concepts and methology were explained and related work was presented. Then, the concepts as they were further developed and adjusted for this thesis have been explained, including the underlying mathematics, followed by the detailed explanation of the implementation in the algorithms of the two PF approaches. Finally the used data set was presented and the results of the bootstrap and the backtracking PF with geospatial analysis have been investigated and discussed.

For the bootstrap PF several methods for the determination of the particles' weights were developed, namely the weighting by line of sight (*wl*) approach, weighting by rooms (*wm*) approach, the weighting by walls (*ww*) approach and the weighting by routing (*wr*) approach. For the backtracking PF algorithm three methods were developed, that determine invalid particles based on different building information, namely the check for line of sight (*cl*), check for room (*cm*) and check for routing (*cr*) method.

The results of the different methods of both PF approaches have been tested with the data from two paths through the HafenCity University (HCU) building and have been compared in regards to the attainable position accuracy as well as the ability to detect certain rooms.

For the bootstrap PF, the *wm* method provided the overall best results, while the *wl* method was less reliable in the *eight* path. The *cm* and *cl* of the backtracking PF reached a slightly better precision than the approaches of the bootstrap PF, with the *cm* method being a bit more robust. The backtracking PF approaches admittedly needed more computation power, resulting in significantly longer run times than the bootstrap PF methods, but they were better in detecting estimating position to be in specific rooms (e.g. the elevator or the final room in the *zerotofour* path). Altogether it was possible to develop two PF algorithms that could reach an accuracy of 2 to 7 m in most test cases for indoor navigation, by only using building information, without any infrastructure or GPS support, if an sufficient step length estimation model is used for the determination of the used step lengths.

There is still potential for improvement, especially for situations, where large hallways provide only little possibilities for corrections. Further, the accuracy developed PF approaches is deendent on the quality of the provided map information, resulting in better performances the better the map material is. Of all the tested methods *cm* method of the backtracking PF approach might be favorable, not only because it's performance regarding the position precision and room accuracy, but also because additional information for the accessibility of rooms can relatively easily be implemented. An broken elevator could for example easily be declared an invalid room.

The weakness of both PF algorithms on the one hand is the sensibility to systematic errors in the step length estimation and on the other hand open spaces, that provide little information for corrections. The quality of the step length estimation could be improved either by scaling the step length every time an absolute position can be attained with high precision and accuracy, either through infrastructure support or when a position is determined with relatively high confidence regarding their correctness, for example when using an elevator change the floor. On the other hand machine learning algorithms could be used for a more precise step length estimation. Another option would be to use data sets of step length in relation to other biometric information from a huge number of people. But these are costly and hard to attain.

Because of the relatively simple and modular code structure, it is possible to add or exchange methods and functionalities to both PF approaches that have been developed in this thesis. This would also include the additional use of position estimation from 5G antennas, as implemented in [24]. Since the already calculated step length and heading is used for the PF algorithms, their usage is not limited for the use by pedestrians, but they could also, for example be implemented for the indoor navigation of robots or any other moving object, as long as travelled distance (which can easily be calculated from the speed) and heading is provided.

# BIBLIOGRAPHY


[1] Paramvir Bahl and Venkata N Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *Proceedings IEEE INFOCOM 2000. Conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064)*, volume 2, pages 775–784. Ieee, 2000.

[2] Moustafa Youssef and Ashok Agrawala. The Horus location determination system. *Wireless Networks*, 14(3):357–374, 2008.

[3] Jaewoo Chung, Matt Donahoe, Chris Schmandt, Ig-Jae Kim, Pedram Razavai, and Micaela Wiseman. Indoor location sensing using geo-magnetism. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 141–154, 2011.

[4] Niklas Karlsson, Enrico Di Bernardo, Jim Ostrowski, Luis Goncalves, Paolo Pirjanian, and Mario E Munich. The vSLAM algorithm for robust localization and mapping. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 24–29. IEEE, 2005.

[5] Stephen P. Tarzia, Peter A Dinda, Robert P Dick, and Gokhan Memik. Indoor localization without infrastructure using the acoustic background spectrum. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 155–168, 2011.

[6] Hossein Shoushtari, Thomas Willemsen, and Harald Sternberg. Many ways lead to the goal—possibilities of autonomous and infrastructure-based indoor positioning. *Electronics (Switzerland)*, 10(4):1–17, 2021.

[7] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A Reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp'12*, pages 421–430, 2012.

[8] Changhao Chen, Peijun Zhao, Chris Xiaoxuan Lu, Wei Wang, Andrew Markham, and Niki Trigoni. Deep-Learning-Based Pedestrian Inertial Navigation: Methods, Data Set, and On-Device Inference. *IEEE Internet of Things Journal*, 7(5):4431–4441, 2020.

[9] Renaud Dubé, Daniel Dugas, Elena Stumm, Juan Nieto, Roland Siegwart, and Cesar Cadena. SegMatch: Segment based place recognition in 3D point clouds. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5266–5272, 2017.

[10] Elzbieta Lewandowicz. Network models of 2d and 3d carastral data. In *Environmental Engineering. Proceedings of the International Conference on Environmental Engineering. ICEE*, volume 9, page 1. Vilnius Gediminas Technical University, Department of Construction Economics & Property, 2014.

[11] Chun Yang, Thao Nguyen, and Erik Blasch. Mobile positioning via fusion of mixed signals of opportunity. *IEEE Aerospace and Electronic Systems Magazine*, 29(4):34–46, 2014.

[12] Jiaqi Yang, Zhiguo Cao, and Qian Zhang. A fast and robust local descriptor for 3d point cloud registration. *Information Sciences*, 346:163–179, 2016.

[13] Sachini Herath, Hang Yan, and Yasutaka Furukawa. Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, amp; new methods. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3146–3152, 2020.

[14] Neil J Gordon, David J Salmond, and Adrian F M Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113, 1993.

[15] Chi Ming Esmond Mok, Chung Ming Lau, L Xia, G Retscher, and H Tian. Influential factors for decimetre level positioning using ultra wide band technology. *Survey Review*, 44(324):37–44, 2012.

[16] Reza Zekavat and R Michael Buehrer. *Handbook of position location: Theory, practice and advances*, volume 27. John Wiley & Sons, 2011.

[17] Jörg Blankenbach, Abdelmoumen Norrdine, Hendrik Hellmers, and Eduard Gasparian. A novel magnetic indoor positioning system for indoor location services. In *Proceedings of the 8th International Symposium on Location-Based Services*, pages 1–11, 2011.

[18] 3gpp. Release 17. https://www.3gpp.org/release-17. Accessed: 2021-07-28.

[19] Adrián Cardalda García, Stefan Maier, and Abhay Phillips. *Location-Based Services in Cellular Networks: from GSM to 5G NR*. Artech House, 2020.

[20] Thomas Willemsen. *Fusionsalgorithmus zur autonomen Positionsschätzung im Gebäude, basierend auf MEMS-Inertialsensoren im Smartphone*. PhD thesis, HafenCity Universität Hamburg, 2016.

[21] Catia Real Ehrlich and Jörg Blankenbach. Indoor localization for pedestrians with real-time capability using multi-sensor smartphones. *Geo-Spatial Information Science*, 22(2):73–88, apr 2019.

[22] Widyawan, Martin Klepal, and Stéphane Beauregard. A Novel Backtracking Particle Filter for Pattern Matching Indoor Localization. In *Proceedings of the First ACM International Workshop on Mobile Entity Localization and Tracking in GPS-Less Environments*, MELT '08, pages 79–84, New York, NY, USA, 2008. Association for Computing Machinery.

[23] Chuanhua Lu, Hideaki Uchiyama, Diego Thomas, Atsushi Shimada, and Rin-ichiro Taniguchi. Indoor positioning system based on chest-mounted IMU. *Sensors*, 19(2):420, 2019.

[24] Hossein Shoushtari, Cigdem Askar, Dorian Harder, Thomas Willemsen, and Harald Sternberg. 3D Indoor Localization using 5G-based Particle Filtering and CAD Plans. IPIN, 2021. accepted (preprint).

[25] Christopher Cappelli. *The Language of Spatial Analysis*. Esri Press, 2013. https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/books/the-language-of-spatial-analysis.pdf, Accessed: 2021-07-01.

[26] Donna Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 21:66–113, 10 1984.

[27] GeoPandas developers. Geometric manipulations. https://geopandas.org/docs/user_guide/geometric_manipulations.html. GeoPandas documentation, Accessed: 2021-07-02.

[28] Sean Gills. The Shapely User Manual. https://shapely.readthedocs.io/en/latest/manual.html#binary-predicates, 2021. Accessed: 2021-07-02.

[29] Jan Wendel. *Integrierte Navigationssysteme*. Oldenbourg Verlag, München, 2011. doi: 10.1524/9783486705720, isbn: 9783486704396.

[30] Prianka Aggarwal, Zainab Syed, Naser El-sheimy, and Abeoelmagd Noureldin. *MEMS-Based Integrated Navigation*. Norwood : Artech House, 2010.

[31] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[32] Thomas Willemsen, Friedrich Keller, and Harald Sternberg. Kartengestützte MEMS-basierte Indoor- positionierung mittels Partikel Filter. In *Oldenburger 3D-Tage 2015*, 2015.

[33] Thomas Willemsen, Friedrich Keller, and Harald Sternberg. A topological approach with mems in smartphones based on routing-graph. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–6. IEEE, 2015.

[34] Franco P Preparata and Michael I Shamos. *Computational Geometry – An Introduction*. Springer Verlag, 1988.

[35] Grinbergnir. Codeproject - is a point inside a polygon? http://www.codeproject.com/Tips/84226/Is-a-Point-inside-a-Polygon. Accessed: 2021-07-20.