Integrating Orthogonal Localization Methods for Smartphone-based Indoor Localization

Bachelor Thesis

Arno Schmetz

RWTH Aachen University, Germany

Chair for Communication and Distributed Systems

Advisors:

Dipl.-Inform.	Jó Ágila Bitsch Link
Prof. Dr.-Ing.	Klaus Wehrle
Prof. Dr.	Bernhard Rumpe

Registration date: 2012-12-12
Submission date: 2013-04-12

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 12. April 2013

Kurzfassung

In dieser Arbeit wird ein Ansatz präsentiert, mehrere Methoden der Lokalisierung in Gebäuden zu kombinieren. Durch die Nutzung einer Gewichtungsfunktion bei der Zusammenführung der Positionsabschätzungen, die von den verschiedenen Mechanismen bereitgestellt werden, kann dynamisch entschieden werden, welche Informationen und Abschätzungen besser sind als andere. Dadurch wird die resultierende Abschätzung der Position verlässlicher und robuster gegenüber äußeren Einfüssen und Konfigurationen einzelner Mechanismen.

Das vorgestellte System ist erweiterbar und kann einfach durch weitere Regeln für die Gewichtung verbessert werden. Auch können Filter einfach realisiert werden inklusive complexer Definitionen oder definiert durch eine Vorbehandlung der Kartendaten beispielsweise im Hinblick auf wichtige Orte oder verbotene Bereiche. Durch die Nutzung dieses Systems konnten wir eine durchschnittliche Verbesserung der Genauigkeit von über 30 % und zudem die Robustheit deutlich erhöhen.

Abstract

In this thesis, we will present an approach of combining several indoor localization mechanisms. Using a weighting function when combining the location estimates provided by the mechanisms we can dynamically decide which information are more useful than others. By that the localization becomes more reliable and resilient to environmental influences and configurations of a single mechanism.

The system we defined is extensible and we can easily enhance it by more rules regarding the weighting. We can implement filters easily including fuzzy queries or a preprocessing of the map concerning points of interest or resticted areas for example. Using this system we achieved a mean improvement of the accuracy of more than 30 % and a significant increase of the robustness.

Brevium

Scriptione preasentamus ideam combinando methodes plura in loco personae aestimando. Definimus functionem auctoritas de combinando aestimandes ab methodes positione aestimando. Eis possumus iudicare informationes provisa dynamice. Hoc facit aestimandes resultes melior robustisque de configurationes pertinendus orbisque ad methodes soles.

Definimus systema amplificatum simplex posit in formae functione auctorita. Filtris etiam facentur includens definitiones cerritus tabula processusque de punctem pensos sicut loces nefastisque. Per systema melioratio mediam accurationem de 30 % pervenimusque melius significans robur.

Acknowledgments

First of all I want to thank Jó Ágila Bitsch Link for supervising and supporting me. We had a lot of fun and he has been motivating me continuously during my work. Furthermore I want to thank Prof. Klaus Wehrle for the chance of working at his chair for computer science and Prof. Bernhard Rumpe for having agreed to be my second examiner even before I told him the precise topic of my thesis. And thanks to Felix Gerdsmeier allowing me to use the dCollector source code. And thanks to Gustav Geier.

Prof. Peter Rossmanith and Felix Reidl from the Theoretical Computer Science Group I want to thank for a great practical course in navigation and intensifying my interest in navigation, routing and localization problems.

Many thanks to my parents for enabling me to study of computer science at this great university and making my way.

Last but heartily I want to thank my friends being available for test runs, having fun and interest in the topic. Thank you Maren, Domme, Eva, Martin, Marlin, Svenja and Adrian.

Contents

1

Introduction

In outdoor environments the Global Positioning System (GPS) provides a reliable localization service. Farmers, the military, car navigation and other location-based services use GPS. Inside buildings the reliability of GPS is poor. Thus, in the past years indoor localization has been an active research topic. All of the localization services are using some sensors and have advantages and disadvantages.

Today's smartphones do have a great set of sensors that can be used for those methods. But most approaches lack in quality of service when having disturbed sensors, malfunctions or even need an exact configuration. In this thesis we want to show a way to combine multiple methods to improve robustness and accuracy for indoor localization. This is sketched in Figure 1.1[*]. The set of modules combines the individual estimates and can provide a good estimate of the location on the underlying map.

We will use markov chains for formal definition of the transition matrix. To improve the qualitiy we define a dynamic weighting function for combining the estimates regarding their accuracy and quality.

1.1 Structure of the thesis

This thesis is structured as follows:

In the upcoming chapter we will explain the background of the techniques we use. We will need them in later chapters and for some basic definitions. We will focus on

[*]For some pictures the rights are reserved:
©Apple Inc. (camera icon (fist on the left side), Wifi (third on the right))
©Pavement Coating (steps (second on the left side))
©Mobiltec Ltd. (satellite (third on the left side))
RFID picture is licensed under GFDL by Kalinko (top centered)
OpenStreetMap logo is licensed under CC-BY-SA by Ken Vermette.

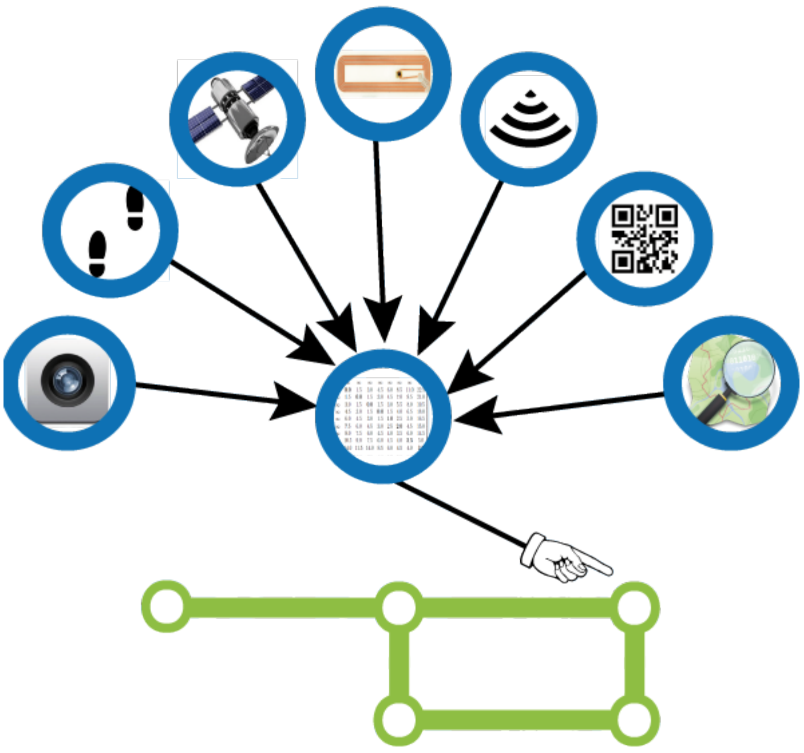


Figure 1.1 Sketch of the principle. The main idea is to let systems like QR-codes, GPS, etc. work on their own, but bring their results together and get a good quality in localization.

the Android and OpenStreetMap system first and then get to know a set of typical sensors that actual smartphones use. After that we will consider Mobile Tagging. At the end we will give a formal definition of markov chains.

In the third chapter we will present several concepts for indoor localization. They will cover step detection inlcuding FootPath and PLATIN, radio-frequency and fingerprinting-based systems. Furthermore we will give a short overview of other approaches including optical, geo-magnetical, ultrasonic and infrared systems. We will adapt and implement a set of those approaches in the following chapters.

In chapter four we will focus on our design. First we will define our adaptions of approaches. Then the common representations of location estimates are presented including ways for transforming them into the same representation. Furthermore we will define a static and a dynamic weighting function bringing the estimates together and weighting them regarding information like accuracy, sensor quality and others.

In the fifth chapter we will sketch the implementation. We use three different applications. The dCollector, originally developped by Felix Gerdsmeier, we will adjust and use. The second application is a simple Wifi Wardriving App for Android, called WiFiWar. We will use it for collecting Wifi fingerprints as defined in the thesis. The main application we present is the EXtensible Indoor Localization Emulator (EXILE) which can use collected data sets and emulate the behavior of the implemented modules. We will use this application for the evaluation of our work.

The evaluation chapter shows some statistics and provides information about the results of nearly 1,500 emulation runs. We will discuss the results and give a conclusion of the system presented in this thesis.

A summary of the thesis and the results will be given in the last conclusion chapter which will finalize the thesis. The last chapter will further provide some ideas for future work and problems that have still to be handled.

2 Background

In this chapter we will present some basics we will use for this thesis. First we will introduce the open source map platform OpenStreetMap, then mention the Android platform for mobile devices and typical built-in sensors of actual smartphones. In the end we will show some mathematical backgrounds we will use in later chapters for formal definitions.

2.1 OpenStreetMap

OpenStreetMap[1] is a platform where users are mapping the world in a collaborative way. All data is available under a special open source license which allows everyone to use, edit and distribute it. This makes the map data often more recent than the data from other sources like Google or TomTom as shown in Figure 2.1. In that figure an area of Aachen is shown, where some years ago the city has transformed a sports field into a residential district ("Am Gutshof"). The data of Google still shows the sports field while OpenStreetMap provides detailed information of the area. Users can export the data as an XML-encoded graph using a visual online representation tool or as dumps[2] providing maps of countries or the whole world at once. It will cover streets, buildings, areas and indoor environmental information.

For visual representation of the map data users developed rendering engines like Osmarenderer, Maperative and Mapnik. To reduce the rendering effort OpenStreetMap provides pre-rendered parts of the map called tiles. Special tile servers provide those tiles at different zoomlevels. For the zoomlevel n the map of the whole world is subdivided into 2^{2n} tiles. Mapnik for example provides 19 different zoomlevels. [3]

To edit or analyze OSM files users can use the Java OpenStreetMap Editor (JOSM). Using this the users can directly add or change nodes, edges or their parameters like street types and indoor tags. The extensibility of JOSM allows users to install plugins providing more features. After editing a map JOSM can directly upload the edited OSM file to the database updating contemporary.[4]

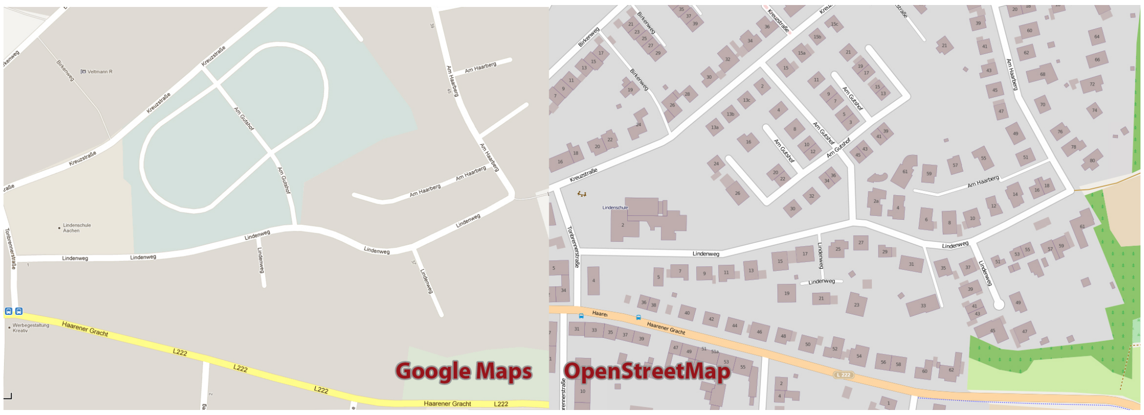


Figure 2.1 2009 in Haaren (small district in the northeast of Aachen) a sports field was transformed into a residential area. Google's maps are still outdated in that area while OpenStreetMap even has shapes of buildings in this area that are still under construction.[5] (state of january 2013)

2.2 Android

The Open Handset Alliance provides the open source software platform Android which allows to develop applications using the Java programming language. Smartphones, tablets, netbooks, gaming consoles[6] and even cameras[7] use this platform.

While Apple contributes only iOS bound to their own devices with programming restrictions, Android on the other hand is way more flexible for pogramming and has different phone models from different companies. Furthermore figure 2.2 shows that the Android platform is one of the most important software platforms for smartphone development.

2.3 Sensors

Sensors are special instruments that are designed to measure certain physical or chemical properties of the environment. These properties can be electromagnetic radiation (like WiFi, visible light or cosmic microwave background), forces (like acceleration or gravity) and others. The measurements are transformed into a signal being a qualitive or quantitive representation of the property. Current smartphones do have a set of sensors only made for measuring but also a variety of sensors for communication like WiFi-antennas. As mentioned before, Android allows us simply to use both kinds of sensors for measuring our environment.

2.3.1 Accelerometer and Gyroscope

Physically acceleration is a directed and vectorized quantity describing the variation of a linear velocity. In systems with a constant mass the acceleration can be determined by the ratio of force and mass. Formally:

$$\vec{a} = \frac{\vec{F}}{m}$$

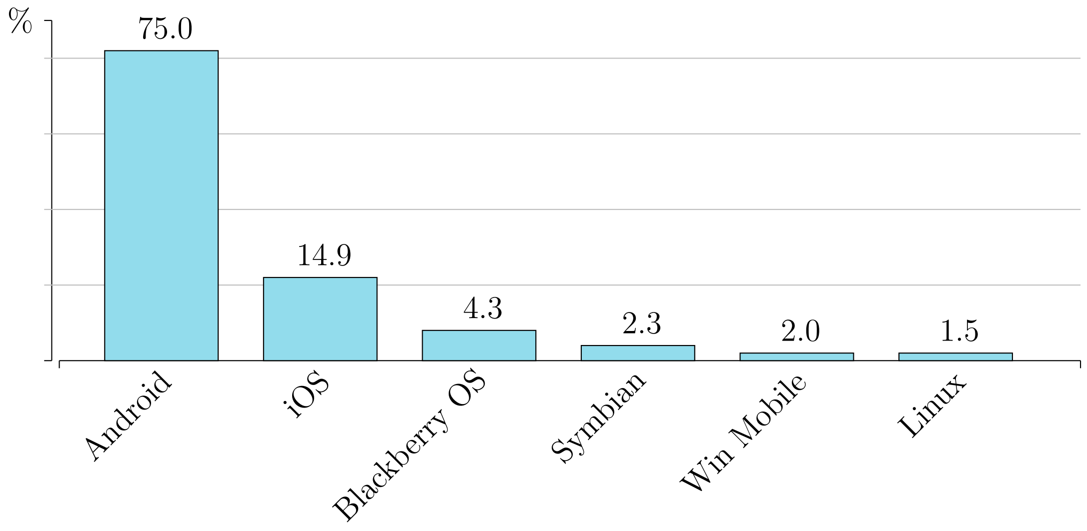


Figure 2.2 Market share of smartphone operating systems of the third quarter 2012 (sales) [8]

We have to differentiate between a static and a dynamic force that is having effect on the mass. The static force can be the negligible varying gravity of the earth pulling everything to the ground. Moving the sensor is a dynamic force that can be translated into a digital representation of the movement.

Gyroscopes are sensors similar to accelerometers. While accelerometers measure linear accelerations the gyroscope is doing this in terms of rotation velocities. Unlike the accelerometer the gyroscope does not need to be lowpassed which has a negative effect on the resolution when motion occurs to determine the orientation of the device. Therefore gyroscope and accelerometers are often used in combination to get a higher accuracy.

2.3.2 Compass

Classical compasses work by having an as freely moving as possible metal needle. That needle will align itself to the earth's magnetic field and point to the magnetic northpole. Nowadays they usually use the Hall effect that describes the observable behavior of current-carrying conductors being placed in a static magnetic field. Sensors based on the Hall effect allow us to determine the orientation of the whole device in respect to all three orientation axes. This makes the compass an easy instrument for determining the orientation of the device but due to the dependency of a magnetic field this sensor can be easily distracted by a magnet close to the sensor.

2.3.3 Camera

Nearly every smartphone today has an integrated camera. Technically these cameras convert optical images to electronic signals using special image sensors. Charge-coupled devices (CCD) are passive pixel sensors. When light hits the sensor it holds the light as an electrical charge which can be converted into digital representations by additional circuits shifting the information row-wise. In contrast there are

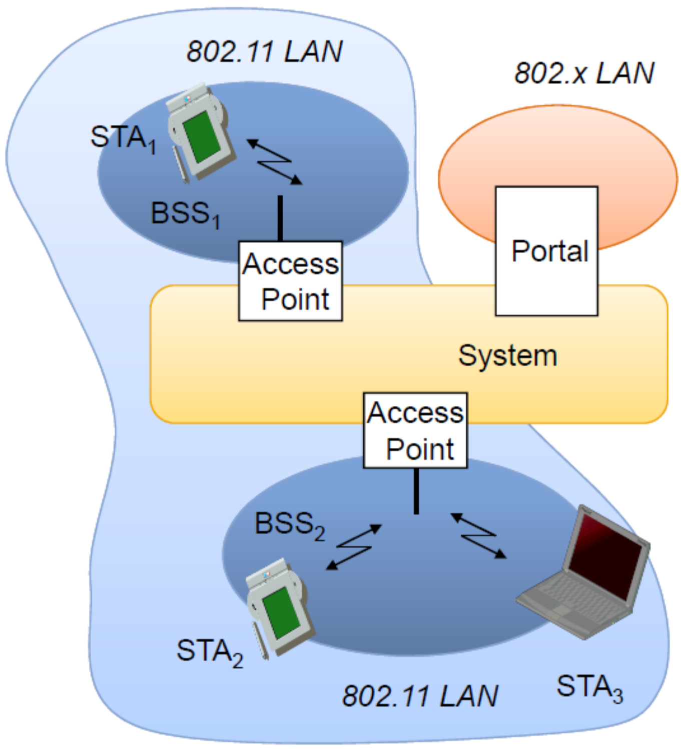


Figure 2.3 A sketch of an inrastructure Wifi network. STA$_i$ are mobile stations, BSS$_i$ are the basic service sets. The stations are connected to the access points granting access to other networks.[9]

complementary metal-oxide semiconductors (CMOS) as active pixel sensors. They directly detect the light and digitalize the information inside the sensor. This allows simultanuous data extraction and direct adressing of pixel regions. Both technologies have advantages and disadvantages and products on the market do use both of them. The fast transformation of light to a digital representation allows us to use this as a sensor for detecting patterns like codes with a smartphone camera.

2.4 Wifi

The IEEE 802.11 standard defines the communication, codings and algorithms used for wireless data communication of Wifi. For that electro-magnetic signals are used. They usually use carrier frequencies around 2.4 GHz and 5 GHz subdivided in several channels for multiplexing. In difference to ethernet Wifi has to deal with more complex interferences of the signals, multipath propagations and dynamic bit rates. The sent out beacons of Wifi also state their names and additional identification data. Using Wifi users can set up an infrastructure network providing access points being attached to an existing network like ethernet. Alternatively users can set up an ad-hoc network directly connecting mobile stations or mesh networks that interconnect access points directly instead of using a wired connection.

Figure 2.3 shows a sketch of a Wifi infrastructure network. The mobile stations like smartphones and laptops can connect to the access point that is managing every station in the reception range. The basic service set is the group of stations within an access point transmission range. The corresponding basic service set identifier is usually equal to the access point's unique MAC adress. To enable users an easy recognition of Wifis the service set identifier can be configured by the provider and therefore is not unique. Mobile devices usually display the service set identifier for selecting a Wifi hiding the corresponding access point. Multiple access points can have the same service set identifier and hide handovers of access points to the user. [9]

2.5 Global Positioning System

GPS is a global postioning system using a set of satellites. These satellites send out their current position and time continuously. It allows user localization with an accuracy of some meters down to centimers. Stationary GPS stations can expand the system. GPS is hosted by the USA but there are alternatives in construction like GLONASS (Russia), COMPASS (China) and GALILEO (Europe). This system works well outdoors in populated countries but can not be used for indoor localization or around the poles. By supplying a special infrastructure this system may be used indoors being called a pseudolite system.[10]

2.6 Mobile Tagging

Using mobile tagging allows a contributor to generate special tags like barcodes, QR-codes, DataMatrix and more. These tags encode some data like strings, numbers, icons or web links. This leads to a physical world connection enabling real world objects to have a connection to digital ressources via these tagging codes.[11]

2.6.1 QR-codes

One of the most common tagging codes is the QR-code (Quick Response code). It stores data in a two-dimensional matrix of squares. It is able to encode digits, alphanumeric characters, bytes or kanji characters with a fixed maximum capacity. The specification[12] includes orientation-independency, selectable error correction levels and version information. Figure 2.4 shows an example of such a code. QR-codes itself underly no patent rights but the word “QR-code” is a trademark of Denso Wave Inc.[13]

2.6.2 RFID and NFC

A RFID (Radio-Frequency Identification) system consists of a transponder and a reading device. The transponder can be active and therefore have a power supply. The reader creates a changing electro-magnetic field that supplies passive transponders with power. The transponders do not create their own electro-magnetic fields

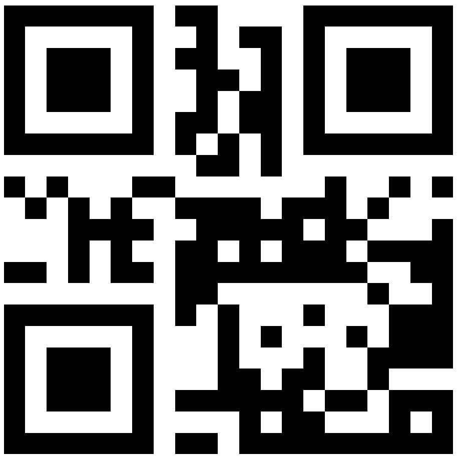


Figure 2.4 The string “COMSYS” encoded as QR-Code

but influence the reader’s one. By that the reader can read and send data and instructions to the transponder. They do work from frequencies of several kHz up to some GHz.[14]

Near Field Communication (NFC) is an international standard for data exchange over short distances. It usually uses the RFID standard for the technical details. Micro-payment services, tickets or even the current german identity card are using NFC systems. By that NFC-tags can be defined and used for mobile tagging. This standard has been finished a short while ago.[15]

2.7 Markov Chains

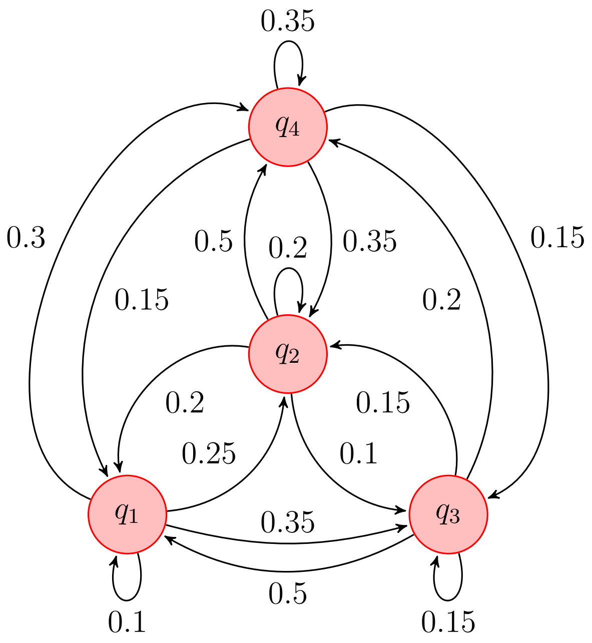
Markov chains are stochastic processes that help us to model the evolution of a system using probabilities for transitions. The markov chains itself are sequences of random variables $X_1, X_2...$ with possible values of a countable set of discrete states $Q = \{q_1, q_2...q_i\}$. These variables have to comply with the *markov property* defining that for every state the past and future states are completely independent.[16] Formally:

$$P(X_t = q_{j_t} | X_{t-1} = q_{j_{t-1}}, X_{t-2} = q_{j_{t-2}}, ..., X_0 = q_{j_0}) = P(X_t = q_{j_t} | X_{t-1} = q_{j_{t-1}})$$

Furthermore we have a function describing the probability p_{ij} of the chain moving from state q_i to state q_j. This probability must not depend upon the previous states of the chain and is called the *markov property.*

Definition 2.1 (Transition Probabilities)**.** *The transition probabilities $p_{ij}(t)$ for time t are defined as:*

$$p_{ij}(t) := P(X_{t+1} = q_j | X_t = q_i),\ \ i, j \in \{1...n\}$$



$$\begin{array}{c} \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{array}\begin{array}{c}\begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \end{array}\\ \begin{pmatrix} 0.1 & 0.25 & 0.35 & 0.3 \\ 0.2 & 0.2 & 0.1 & 0.5 \\ 0.5 & 0.15 & 0.15 & 0.2 \\ 0.15 & 0.35 & 0.15 & 0.35 \end{pmatrix}\end{array} = \Phi$$

Figure 2.5 The transition probabilities can be intuitively described as a directed graph, also called transition diagram (left) or as a transition matrix (right)

This function of transition probabilities leads to a definition of a matrix representation.[17]

Definition 2.2 (Transition Matrix)**.** *With given transition probabilities as defined in definition 2.1, the definition of a transition matrix is:*

$$\Phi_t := (p_{ij}(t)) = \begin{pmatrix} p_{11}(t) & \dots & p_{1n}(t) \\ \vdots & \ddots & \vdots \\ p_{n1}(t) & \dots & p_{nn}(t) \end{pmatrix}$$

Another way of representing this function is a directed graph also called *transition diagram*. Figure 2.5 shows a simple stochastic process represented as a matrix and as diagram.[18]

3 Related Work

In this chapter we will present approaches related to our work. In the beginning we will present Wifi fingerprinting which is trying to match scans during a walk with fingerprints collected in advance. After that we will mention FootPath that is using step detection to track a user walking along a known path. PLATIN uses step detection too but does not need a known path and uses probabilistic models. Pseudolites try to imitate the GPS system indoors needing to install a special indoor infrustructure. Other approaches we will present in brief are using optical sensors, geo-magnetism, ultrasonic or infrared light.

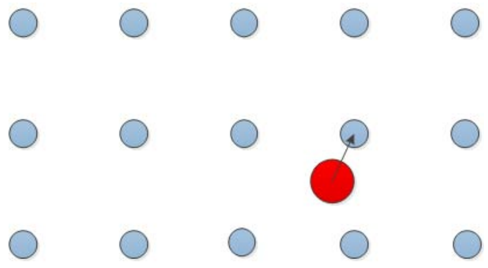
3.1 Wifi Fingerprinting

The idea of fingerprinting is to collect data describing the environment to enable the recognition concerning this place. Humans do this automatically for example using their sensoric abilities. In short time they detect a visual overview over a room, the sounds, smells, movements and temperature. These samples enable the human being to recognize this room later on. This system will now be adapted using Wifi (cf. 2.4) to fingerprint the environment. There are several measurable metrics users can use for Wifi fingerprinting but we will focus only on the **Received Signal Strength Indication (RSSI)** that we can measure in decibels using the formula

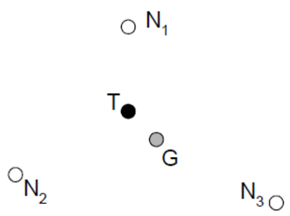
$$x = 10 \cdot log_{10} \left(\frac{P_1}{P_2} \right)$$

where P_1 is the received signal power and P_2 a reference power.

Those samples will have to store the Basic Service Set Identifier (BSSI) which is pseudo-unique per base station and the corresponding RSSI. We can use the euclidean distance, given by



(a) Illustration of matching a point to its nearest neighbor in space. [19]



(b) The real position (T) can be approximated best by the guess (G) of kNNSS than by matching to one of the neighbors (N_i) in this case. k is set to three in this example.[20]

Figure 3.1 Illustrations of NNSS and kNNSS

$$d_E(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

to compute the distance of two points in a n-dimensional signal space, where $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ are n-dimensional vectors.

In the beginning there is a training phase in which users collect Wifi samples for reference points and store them in a database. In the positioning phase later on the mobile station calculates a new sample and uses the data of the database for localizing using different patterns. One pattern is the matching of the **Nearest Neighbor in Signal Space (NNSS)** which is illustrated in figure 3.1(a). One can improve this by the **k Nearest Neighbors in Signal Space (kNNSS)** which determines the k nearest neighbors and calculates a guess which can be more precise than the NNSS matching which can be seen in figure 3.1(b).[19]

Factors like the number of reference and access points, grid spacing and more are strongly limiting the estimated accuracy of this fingerprinting approach. Furthermore the environmental conditions may change over time. Human bodies built up mostly of water and walking around can absorb parts of the Wifi signals. Furthermore isotropic radiators only exist in theory letting the orientation of a device influence the received signal strength.[9]

3.2 FootPath

FootPath was initially developed at the Chair for Communication and Distributed Systems at RWTH Aachen University by Paul Smith and Jó Bitsch. It is designed to track the progress of walking users along a path for indoor navigation. Step detection is the basic instrument for FootPath. To detect the steps the accelerometer (cf. 2.3.1) is used. Figure 3.2 illustrates how a step is detected using sensor data. To determine the direction the person is walking the system uses the compass (cf. 2.3.2).

The detected steps are matched with the map for the path the person is walking on. Due to different step sizes of people and long edges in the graph representation of

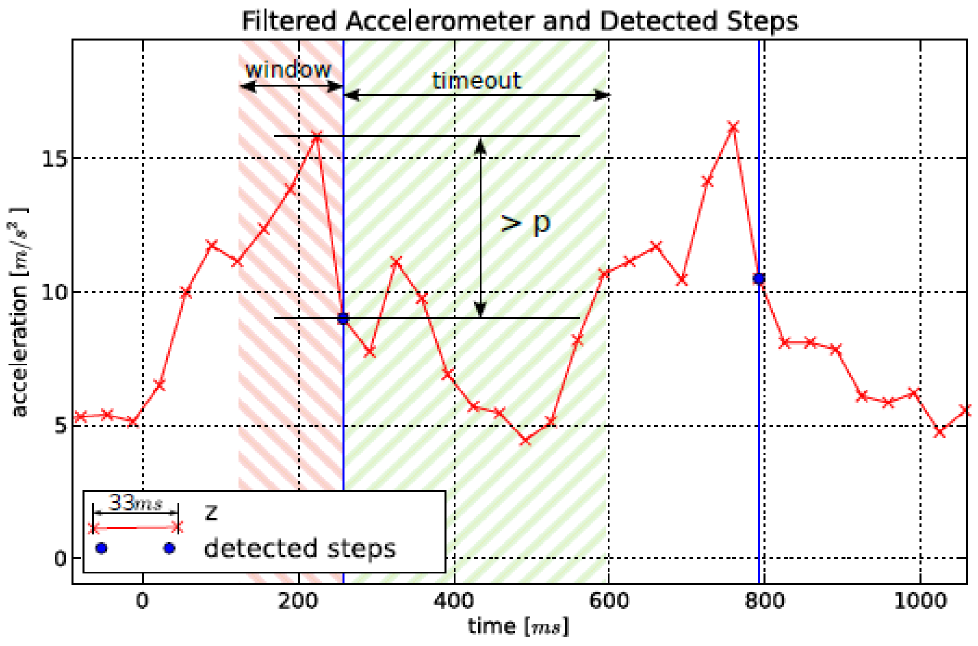


Figure 3.2 If a difference of at least p occurs on the (low pass filtered) z-axis of the accelerometer a step is detected. This has to occur during a window (red dashed area). Furthermore to avoid false detections there is a timeout (blue dashed area) after every step.[21]

a map the way has to be subdivided into smaller pieces the steps will be matched with. Now there are expected steps based on the map and path data and detected steps that have to be matched. Therefore FootPath creates a penalty matrix and updates it with every step. The penalties are calculated by comparing the expected and measured angles (orientation) as well as the distance to the expected location. The system adds the penalties in columns to a former initialized matrix D_0.

$$D_1 = \begin{pmatrix} 0.0 & \infty & \infty & \dots \\ \infty & 1.0 & & \\ \infty & 3.5 & & \\ \infty & 5.5 & & \\ \vdots & \vdots & & \end{pmatrix} \rightsquigarrow D_2 = \begin{pmatrix} 0.0 & \infty & \infty & \dots \\ \infty & 1.0 & 4.5 & \\ \infty & 3.5 & 1.0 & \\ \infty & 5.5 & 4.5 & \\ \vdots & \vdots & \vdots & \end{pmatrix} \rightsquigarrow D_3 \rightsquigarrow \dots \rightsquigarrow D_m$$

Figure 3.3 The penalty matrix gets filled columnwise with entries for every detected step.

Figure 3.3 shows a typical evolution of the penalty matrix over time. We can describe the expected position after j steps by

$$p = \underset{1 \le i \le n}{\operatorname{argmin}}(d_{i,j})$$

where n is the number of rows of D_j and p the count of map steps representing the expected position on the observed path.

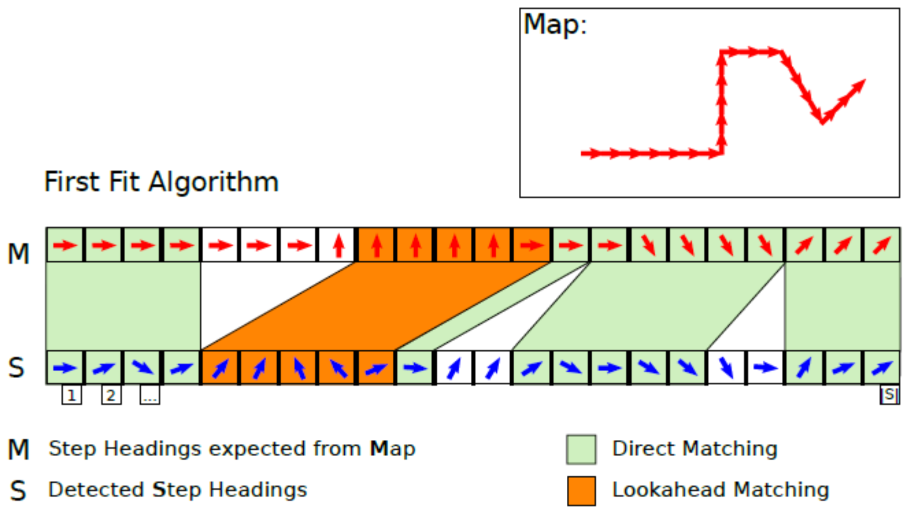


Figure 3.4 Sequences of expected and detected steps using First Fit for matching. Steps can be matched directly or if this fails the the lookahead matching is used.[21]

Because the expected steps are not always exactly the steps that one can detect or have the same quantity the user needs for a path there are algorithms for how to match detected with expected steps. The **First Fit** method assumes that the detected steps heading directly correspond to the edge's direction. Using this assumption this method uses the *Direct Matching Mode* as long as the headings match the edge's heading. If this fails for some consecutive steps the *Lookahead Matching Mode* is used searching for a match of the headings at the neighbored edges. It corrects the actual positioning right before switching back to the direct matching mode. Figure 3.4 shows an illustration of such a matching.

Another possibility is the **Best Fit** method which relates the matching problem to the String-To-String Correction Problem[22]. It is using the above defined matrix D_m to match the best fitting step. This lowest value of the column corresponding to the step is representing the best fitting step. A variation is the **Best Fit Traceback** which is not starting at the position $d_{1,1}$ but at the last position matrix D is representing and iteratively tracing back the optimal edit distances in terms of the String-To-String Correction Problem.[21]

FootPath does not need any infrastructure to be provided by the building except a map of it. By that FootPath is independent and provides a good precision in tracking a person indoor. on a known path Actual improvements even detect movements of wheelchairs precisely. For that smartphones can use the camera positioned to the ground recording a video. The system then extract the motion vectors of the compressed video to detect the movement speed and direction.[23]

3.2.1 PLATIN

PLATIN, short for (P)robalistic (L)ocalization by (A)nalyzing (T)imed Map States for (I)ndoor (N)avigation, was developed by Felix Gerdsmeier less than a year ago. Instead of the FootPath approach for tracking the progress of a walk of an user, Gerdsmeier described a probalistic way for tracking.

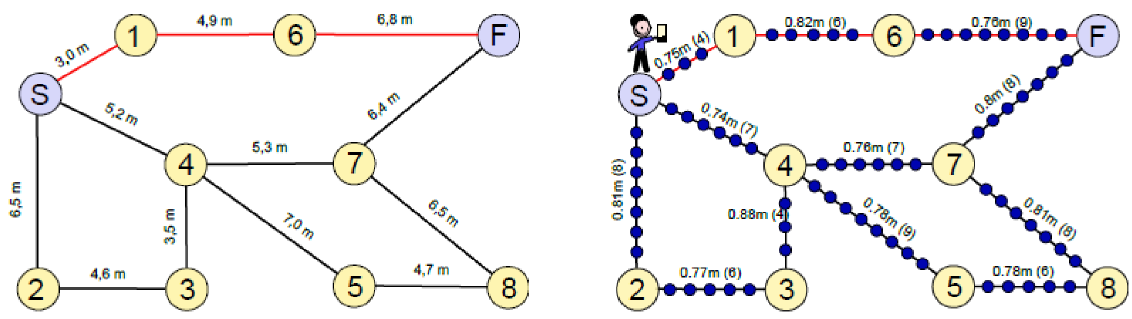


Figure 3.5 The figure on the left shows an example map with the edges' lengths. The right figure shows the map after creating virtual steps. The edges' labels contain the steplength for this edge and the resulting number of steps for this edge. The base step length assumption for this splitting was 0.8 m.

In PLATIN the map has to be refined first taking into account the edges not having the length of a step. Therefore those edges get some virtual steps as can be seen in figure 3.5. After this adjustment Gerdsmeier gives some basic definitions.

Definition 3.1 (Map State). *Let $G = (V, E)$ be an undirected graph containing nodes $\{n_i\} \in V$ and edges $\{e_j\} \in E$. The map state $\Psi^{(t)}$ at time t is a set of probabilities with*

$$\Psi^{(t)} = \{p(n_i)\}, \forall n_i \in V$$

This way the map states define the probabilities of an user located at a certain position at a certain point in time.

Definition 3.2 (k-Neighborhood). *Let $G = (V, E)$ be defined as before in definition 3.1. Then the definition of the k-neighborhood $N_k(n_i)$ of the node n_i is given by:*

$$\begin{aligned} N_k(n_i) \quad = \quad & \{n_j \in V \mid \exists \; path\; p^* = (n_i, \dots, n_j) \; with \; |p^*| = k \\ & \wedge \; \forall p_l = (n_i, \dots, n_j) : \min |p_l| = k\} \end{aligned}$$

In addition to this definition there is a definition for the degree of the neighborhood.

Definition 3.3 (Degree of Neighborhood).

$$deg(n_i, n_j) = k \Leftrightarrow n_i \in N_k(n_j)$$

In simple words the k-Neighborhood is a set of nodes that are reachable from the node n_i by exactly $k-1$ hops. In FootPath there is a definition of a scoring function taking into account the detected and expected direction of a step. In PLATIN Gerdsmeier refines this function reversely and without using a zero to avoid divisions by zero in later computations. The PLATIN scoring function is given by

Definition 3.4 (PLATIN Scoring Function).

$$score(\alpha, \beta) = \begin{cases} 20.0, & if\ \measuredangle(\alpha, \beta) \leq 45° \\ 10.0, & if\ 45° < \measuredangle(\alpha, \beta) \leq 90° \\ 5.0, & if\ 90° < \measuredangle(\alpha, \beta) \leq 120° \\ 1.0, & otherwise \end{cases}$$

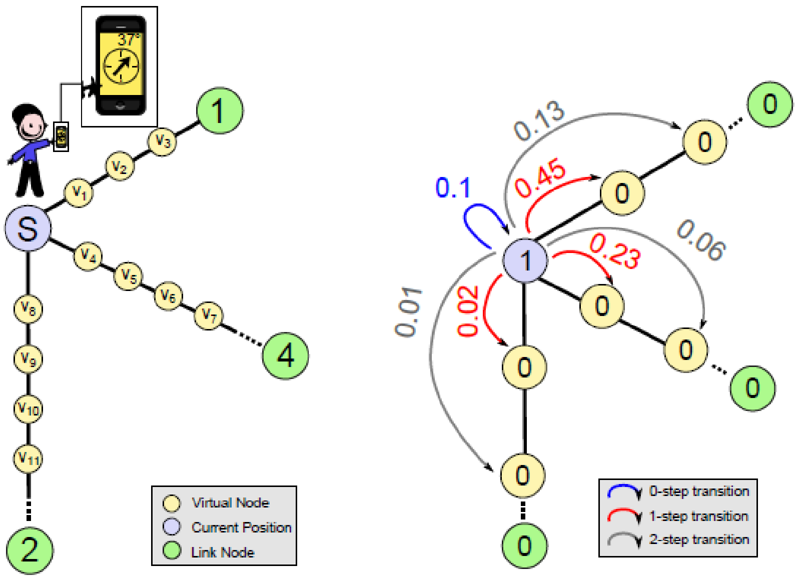


Figure 3.6 The left figure shows a graph including the current position (S), other nodes (1,2,4) and the virtual steps computed in between (v_i). The user's compass detects a bearing of about 37°. The right figure shows the propability propagation that was given in definition 3.5. The transitions are computed for 0, 1 and 2 steps. With a probability of about 50% the user's position has changed to v_1.

Now it is possible to compute the transition probabilities.

Definition 3.5 (PLATIN Probability Transition)**.**

$$prob_{ij}(t) = p_{step}(k(i,j)) \cdot \frac{score(\alpha_{ij}, \beta)}{\sum_{n_l \in N_{k(i,j)}(n_i)} score(\alpha_{il}, \beta)}$$

where $p_{step} = (p_i)_{0 \leq i \leq 2}$ describes the probabilites for performing i steps. N denotes the entire sequence of bearings and $\beta := N(t)$ the next compass bearing. α_{ij} with $k(i,j) := deg(n_i, n_j) \leq 2$ describes the direction from n_i to n_j, $\measuredangle(n_i, n_j)$. By that there is a function describing transition probabilities based on the last position, the detected direction of a step and a maximal distance the probabilities are computed on. To make this work now a known starting position is needed. For node n_i being the starting position one can set up the first map state by:

$$\Psi^{(0)} = (p(n_1), \ldots, p(n_i) \ldots, p(n_k)) = (0, \ldots, 1, \ldots, 0)$$

The figure 3.6 shows an example of the system computing the probability propagation and giving a good estimate on the user's new position. In this example the bearing could be matched well to the underlying graph representation. If the compass bearing had been exactly between the bearing of the edges to nodes 1 and 4 the system would have problems giving the right estimate and maybe targets the wrong edge.[24]

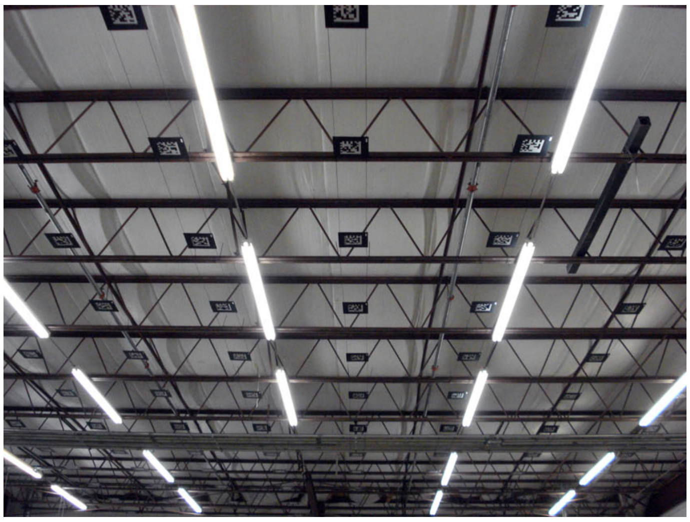


Figure 3.7 Tagging codes at the ceiling in combination with cameras scanning them provides a precise locating service.[27]

3.3 Pseudolites

Installing pseudolites is another way of indoor localization. These pseudolites are little systems imitating the functionality of a GPS satellite (cf. 2.5) at their installed position. This system can adapt most of the algorithms and implementations based on GPS but multipath propagation and exact time synchronization problems have to be solved for precisely locating a receiver at its position. A similar system in reasearch bases on the so-called *repealites* being a combination of pseudolites and standard GPS. The repealite stations repeat the GPS signals in the indoor environment providing good accuracy.[25] The problem of those systems is the need for installing additional infrastructure at high costs. [26]

3.4 Optical Systems

There are also localization methods using optical systems. The simplest version can be implemented by providing a dense structure of visual codes (cf. 2.6) telling the device its current location when scanning. One popular system is the **Sky-Trax** system which is based on tagging codes at the ceiling and cameras (cf. 2.3.3) scanning them like shown in figure 3.7.

Another example of optical systems is the **Camera and Laser Indoor Positioning System (CLIPS)** which uses lasers of different colors and shapes installed on a fixed laser hedgehog. Those lasers can be scanned by a mobile camera. The camera now computes the relative position to the hedgehog by using knowledge about the installation and the scan. Figure 3.8 illustrates this method.[26]

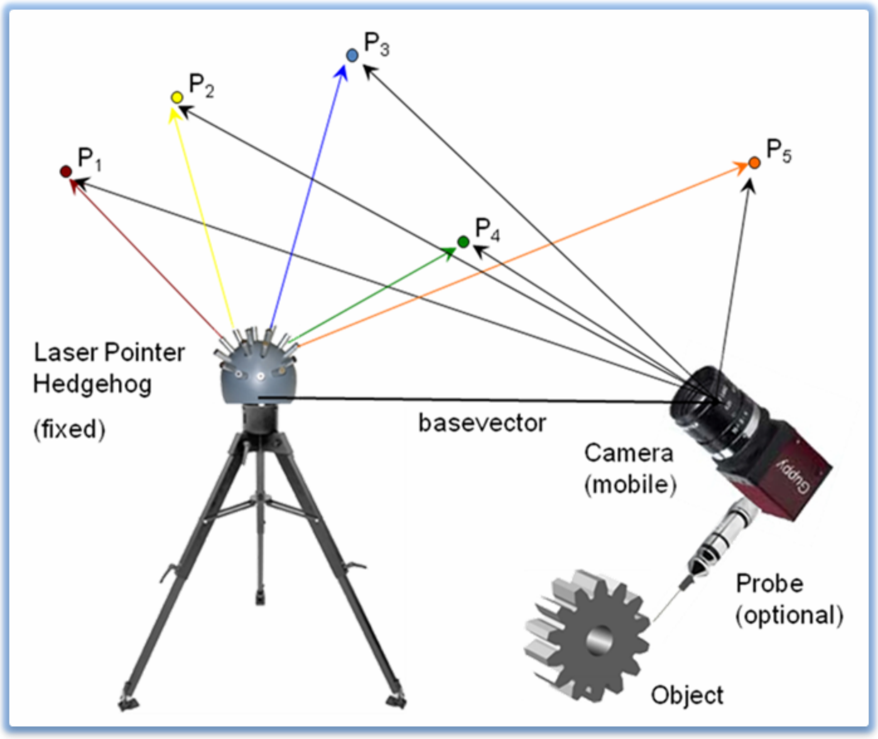


Figure 3.8 The principle of CLIPS is the laser hedgehog sending laser rays in different colors and shapes a mobile camera can scan and by that computing its own position relative to the laser pointer hedgehog.[28]

3.5 Other Approaches

There are multiple other approaches available and in research. Furthermore a set of existing systems is using different spectrums of the radio frequencies like the infrared spectrum as used in **iGPS** by providing rotating IR-planes. Other systems rely on sound like ultrasonic as used in **Active Bat**. Futhermore there are approaches based on the fingerprinting of the **geo-magnetism**[29] to localize the user.[26]

3.6 Combining Step Detection and Wifi localization

In [30] Kothari et al present an approach of combining step detection and Wifi fingerprinting. They initially create a dense Wifi fingerprinting map using pioneer robots that automatically move along a path creating and storing Wifi fingerprints. Figure 3.9 shows an illustration of the architecture. The particle filter combines the information and calculates the position estimate.

Alonso et al [31, 32] focus on the human activities in their work. They define a human activity and posture recognition which can determine wether a person is standing, walking or sitting. Further, they do not use a typical map of the environment but define discrete zones regarding the area. Based on that they define an automata for posture and position transitions.

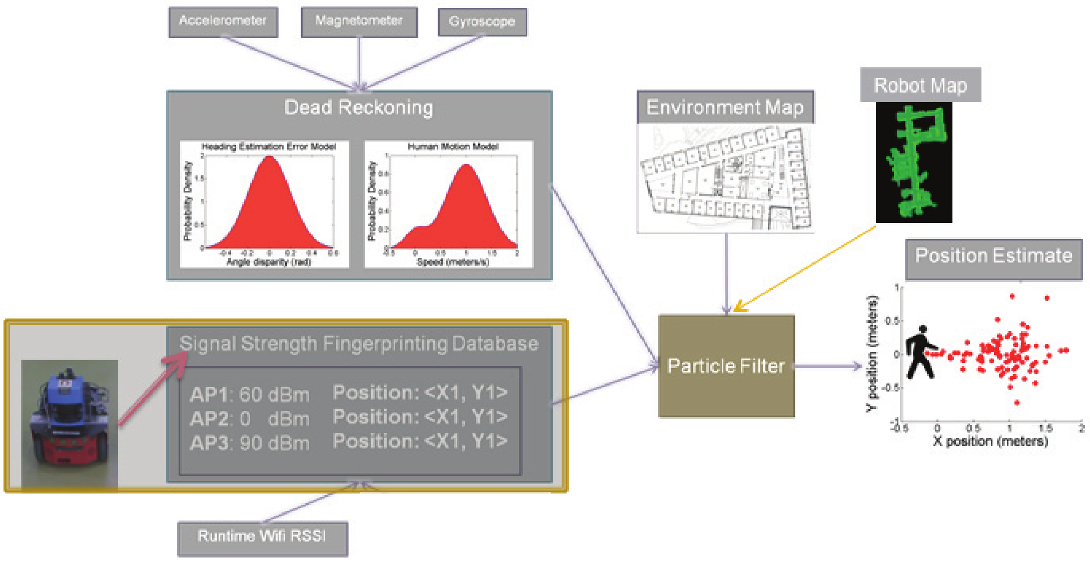


Figure 3.9 The step detection (Dead Reckoning) and Wifi fingerprinting an environment map and the robot map provide information about environment and detections. The particle filter combines these information and determines a position estimate.[30]

3.7 Conclusion

In this chapter we presented several approaches of localizing an user indoors. Many systems need pre-installed infrastructure, others need a database or knowledge how to interpret or compare measured data to provide estimates on the user's position. Some systems can be adapted for nearly every area and others differ in accuracy based on the environment. In difference to outdoor positioning there is no dominant system for indoor tasks so far. However, it is a very active research domain.

4

Design

In this chapter we will present how the different methods of localization are combined in a formal theoretical way. For that we will use Markov chains and apply them using different algorithmic patterns. The basic data structure comes from the map data as set of nodes and edges building a map graph. For the localization, we construct the basic structure of a transition matrix. In general we assume the map to be small in relation to the whole world. This allows us to use some simplifications like the linear approximation of GPS coordinates instead of using spheric geometrics.

The basic idea of this thesis is to let several localization and positioning systems work independently and combine their estimates afterwards. For our design, we will use an approach of step detection and a transition matrix for the basic structure and extend it by other methods providing different data representations. In the following, we will now present adapated localizing methods for our system. After that, we will define how different data representations will intertwine and in the end of this chapter we will present approaches on how the systems can be weighted to improve the accuracy and reliability.

4.1 Step Detection

In our system, we use a system using step detection and matching. This system should give us a location estimation for every detected step like PLATIN (cf. 3.2.1). In the following, we use the usual graph representation given by $G = (V, E)$.

While PLATIN computes the probabilities only for a small neighborhood we want to compute more possible steps and always compare the direction that is detected to the available bearing of an edge. In fact we want to compute every possible step's probability until we reach a certain threshold p_{thresh}. Concerning a starting position node n_i we can calculate:

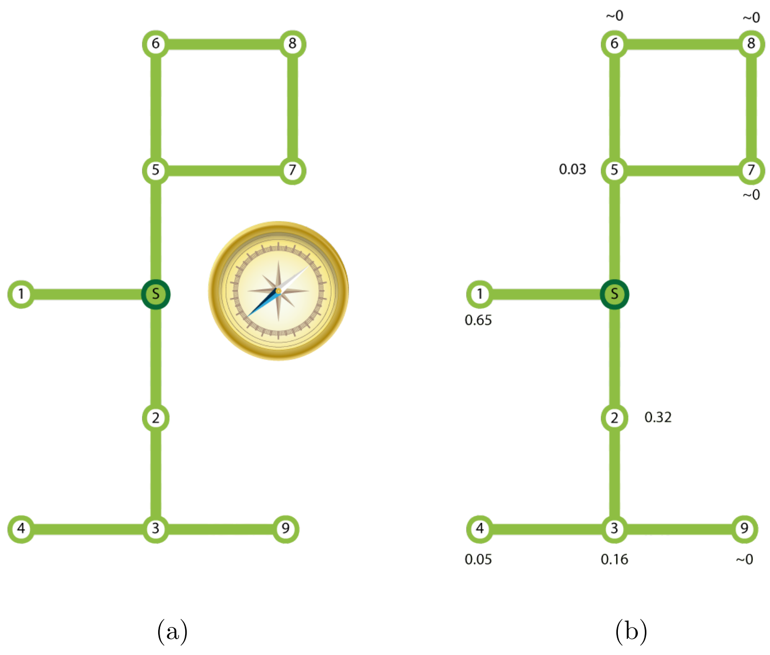


Figure 4.1 The left figure (4.1(a)) shows a map with nodes and edges. The compass denotes the detected step direction where S is the starting position. The computation results are shown in the right figure 4.1(b). The node 4 being three steps away has a higher probability score than the node 5 being a direct neighbor to the starting position.

$$\forall(n_j \in N_1(n_i)) : prob_{ij}(t) = \frac{score(\alpha_{i,j}, \beta)}{\sum\limits_{n_c \in N_1(n_j)} score(\alpha_{i,c}, \beta)}$$

where $\alpha_{i,j}$ is the estimated bearing from n_i to n_j. β is the detected direction of the actual step. $N_1(n_i)$ is the set of neighbors of the node n_i that is reachable by exactly one hop and is defined as:

$$N_1(n_i) = \{n_j \in V \mid (n_i, n_j) \in E\}$$

By that we can compute the probabilities concerning the direct neighbors of the starting node. For the rest of the graph we now use a recursive function:

$$\forall(n_j \notin N_1(n_i)) : prob_{ij}(t) =$$

$$\max\{p_{thresh}, \tfrac{1}{d(i,j)} \cdot prob_{i,m}(t) \cdot \tfrac{score(\alpha_{m,j},\beta)}{\sum\limits_{n_c \in N_1(n_m)} score(\alpha_{m,c},\beta)}\}$$

where $d(i, j) > 0$ is the (minimal) number of hops from n_i to n_j and m is given by:

$$m := \operatorname*{arg\,max}_{n_m \in N_1(n_j)} \{prob_{im}(t)\}$$

This function allows us to compute the probabilities for several steps always concerning the direction of the detected and possible steps. We use a slightly adapted scoring function which replaces the 45° values by 42°. Figure 4.1 shows an example. While the probabilities concerning the nodes to the right are falling fast to the threshold value the values to the left are still greater than the threshold. Using only this step detection system there is no difference to PLATIN in which steps the system concerns to be best matching. If the system has more information (e.g. Wifi) regarding nodes 4,9,8,7, it may choose node four. This is the main reason for choosing that adaption although needing more computational effort than a pure PLATIN approach.

4.2 Wifi Fingerprinting

As presented in 3.1, one can use Wifi fingerprinting for locating a user in an indoor environment. For our system we will use three different approaches for easy and fast location estimation. The first one is the **Nearest Neighbor in Signal Space (NNSS)** which simply computes the nearest neighbor using the euclidean distance formula for the complete signal space. A variaton of the NNSS is the **direct k Nearest Neighbors in Signal Space (direct kNNSS)** which determines not only one nearest neighbor but k neighbors. By having several neighbors we can compute distances between those neighbors and evaluate the quality of the acuracy. We will come back to this later on in chapter 4.5.

The last method we will use is the **indirect k Nearest Neighbors in Signal Space (kNNSS)** which is the simple approach for kNNSS as presented in chapter 3.1. NNSS and direct kNNSS provide discrete information based on the reference fingerprints. They try to map the users position to the reference points without taking the positions in between into account. The indirect kNNSS by contrast provides an estimate on the real position and enables a more accurate estimation in theory, formally:

$$p = (p_{lat}, p_{lon}) = \left(\frac{\sum_{r \in D_k} r_{lat}}{k}, \frac{\sum_{r \in D_k} r_{lon}}{k} \right)$$

where D_k is the set of the GPS positions of the k nearest neighbors in signal space. Later on this p has to be mapped to the best fitting map node like other GPS-based estimates.

4.3 Floor Color

Another system we will define is to exploit information about the floor colors. Many buildings do not have the same floor everywhere. For example the computer science building of the RWTH Aachen University has several different grey, white and black floors, and some green, red and blue floors. This information can help to exclude or include whole areas of the map. A simple approach is to get the mean color[*] of an image. We calculate the mean color for every component of an image or exploit special properties of an image representation to get the mean color. In JPEG for example there are blocks of pixels storing their average color first (Direct Current - DC) and the alternating from this value is stored after (Alternate Current - AC). Accessing all pixel values per color component we can naively compute:

$$M_l = \frac{\sum_{i=1}^{k} \sum_{j=1}^{m} c_{ij}(l)}{m \cdot k}$$

where M_l is the mean color of the component l. $c_{ij}(l)$ denotes the color value of the l-th component at position (i, j) of the picture with a dimension of $m \cdot k$ pixel. We store all M_l in an n-dimensional tuple, where n depends on the color scheme.

The system can now take photos or a video and compare the mean color to the reference colors provided. We are using the euclidean distance formula

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

where $x = (x_1, x_2, \dots)$ and $y = (y_1, y_2, \dots)$ are n-dimensional tuples of mean colors as defined above. Now it can choose all refernce points with a the minimal euclidean distance or give a penalty on the others.

4.4 Common Representations

Location Estimation Vectors

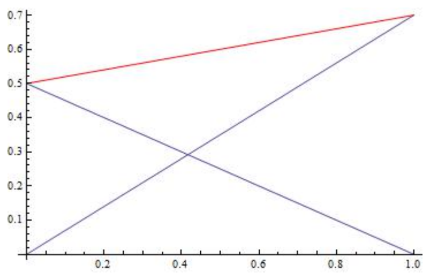
A common representation used is a simple location estimation vector. This vector gives probabilities of being positioned at a point of the map represented by a node in the map graph.

$$v = (v_1, v_2, \dots), v_i \in [0, 1]$$

Therefore this vector will have less elements than the transition matrix containing every possible location inclduing virtual steps as mentioned in chapter 3.2 has. A simple way to determine the missing values is to define a linear approximation based on the virtual steps of the map on the corresponding edge. For nodes i and j the corresponding edge's values can be approximated by

$$prob(i, j, f, n) = \frac{f}{n} \cdot v_j + \left(1 - \frac{f}{n}\right) \cdot v_i$$

[*]In fact there are several color models that can be used like RGB, YUV or CMYK. In this section we only want to give a generic overview of possible computations.


(a) A linear approximation of the values between two nodes. The red line denotes the values between the nodes with values of .5 and .7

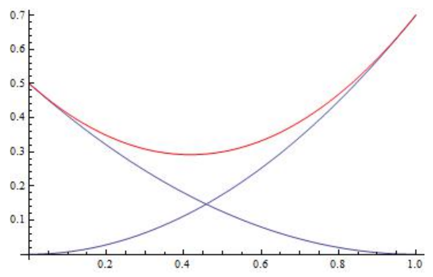

(b) A squared approximation of the values between two nodes. The red curve denotes the values between the nodes with values of .5 and .7

Figure 4.2 Simple plots of linear (left) and squared (right) approximation functions for in between nodes.

where n defines the number of virtual steps between the nodes i and j, f is the number of hops needed from node i to the considered virtual step. This formula leads to a linear in- or decrease of the value for the virtual steps between two nodes. This can be seen in figure 4.2(a). Another possibility is to do it with squared coefficients:

$$prob(i,j,f,n) = \left(\frac{f}{n}\right)^2 \cdot v_j + \left(1 - \frac{f}{n}\right)^2 \cdot v_i$$

This will lead to decreasing values in between the nodes. Especially if there is a long edge or at least many virtual steps along the edge, the values decrease strongly when the considered position is far away from the original provided estimate. This can be seen in Figure 4.2(b).

In both procedures the highest probability values will belong to nodes and not to edges by design. This can be compensated by a decrease of these nodes' corresponding values. This might be done depending on the number and value of the neighbors.

Systems like the QR-code tagging approach will provide exact information where the user is located at the moment. Unlike absolute positions like GPS this tagging approach refers directly to the underlying map structure. Locating a user at a position represented by node n_j leadst to the location estimation vector

$$v = (v_1, v_2, \ldots, v_j, \ldots, v_n) = (0, 0, \ldots, 1, \ldots, 0)$$

Absolute Positions

Systems like GPS do provide absolute positions independent from the underlying map. To map these coordinates to our map we have GPS coordinates of all our nodes and information of their level in case of buildings with multiple levels. We can use these information to get distances to those absolute positions. Iteratetively we can now check the distance of the GPS estimation to all known nodes and virtual steps. The system can do this using the haversine formula, the spherical law of

$$D_1 = \begin{pmatrix} 0.0 & \infty & \infty & \dots \\ \infty & 1.0 & & \\ \infty & 3.5 & & \\ \infty & 5.5 & & \end{pmatrix} \rightsquigarrow D_2 = \begin{pmatrix} 0.0 & \infty & \infty & \dots \\ \infty & 1.0 & 4.5 & \\ \infty & 3.5 & 1.0 & \\ \infty & 5.5 & 4.5 & \end{pmatrix} \rightsquigarrow D_3 \rightsquigarrow \cdots \rightsquigarrow D_m$$

Figure 4.3 An example for penalty matrices storing new information in new columns. Lower values mean higher probabilities for transition.

cosines[33] or a simple euclidean distance. We will use the spherical law of cosines because it provides a good accuracy at reduced computational effort compared to the haversine formula.

Penalty Matrices

Systems like FootPath will present their information by using a penalty transition matrix. The entries of this matrix do have values of 0 to infinity. By every detected step this matrix will grow by one column. The range of values of the entries are varying with each new column.

D_m is the matrix implied by m detected steps. In the top left corner there is a 0.0 representing the starting point while all other entries of this row and column are set to infinity.

First, we define our own transition matrix D'.

$$D'_m = (d'_{ij}), 1 \leq i \leq n, 1 \leq j \leq m$$

where n is the dimension of the columns implied by the available positions. We will now transform this matrix from a penalty matrix to a transition matrix as defined before. Therefore we calculate the matrix D'_m instead of D_m. First we transform the first row and column.

$$d'_{11} = 1.0, d'_{1k} = 0.0, d'_{k1} = 0.0, \forall k \neq 1$$

Now we calculate the other entries column-wise.

$$d'_{ij} = 1 - \frac{d_{ij}}{\sum\limits_{1 \leq k \leq n} d_{kj}}, d_{ij} \in [0, \infty),$$

Now we have a probability matrix that is giving us the highest value entry for the most probable transition.

$$D'_1 = \begin{pmatrix} 1.0 & 0.0 & 0.0 & \dots \\ 0.0 & 0.9 & & \\ 0.0 & 0.65 & & \\ 0.0 & 0.45 & & \end{pmatrix} \rightsquigarrow D'_2 = \begin{pmatrix} 1.0 & 0.0 & 0.0 & \dots \\ 0.0 & 0.9 & 0.55 & \\ 0.0 & 0.65 & 0.9 & \\ 0.0 & 0.45 & 0.55 & \end{pmatrix} \rightsquigarrow D'_3 \rightsquigarrow \dots \rightsquigarrow D'_m$$

Figure 4.4 The example of figure 4.3 after transforming from a penalty to a probability matrix. Now the highest value define the most probable transition.

In this example we transformed a complete penalty matrix by transforming every column vector of the matrix before inserting it. By that we have shown that the vectors being inserted into a penalty matrix can be easily converted to a representation we already know from a definition above – the location estimation vector. Because we assume the markov property for the system, the penalty-based system can easily get the last state of the system and eventually invert the transformation for own computations.

4.5 Weighting Function

Different ways of localizing indoors will have different qualities in terms of accuracy, actuality, reliability and more. To compensate bad or inaccurate information and priorize better information we use a weighting function. This function has initial values depending on the information known about the modules in advance.

Furthermore this weighting function can be time-dependent. Sensor malfunctions or temporal lack of information of the modules can be compensated dynamically.

We add the weighted localization estimation vector to the new column vector of the base matrix, formally

$$d_k^{\Psi} = d_k + \psi_{WiFi}(t) \cdot e_{WiFi} + \psi_{GPS}(t) \cdot e_{GPS} + \dots$$

where d_k is the current column vector of the base matrix. $\psi_i(t)$ denotes the weighting function at time t for module i whose localization estimation vector is given by e_i.

For this weighting there is a need of evaluation criteria and metrics to determine the actual quality of the provided data of a module.

Intuitive metrics could be:

Sensors used: Different methods of localizing use different sets of the sensors provided by some smartphones. The more sensors the system has to rely on the higher the probability a random sensor malfunction will hit one of the needed ones. This can be quantified before the start and in most cases won't change over time. Therefore we can use this metric for static and dynamic weighting.

Expected Accuracy: Different methods and different environments influence the expected accuracy of the localization. We may gauge this before starting the whole system but especially different environments will need a dynamic weighting. Therefore we can use this metric for static weighting only in a limited way.

Delay/Frequency: The base matrix will fill at the rate of the step frequency of the user. Therefore when standing or moving slowly high-delay methods can provide better estimates of the position than when the user is walking fast. This metric may be estimated before starting but may change over time. Therefore we can use this metric for static weighting in a limited way.

Inaccuracy History: In case of false calibrations for example the localization method will continuously provide bad estimates and is not reliable any more. Only dynamic weighting can investigate here.

Computational Effort: The computations needed for estimating the actual position in a module should meet the available ressources and should not be finished a long time after the corresponding sensor data. This metric can be assessed in advance and refined during the dynamic weighting process.

Energy Consumption: Mobile devices have limited energy ressources. We can use this metric to determine if a method should be used or set sleeping to protect the battery.

4.5.1 Static Weighting

The static weighting function is simple and fast. Before the start of the system the weighting factors are set. Independent from the environment and the actual quality those values are fixed. We can multiply provided location estimation vectors by the fixed factors and then sum them up. The only exception is a synchronizing provider meaning a 100% probability of being at a certain point. users can do this for example manually or by special modules like the QR-code recognition.

4.5.2 Dynamic Weighting

The dynamic weighting is an extensible function that can react on several environmental or technical changes. We defined some of those rules as follows.

We can reduce the weight of *Step Detection* if the detected step direction has only low scores concerning the possible walking direction provided by the map. For example users continue walking at a dead end. Formally:

$$penalty(t) = \begin{cases} 0.0, & \text{if } max(scores(t)) = 20.0 \\ f_{penalty}, & \text{if } max(scores(t)) = 10.0 \\ 2 \cdot f_{penalty}, & \text{if } max(scores(t)) = 5.0 \\ 3 \cdot f_{penalty}, & \text{otherwise} \end{cases}$$

where $f_{penalty}$ is the predefined value for penalties and $scores(t)$ is the set of the scores computed for the actual step concerning the neighboring nodes. By reducing the weight the system is now more sensitive to other systems that can reset the position to the correct location.

The weight of the *Wifi fingerprinting* depends on several factors. Most Wifi networks are set up indoor and by that usually the access points are inside the building. When being outdoors the propagation source is provided from one direction only in most cases – the direction of the building. By that outdoor localization using Wifi may not result in good qualitiy and therefore we introduced an outdoor penalty reducing the weight. Furthermore the quality strongly depends on the signal space a scan does recognize. We introduced a threshold which denotes the minimal signal space dimension we want to have. If a scan can detect only one or two known Access Points the resulting estimate is strongly penalized. When concerning (in)direct kNNSS we have a set of reference points the system uses for estimating the location. In case of direct kNNSS we look at the mean distance of the k neighbors to each other and in case of indirect kNNSS we use the mean distance of the approximated location to the reference points. If these distances are greater than a defined threshold we will ignore the estimate. The accuracy computation for direct k NNSS is given by:

$$\frac{\sum\limits_{i=1}^{k-1}\sum\limits_{j=i+1}^{k} l(d(i), d(j))}{\frac{k\cdot(k-1)}{2}}$$

where $d(i)$ is the i-th nearest neighbor in signal space and $l(x, y)$ the distance between nodes x and y in cm. The counterpart for indirect kNNSS is given by:

$$\frac{\sum\limits_{i=1}^{k} l(\tau, d(i))}{k}$$

where additionally τ is the node the indirect kNNSS matches for the best mapping position.

With regard to *GPS* we first check for being in an indoor environment. In that case we will penalize the GPS estimates. Furthermore we do check the GPS history. If the last estimations have been all the same but we know there have been detected steps we penalize the GPS estimation.

Floor Color can give information about whole regions. If the whole building except two rooms have the same floor color the color will be not interesting as long as the user is not located in one of the rooms or the detected color changes. Therefore usually the weight depends on the fration of the actual detected color to the total of known colors. By that most of the time the weight is low but rises strongly when detecting rare colors. Formally we simply define:

$$w_{ColorFraction} := \frac{\sum_{c_d \neq c \in C_r} 1}{|C_r|}$$

where c_d is the detected (respectively mapped) color and C_r the multiset of all reference colors. By that rare colors automatically get a high priority.

All systems do rely on sensors to get information about the user and the environment. The weighting function allows the system and modules to report damaged sensors or temporarily bad qualities. All locating modules using damaged sensors can be penalized or temporarily disabled until the sensors may work again.

4.6 Transition and Position

In chapter 2.7, we introduced the markov chains and transition matrix. After converting and combining the representations of location estimations by the modules we now have a filled transition matrix. Every column will have several values we can use for determining the transition. A direct transition column by column corresponds to the First Fit Algorithm that was shown in FootPath (cf. 3.2) and we will use it. We will not discuss possible BestFit, MultiFit or Backtrace variations in this thesis.

If needed we can furthermore normalize the matrix columns to get a left stochastic matrix[34] by computing

$$\bar{d}^{\Psi}_{ij} = \frac{d_{ij}}{\sum\limits_{1 \leq k \leq n} d_{kj}}$$

which leads to a matrix with values of [0,1] and can be used by systems that require left stochastic matrices.

4.7 Conclusion

In this chapter we have shown a system for a stochastic based step detection and mapping approach that is giving rough information beyond the direct neighborhood of the last estimated position. These rough information in combination with more estimations of other systems can tip the balance at some time. Furthermore we have shown how to cope with different representations for the location estimations and converting them into the same domain and representation. This representation – the location estimation vector – can now be used for combining these different estimations using the weighting function. We can use either a simple and fast static weighting function or a more flexible dynamic and extensible weighting function for which we have shown some metrics and rules.

Because the localization methods work autonomously, users and systems can realize simple filters. For example we can realize a filter for only regarding some floors of a building or fuzzy queried areas by providing a location estimation vector with a sufficient weighting factor. Even a progress tracking for a path that is known in advance like in FootPath can be realized by adding a filter only providing values for the steps lying on the path.

5 Implementation

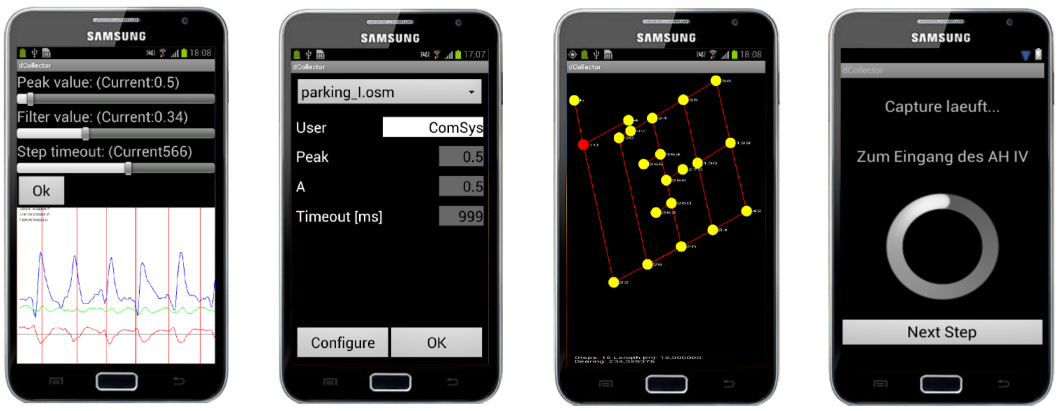
In this chapter, we will present the applications we implement and use for this thesis. We will enhance the dCollector application that enables us to capture all sensor information during a walk with a graphical user interface for Android for more complex pre-defined routes users can walk. We presented basics of Wifi fingerprinting in chapter 3.1 and now need reference points in the signal space for the area we will concern in the evaluation chapter. To do this we implement a simple Wifi wardriving application for Android, WiFiWar, that captures several scans and stores data tuples containing the pseudo-unique Basic Service Set Identifier (BSSID), its frequency, capabilities and signal strength measured as RSSI. We can use these data tuples as reference points later on.

The last application we implement is the EXtensible Indoor Localization Emulator (EXILE). This emulator enables us to use collected data (e.g., by using dCollector), a map (e.g., from OpenStreetMap) and some environmental information (e.g., Wifi fingerprints collected by WiFiWar) to emulate the behaviour of the implemented modules during the walk. By this we can vary the configuration parameters, the modules to use and the weighting function defined in chapter 5.2.3.

5.1 WiFiWar and dCollector

dCollector

Felix Gerdsmeier[24] initially developed the dCollector application during his thesis. Figure 5.1 shows the activities and screens. The dCollector has mainly been used for collecting parking lot walks whose graphical representation can be seen in 5.1(c). In our later setup, we will have a bigger map with several levels, more nodes and edges and a partially high density of nodes. Furthermore, we will use pre-defined routes for testing as we describe in the upcoming chapter. Because of this, we decided to enhance dCollector with an alternative activity for recording the data.



(a) Setup for step detection (b) Settings for user and map (c) Capturing mode (d) Text-based recording activity

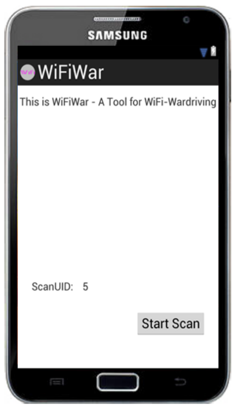
Figure 5.1 In 5.1(a) there is an user interface for the parameters of step detection like the step timeout as described in 3.2. 5.1(b) shows the starting screen which enables the user to select a map and a name for the walk. The parameters selected in 5.1(a) are automatically inserted if changed. The capturing and recording mode 5.1(c) shows the actual map as graphical representation. The user can now see possible directions (if provided by the map data) and touch nodes to notify about her actual position.[24] 5.1(d) shows the new alternative text based recording activity. The text (here written in german due to some testpersons speaking german only) provides information where the user should go. At the bottom a button for notifying the system of reaching the target is given. While capturing data the progressbar is running to give the user some viusual feedback.

Figure 5.1(d) shows the new activity that relinquishes the complete graphical representation of the map for the user. Instead we use a mainly text-based frontend. The provider can now define a route on the map and define a text, the application will show after a user has reached a certain point. By that we can supply continuously text-based instructions to the user. In the graphical representation, the user can touch a node to notify the application of being at a certain point. The text-based activity provides a button that the user can touch or click when having reached a target given to the user by the text.

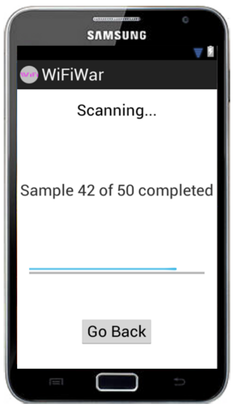
Especially when we want a user to walk a certain path this approach can reduce the load an user has to keep in mind. All information needed for the walk is provided in the world – the User Interface – and not needed to store it all in the mind[35]. That means the user can get all information she needs by reading instead of keeping it all in mind. In contrast to the graphical representation the application can scale this activity to nearly every map and path. The knowledge the user needs to keep in mind remains constant. If we want the user to walk an arbitrary path, we can not use this interface but the oiriginal one.

WiFiWar

We can use this application for taking fingerprints of the Wifi signal space at certain map positions. The system needs to provide:



(a) Main Activity



(b) Scanning Activity

Figure 5.2 5.2(a) shows the main activity. The user can now start a scan and see the actual identifier. 5.2(b) is the capturing activity providing feedback to the user via the progressbar while storing data and computing the mean values of signal strengths.

Sufficient Number Of Scans: Environmental influences and other stations in a network do influence the signal strength and absorption ratio. Therefore one scan alone is neither reliable nor sufficient. Experiments[19] showed that at least 30 scans per position provide a good quality for fingerprinting. Initially we will use 50 scans per position for WiFiWar.

Identification: To recognize a fingerprint later on we will have an automatically increasing identifier for the measurements. In combination with timestamps this will provide a good recognition after measuring. The application can store the data as text files containing the identifier and timestamp as well as further information.

Feedback: Good usability can not be realized without providing feedback[36]. When scanning we will show a progressbar that will increase for every scan which the system receives and logs. Furthermore the main activity interace will show the actual identifier.

Figure 5.2 shows the two activities of WiFiWar. The main activity is simply providing the information about the actual identifier and enabling the user to trigger a series of scans. If the user does this, the scanning activity is providing feedback on the actual progress concerning the scan series. Simultanuously the application will log the scan results and the mean signal strengths of all detected networks, identified by their pseudo-unique BSSIDs to Files.

Used Framework

For developing WiFiWar and entending the dCollector Android application we used the Android SDK [37] and Android Developer Tools for Eclipse. Both applications require the minimal API version 8. By that users can install these applications on Android devices with at least Android 2.2 which was released years ago in May of 2010[38].

5.2 EXILE

The EXtinsible Indoor Localization Emulator (EXILE) allows the user to input collected data sets, known environmental information and the map of the concerned area. It emulates the behavior of the implemented modules using the provided data sets.

5.2.1 Basic Model

EXILE is subdivided into a set of packages. The **import** package only consists of two classes providing the functionality to import OpenStreetMap files. Those files follow a strict XML-syntax allowing the importer to use XML parsing methods of the Java standard packages. To represent the imported map data we need a graph representation.

We provide our graph representation in the **graphenbib** package. This package contains classes describing nodes (*MapNode*) and edges (*MapEdge*) as well as GPS coordinates (*GPSCoordinate*). The *MapNode* class manages the edges it is connected with and can be uniquely identified by an ID we can extract for example from the OSM data. Beside representing an edge in the graph the *MapEdge* provides methods to create virtual steps as mentioned in chapter 3.2.1. Furthermore it stores the bearing of the node as defined in [33]. The *GPSCoordinate* class can store the latitude and longitude of a position and compute the distance to another node using the spherical law of cosines [33]. We attach those coordinates to the MapNodes while importing and to the virtual steps on the edges by simple linear approximation. The *MapGraph* class manages the complete graph storing the nodes in a HashSet using the unique ids as keys.

The **mods** package stores all implemented modules for localization and provides interfaces and an abstract class for typecasting the modules. We present this in detail in chapter 5.2.2.

Figure 5.3 shows an UML class diagram of the main package and its interconnections with other packages. The *Matrix* is a simple matrix data structure implementation. During the emulation the system often reads and stores complete columns of the matrix. Because of that the matrix stores its values columnwise and adapts its dimensions dynamically. To prevent modules from editing the matrix values, the methods for editing or inserting data into the matrix is only visible inside the package. To match the matrix column indices to a position on the map we define a *BiMap<E,T>* which is a bidirectional map implementation. By that we can get the index representing a position as well as the position an index is describing. The *Logger* singleton provides logging functionalitites. The system can configure the Logger to directs its output into the command line, a File or a MessageBox. Furthermore the Logger can accompany every output by the corresponding timestamp.

The *Emulator* class provides the usual `static void main` method that is starting the complete emulation. First the class imports all basic data sets and creates the graph object by importing the OSM file. After this it registers and initiates all known modules that we want to use. The *Config* class stores the definition what modules to use for emulation. It furthermore contains several parameters for modules and

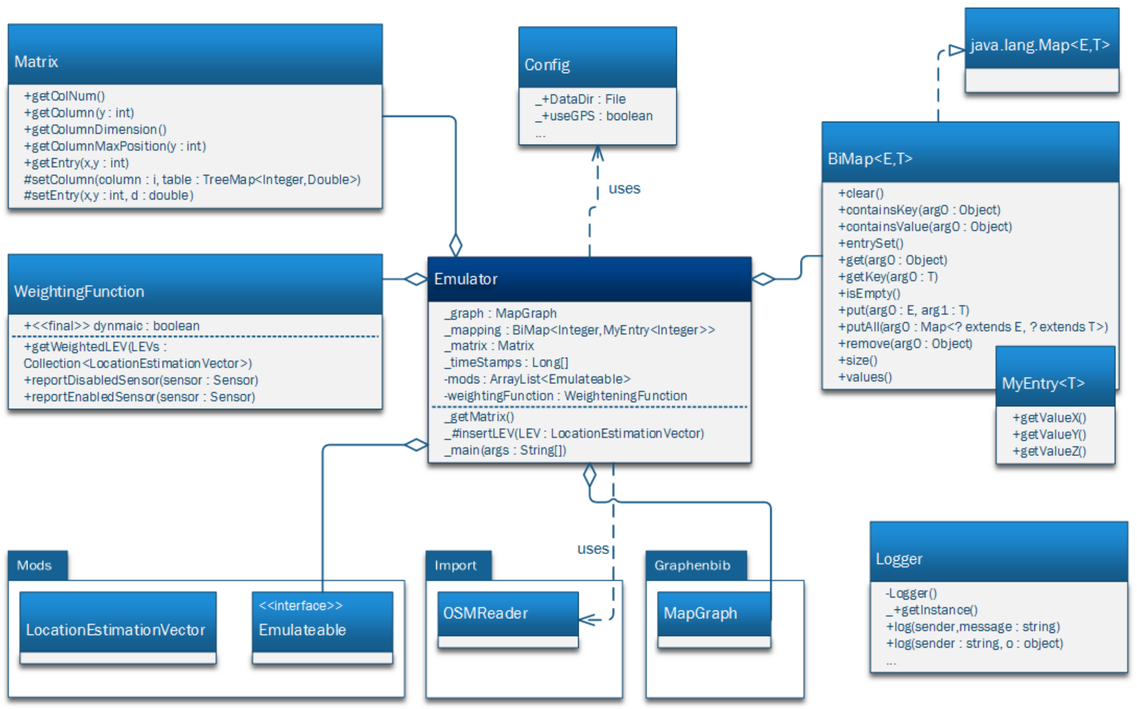

Matrix
+getColNum()
+getColumn(y : int)
+getColumnDimension()
+getColumnMaxPosition(y : int)
+getEntry(x,y : int)
#setColumn(column : i, table : TreeMap<Integer,Double>)
#setEntry(x,y : int, d : double)
Config
_+DataDir : File
_+useGPS : boolean
...
uses
java.lang.Map<E,T>
BiMap<E,T>
+clear()
+containsKey(arg0 : Object)
+containsValue(arg0 : Object)
+entrySet()
+get(arg0 : Object)
+getKey(arg0 : T)
+isEmpty()
+put(arg0 : E, arg1 : T)
+putAll(arg0 : Map<? extends E, ? extends T>)
+remove(arg0 : Object)
+size()
+values()
MyEntry<T>
+getValueX()
+getValueY()
+getValueZ()
WeightingFunction
+<<final>> dynmaic : boolean
+getWeightedLEV(LEVs : Collection<LocationEstimationVector>)
+reportDisabledSensor(sensor : Sensor)
+reportEnabledSensor(sensor : Sensor)
Emulator
_graph : MapGraph
_mapping : BiMap<Integer,MyEntry<Integer>>
_matrix : Matrix
_timeStamps : Long[]
-mods : ArrayList<Emulateable>
-weightingFunction : WeighteningFunction
_getMatrix()
_#insertLEV(LEV : LocationEstimationVector)
_main(args : String[])
uses
Mods
LocationEstimationVector
<<interface>>
Emulateable
Import
OSMReader
Graphenbib
MapGraph
Logger
-Logger()
_+getInstance()
+log(sender,message : string)
+log(sender : string, o : object)
...

Figure 5.3 UML class diagram of the main package of EXILE. The *Emulator* class provides the `static void main` method and starts the emulation. What data sets are used for emulation can be configured in the *Config* class.

output behavior. After initiating all modules the emulator iteratively tries to get estimates for the timestamps that the step detection provides. The system gives the estimates to the *WeightingFunction* which we will concern in detail in chapter 5.2.3. The emulator stores the resulting estimate in the matrix object. The modules can have read-only access to the matrix if they need the last position for computing the next estimate. The emulator provides some methods for outputting the results. It can simply print the whole detected and estimated route including the area the user has been in reality or just print the estimated positions at certain points of time, at which the user informed the system about her actual position. The emulator then calculates the distance of the estimated position to the real position. The unit for this is how many steps a person has to go to reach the other position. It can also print the whole matrix or store it in a file.*

The last package is the **tests** package containing some Unit Tests for the import and graphenbib package. Furthermore there are configurable quick tests for the user to check the loadability of data sets.

5.2.2 Modules

Figure 5.4 shows an UML class diagram of the mods package. *StepDetection, WiFiFinger, FloorColor, GPS, QR* are concrete implementations of localization methods. They all extend the abstract *AbstractMod* class defining a constructor

*The matrices number of entries is equal to $(\#_{detectedSteps}+1)\cdot(\#_{Nodes}+\#_{virtualSteps})$. Printing huge matrices to command line or files will take some time.

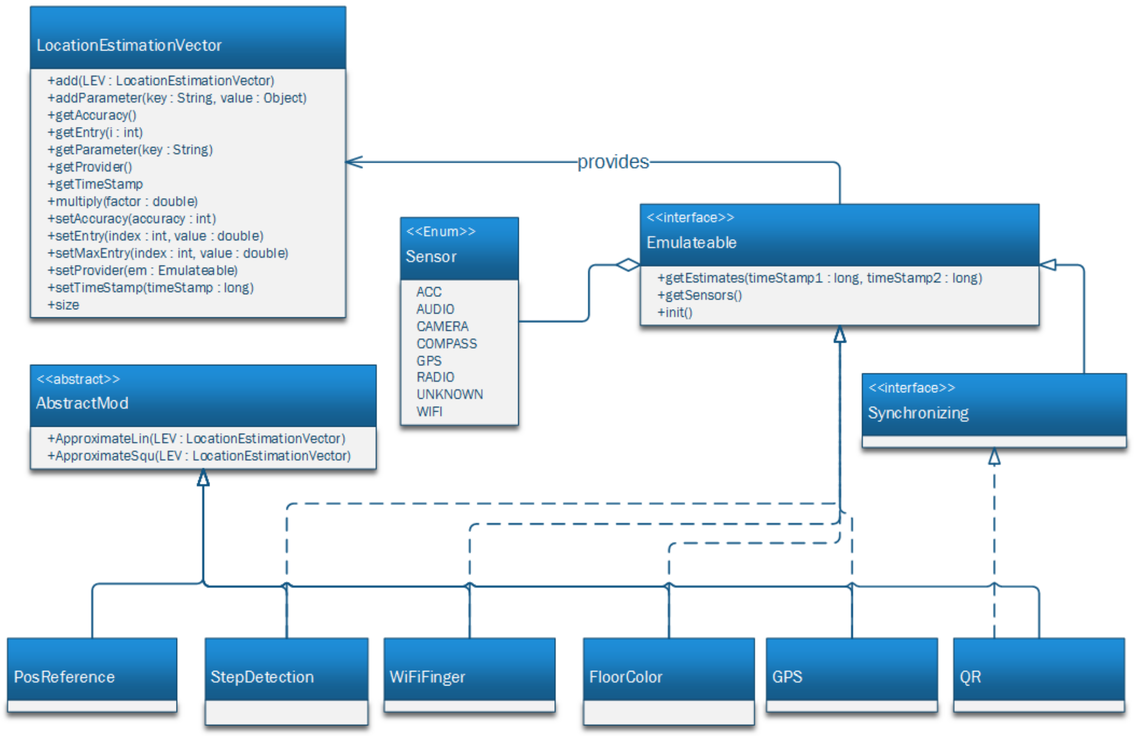

LocationEstimationVector
+add(LEV : LocationEstimationVector)
+addParameter(key : String, value : Object)
+getAccuracy()
+getEntry(i : int)
+getParameter(key : String)
+getProvider()
+getTimeStamp
+multiply(factor : double)
+setAccuracy(accuracy : int)
+setEntry(index : int, value : double)
+setMaxEntry(index : int, value : double)
+setProvider(em : Emulateable)
+setTimeStamp(timeStamp : long)
+size
provides
<<Enum>>
Sensor
ACC
AUDIO
CAMERA
COMPASS
GPS
RADIO
UNKNOWN
WIFI
<<interface>>
Emulateable
+getEstimates(timeStamp1 : long, timeStamp2 : long)
+getSensors()
+init()
<<abstract>>
AbstractMod
+ApproximateLin(LEV : LocationEstimationVector)
+ApproximateSqu(LEV : LocationEstimationVector)
<<interface>>
Synchronizing
PosReference
StepDetection
WiFiFinger
FloorColor
GPS
QR

Figure 5.4 An UML class diagram of the mods package. Indendent from detailed implementations the interfaces can be used for getting estimates. The *LocationEstimationVector* class is the implementation of the common representation introduced and defined in chapter 4.4.

awaiting the graph representing the actual map we concern. Additionally the AbstractMod provides methods for linear and squared approximation as introduced in chapter 4.4. The *Sensor* enum represents sensors that systems use for calculating estimates. We can use this enum in the weighting function later on.

If the system should emulate an implementation, the module needs to provide some methods defined by the interface *Emulateable*. The `init()` method should initiate the module and check for available and (formally) correct input data. If there is no data or an error occurs during importing the data, the method should return false. Recognizing this the emulator can remove this module from the set of emulateable modules for this execution. `getEstimates(timeStamp1, timeStamp2)` is the call for getting localization estimates. The two timestamps restrict the data the modules should use. The first timestamp denotes the actual detected steps timestamp and the second timestamp concerns the next detected step. By that the modules can decide what data to use if there are multiple data sets in between the two timestamps. This method needs to return a LocationEstimationVector object.

The *LocationEstimationVector* class is an implementation of the location estimation vector we present in chapter 4.4. It allows to store the values for the actual estimation as well as meta information. It stores the provider of the estimate, the concerned timestamp and allows modules to put self-defined parameters. The weighting function can use these parameters later on. In addition to that this class implements basic arithmetic operations like adding vectors or multiplying the vector with a scalar value.

Some modules do not give simple estimates on the actual position but exact information about the user's position at this point in time. This can happen for example when the user manually resets the position in the system or scans a tagging code, a module knows its exact position. (We implement this by a pseudo QR code module.) These modules do not simply implement the Emulateable interface but the *Synchronizing* interface which is an extension of the Emulateable one.

5.2.3 Weighting Function

The weighting function introduced in 4.5 has two operation modes. The first mode is the static weighting mode (the *dynamic* flag is set to false) which iterates over the complete set of given location estimation vectors and multiplies each one with the respective factor defined before executing the emulator. After this it sums up all vectors and returns the resulting vector to the emulator which can store it in the matrix. The only exception is the detection of a location estimation vector provided by a synchronizing module. In that case the function drops all estimates and returns the sychronizing vector directly.

The dynamic weighting in contrast will do various calculations before adding the estimates. We gave the mathematical and formal definitions of the weighting function in chapter 4.5. In cases like Wifi fingerprinting the function reduces the factor by checking properties of the map like being indoors. The weighting function can then consult the matrix the Emulator class stores for the last position and the graph for details about this position. Other rules define an accuracy threshold. The weighting function can compare those values and then adjust the factor. Furthermore, the implementation allows the modules to store self-defined data objects in the LocationEstimationVector object as parameter. By that the modules can provide the weighting function additional information that the function uses for weighting. In chapter 4.5 we defined a dynamic weighting for the floor color detection regarding the fraction of the detected color and all reference colors. The weighting function can use this by extracting the value from the parameters:

```
weightFloorColor(LocationEstimationVector tempVect) {
        /********************************
         * 1st Step: Check for color distribution
         *******************************/
        double colorFract = 0;
        Double d = (Double) tempVect.getParameter("ColorFraction");
        if (d != null) {
            colorFract = 1-d;
        }
        //...
        factor = factor * colorFract;
        tempVect.multiply(factor);
    }
```

The weighting function can check if the estimates of the modules do change over time. If the system knows one (or more) steps have been made but a module is still giving the same estimates, the function can reduce the estimate's weight. We do this concerning GPS for example:

```
weightGPS(LocationEstimationVector tempVect) {
        //...
        /********************************
         * 2nd Step: Check GPS-History
         * (Several times the same LEV is not a good quality)
         *******************************/
        boolean unchanged = false;
        //Only check if there is a history...
        if (this.lastGPS != null) {
            if (lastGPS.equals(tempVect)) {
                unchanged = true;
            }
        }
        //...
        if (unchanged) factor = (factor/4);
    }
```

5.2.4 Adding New Modules

EXILE is extensible with more modules. To show the extensibility of our implementation we will now show a little and quick example how to implement and register a new module to EXILE with little effort.

We can use the modularity to realize special filters on the emulation like filters for building floors or fuzzy queried ones. We will now design a filter module called *NoPrimes* which will not allow transitions to positions, nodes having a prime number[†] are representing.

First we have to create the class NoPrime implementing the Emulateable interface and extending the abstract class AbstractMod. Now we will implement the filter in the getEstimates method:

```
getEstimates(long timeStamp1, long timeStamp2) {
    //create new LocationEstimationVector
    LocationEstimationVector LEV =
        new LocationEstimationVector(Emulator.mapping.size());
    //create List/Array
    ArrayList<Integer> filtered = new ArrayList<Integer>();
    //search for every prime-IDs
    Iterator<MapNode> it = this.graph.getNodeIt();
    while (it.hasNext()) {
        Integer uid = it.next.uid;
        if(isPrime(uid)) filtered.add(uid);
    }
    //Now store these information in the LEV
    for (Integer id : filtered) {
        //getting the index representing node id
        int index = Emulator.mapping.getKey(
            new MyEntry<Integer>(id,0,0));
        //Let's use penalties for this
        LEV.setEntry(pos,-1);
    }
    //return the LEV
    return LEV;
```

[†]Assuming identifiers are represented as numbers as usual in OpenStreetMap

```
}
```

To complete this class we only need to implement two short methods defined by the interface:

```
init() {
    //This module will work, just return true
    return true;
}

getSensors() {
    //Just return an unknown sensor
    Sensor[] sen = new Sensor[1];
    sen[0] = Sensor.UNKNOWN;
    return sen;
}
```

Now the module is finished and we have to tell the emulator to use this module. We do this in the main package by simply adding

```
//Creating and registering NoPrime
NoPrime np = new NoPrime(graph);
if (Config.useNoPrime) {
    premods.add(np);
}
```

in the Emulator class below the other modules' creation and registration and

```
/**
*   Enables/Disables the NoPrime Module
*/
public static final boolean useNoPrime = true;
```

in the Config class. In fact, we could easily realize this without using the Config class but for consistency reasons we include full configuration possibilities into the Config class. At last we only need to extend the weighting function by simply adding:

```
getBaseWeighting(Emulateable mod) {
    //...
    if (mod instanceof NoPrime)
        return 1.0;
}

getBaseDynWeighting(Emulateable mod) {
    //...
    if (mod instanceof NoPrime)
        return 1.0;
}
```

Now the NoPrime module that is realizing a filter is finished and working. The user can enable or disable it by simply toggling the boolean value in the configuration. For now the module gives a penalty on every position that meets the filter definition. If there is a set of multiple other modules we can increase the penalty factor dynamically concering the number of modules providing estimates. We can implement this by some lines in the weightingFunction class by simply adding 0.2 to the factor for the NoPrime module for every module that is providing estimates.

```
weightDynamically(LocationEstimationVector LEV,
                    Collection<LocationEstimationVector> LEVs) {
    //...
    //already got double factor and the actual Module em by default
    if (em instanceof NoPrime) {
        //Let's say we increase the factor by 0.2
        //for every module that gave an estimate
        factor += (0.2*LEVs.size());
    }
}
```

By that we have shown how to implement and add a new module to the system with low effort and lines of code. This enables the user to implement complex filters and context sensitive weighting functions (e.g. automatically increase weight of modules in certain areas).

5.3 Conclusion

In this chapter we have presented the implementation of three different applications. dCollector and WiFiWar are simple Android applications for scanning and recording environmental informations. The main implementation effort has been in the EXtensible Indoor Localization Emulator (EXILE). We have shown that the emulator has implementations of the modules we presented in the design chapter and provides a simple to use configuration and starting routine. Especially we have shown the extensibility allowing every new module to intertwine with the other modules with only small effort. For more information about the implementation one can read the Javadoc provided with this thesis.

The emulator and the dCollector allow us to record walks with test persons and emulate the behavior of the modules later on. This will be presented in the next chapter.

6

Evaluation

In the previous chapters we have shown what problems we want to tackle and how we want to do this. The previous chapter introduced the EXtensible Indoor Localization Emulator (EXILE) we use to emulate walks and the corresponding behavior of the modules we defined and how the weighting function will bring the estimates of the providers together.

We will now show our setup for the walks that we have recorded using the dCollector application and how we define our metric. Then we will present the results and evaluations of the emulations. Especially, we will give details in special behaviors of the system. In the end, we will discuss the results and give a conclusion.

6.1 Setup

We define two routes that the test persons should walk and the dCollector application should record. The first route starts in front of the computer science buidling and ends at another exit about 200 m away. The other route is a short trip on the (western) parking lot of the building.

Computer Science Building

The main building of the computer science building complex will be used for a route the test persons should walk. Figure 6.1 shows a rough representation of the routes. The walk starts in front of the main entrance and then right after having entered the building the walking person has to head left towards the AH IV lecture hall. After about half the way towards the AH IV the user will turn and head back to the entrance. Now the participant is heading towards the janitor's office. After that the user will continue the walk towards the steps connecting the entrance hall and the floor. Then the user walks straight up the floor until reaching room 2015. She has to enter the room and than head back to the floor continue walking down the

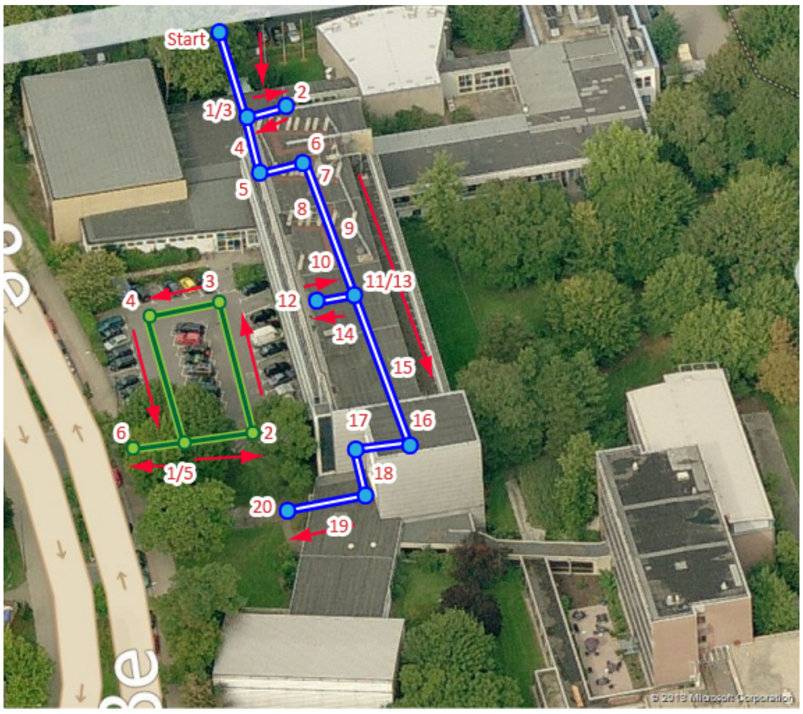


Figure 6.1 A picture of the main building of the computer science building complex of the RWTH. The white/blue path represents the route the test persons have been walking. The green path represents the parking lot walk that has been performed.

floor until reaching the AH I lecture hall. Now the user just walks towards the next exit being only two doors and turns away. The numbers in the picture denote the reference points we use.

By that we have multiple interesting characteristics. We have transitions of indoor and outdoor environments. We expect the Wifi fingerprinting capabilities to increase after entering the building. Walking only a fraction of the edge towards the AH IV lecture hall we expect modules focussing on nodes to get problems. The hallway between the entrance hall and the lecture hall AH I is quite long. By entering a room in between we are interested in what emulations do detect the right room for entering. Furthermore, we are interested in problems when entering a room for a specific localization methods.

We collect several fingerprints inside the entrance hall, inside the hall way and near the exit at different distances. Those fingerprinting positions can be mapped to nodes of the map representation directly. We provide fingerprints lying on the path as well as fingerprints at map positions apart from the path the participants walk.

We recorded walks of the participants Marlin, Svenja, Arno, Martin, Adrian, Eva, Maren and Domme. For details about the walks and those data sets confer the enpacked CD (cf. A.1)

Parking Lot

The parking lot experiment is a quite small walk as figure 6.1 represents. The user will start at the position of the node in the middle of the threeway intersection and then follows the parking lot guidance and in the end heads towards the street.

6.2 Results

During the test runs the users notify the system when having reached a certain point. By this we have tuples of timestamps and the corresponding users positions. We will use this position informations for reference, which we will compare with the estimated positions the modules provide. For every of those references the system can calculate the distance between the reference and estimated position. We provide detailed tables of the emulation results and the distances per reference position on the enclosed CD (cf. A.1). In diagrams we may refernce the name of a person whose walk we are regarding.

Furthermore we consider the estimated route the system is providing and check if the system is stuck in a region and the way it deals with entering a certain room in between other ones in the compuer science building experiment.

Metric: Distance to the real position

We measure the distances of real positions to the estimated ones in terms of steps to go to reach the real position starting from the estimated one. By that we get distances using the unit of steps to go along the edges. The emulations usually use values of step sizes in between 80 and 180 cm. By that the mean step size of the emulations is a value around 130 cm. Our first experiments showed that in between these values we get the best results. Because the best fitting step size has been a diferent one in each walk we now consider this interval for every walk. Because of 'jumps' occuring when the system is reseting the position to a not neighbored position the values of distances may switch between small and high values rapidly. In fact, the step detection and matching will not do this but may increase the distance constantly over time. Regarding this the mean distance to the real position is not a useful metric and we will use the median instead to characterize the quality of an emulation.

6.2.1 Computer Science Building

First we focus on the step detection and matching approach and compare it using different step sizes to the combination with Wifi fingerprinting. Furthermore we consider how the static and dynamic weighting differ in quality. In the end we evaluate some emulations using additionally floor color and QR tagging approaches.

In the following we use the following abbreviations:

SD: Denotes the step detection and matching approach introduced in chapter 4.1

ind./d. kNNSS: The indirect and direct kNNSS Wifi fingerprinting methods defined in chapter 4.2

Lin/Squ: The application of linear/squared approximation before returning the location estimation vector (cf. 4.4).

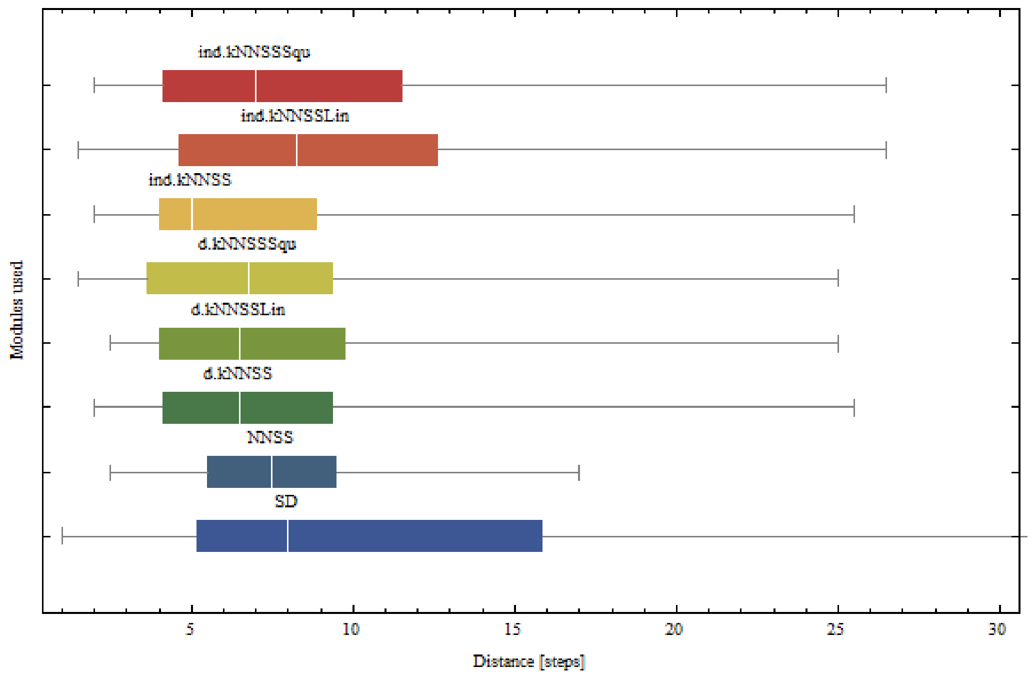


Figure 6.2 Boxplots of all median distances of the estimated and real position using static weighting. In many cases the use of Wifi fingerprinting in addition to the SD approach provides a better excpected quality in estimating the position. The maximum value of the SD has been way more due to getting stuck during some walks. The maximum value of SD is 82.5.

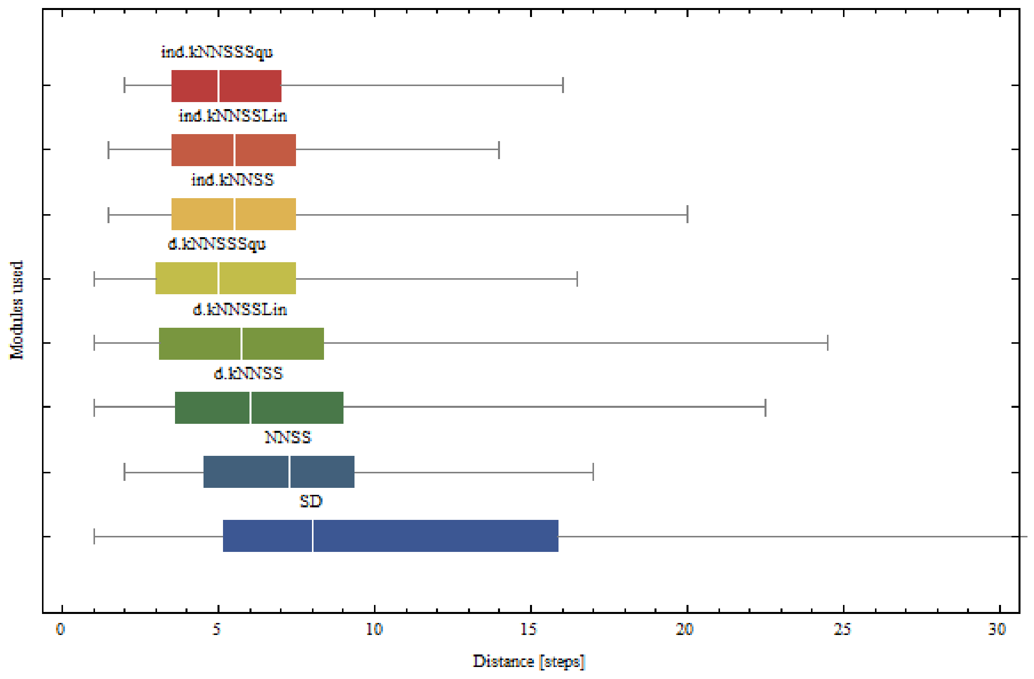
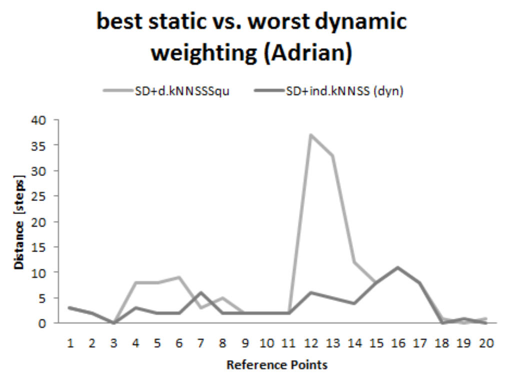


Figure 6.3 The same diagram as in figure 6.2 for dynamic weighting. The values are smaller than in static weighting while having smaller ranges. In most cases the dynamic weighting provides a better quality than the SD approach. The maximum value of SD is 82.5.

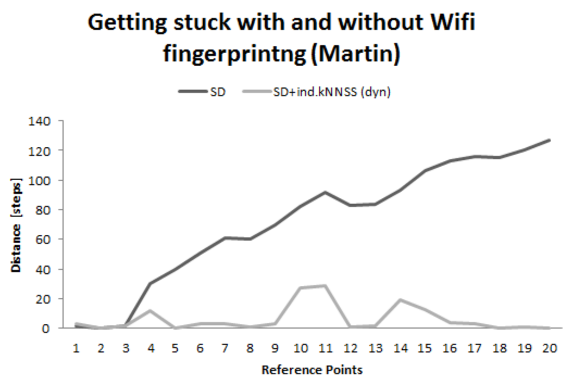
The figures 6.2 and 6.3 show boxplot diagrams of the median distance of the estimated position to the real one in terms of steps. The dynamic weighting is providing a better quality compared to the static one while having a smaller range of values. This means the dynamic weighting is showing less variance in quality. The maximum value of the SD only boxplot is greater than 80 and correspond to a walk where the SD had problems and did not leave a certain edge. We will describe this problem later on.

Using a well fitting step size during emulating the walk of Adrian the static weighting is reseting his position to a far away location because of an estimate of the Wifi fingerprinting. At this point the accuracy of the fingerprinting has been bad and estimated an outdoor position. The dynamic weighting can detect such bad estimates because the best fitting reference points are widely spread over the map. This allows even the worst dynamic weighted emulation to provide a better quality than the best static weighting does.

During the emulation of a walk of Martin the SD gets stuck in between two nodes. This happens because during the turn of Marting the SD detected steps and matches them towards the AH IV lecture hall (some steps beyond the second reference point). By that the SD matches the next steps to virtual steps on the edge and not to another edge. The estimation is then too far away from other edges resulting in increasing distances to the real position and not providing any useful data to the user. We add the Wifi fingerprinting to this emulation using dynamic weighting and achieve a good overall quality during the emulation.



(a)

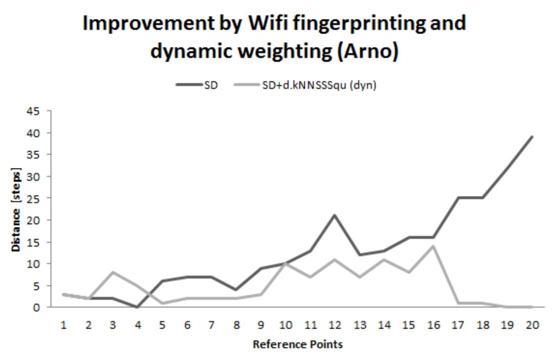


(b)

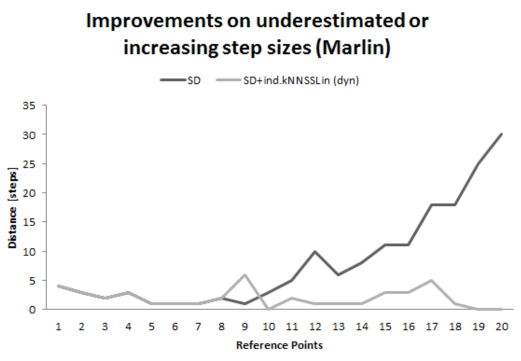
Figure 6.4 Figure 6.4(a): During an emulation of Adrians walk the best configuration of static weighting using SD and Wifi fingerprinting provides still a worse estimation of his position than the worst dynamic weighting configuration does. Figure 6.4(b): If there are problems during the SD the localizing mechanism can get stuck by constantly trying to walk through a dead end for example even if the step size is good. Adding another module can reset the estimated position enabling the system to continue estimating the position.

Dealing with fixed step sizes

Figure 6.5 shows a typical problem of the fixed step sizes used for SD. Slighty under- or overestimated step sizes will not be noticed at first but the accumulative error



(a)



(b)

Figure 6.5 Figure 6.5(a): A slightly imperfect setting of the step size can lead to slightly increasing inaccuracy. In the beginning the SD has been more precise in localizing the user but later on the accumulative errors of SD get serious as the weighting of two modules has a nearly constant and reliable accuracy. Figure 6.5(b) shows a walk of Marlin increasing his step size over time. The SD gets inaccurate after about half the way as the weighting of two modules provides a reliable quality.

is a serious problem. Furthermore, the user can vary her step size while walking. This could occur if the user is feeling unsure where to go and decreases the step size or is hurrying up because of being late for example. Even if the user tries to have a constant step size environmental influences like hills or sandy grounds still have influences on the step size. Using more mechanisms for localizing can decrease the accumulative error making it nearly a constant value figure 6.5 shows.

Another possibility to deal with the problem of step sizes is to use tagging codes like QR codes. The emulated walk represented in figure 6.6 uses two tagging codes (reference positions eleven and sixteen) reseting the users position and the accumulative error at the corresponding point in time.

Static weighting of Wifi fingerprinting and SD does not check the expected accuracy of modules and by that can lead to high distances of estimated and real positions as figure 6.7 shows. In chapter 4.5.2 we introduced mechanisms to estimate the accuracy of the Wifi fingerprinting approach. As a result the dynamic weighting reduces the weight of Wifi fingerprinting most of the time and provides a better localizing accuracy.

Dominik did some walks the SD has severe problems with. Varying step sizes, little cycles and some steps backwards did not allow the SD to provide a good quality in estimating the position in any emulation. Bringing more modules together is increasing the quality. Figure 6.8 shows the evaluation of using SD, Wifi fingerprinting, FloorColor and Mobile Tagging. In the evaluation, we use few reference points for FloorColor and only two tagging codes.

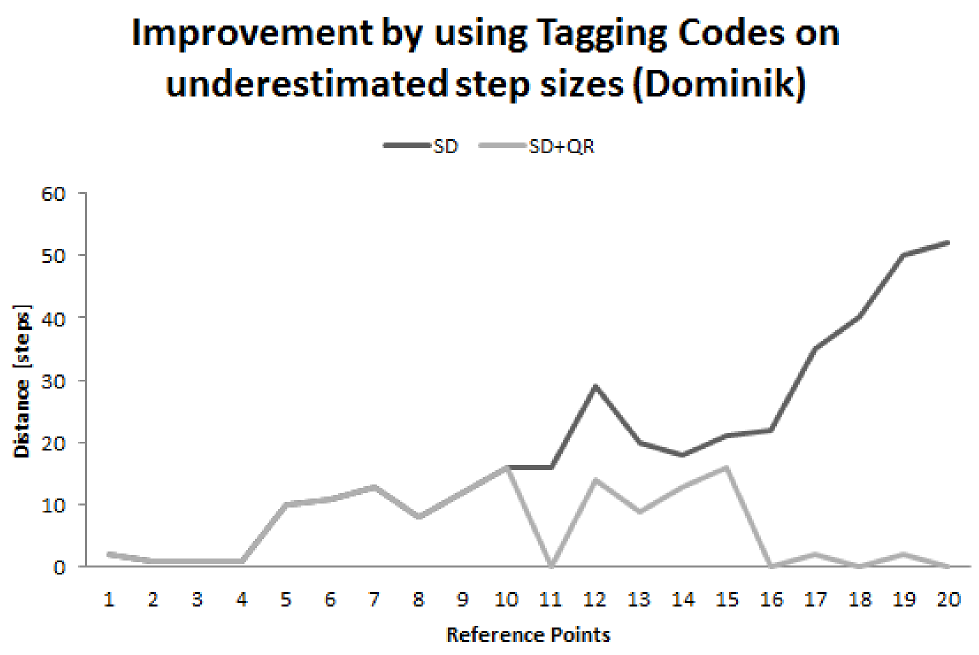


Figure 6.6 Using tagging codes to reset the users position can deal with underestimated step sizes if provided in a sufficient frequency.

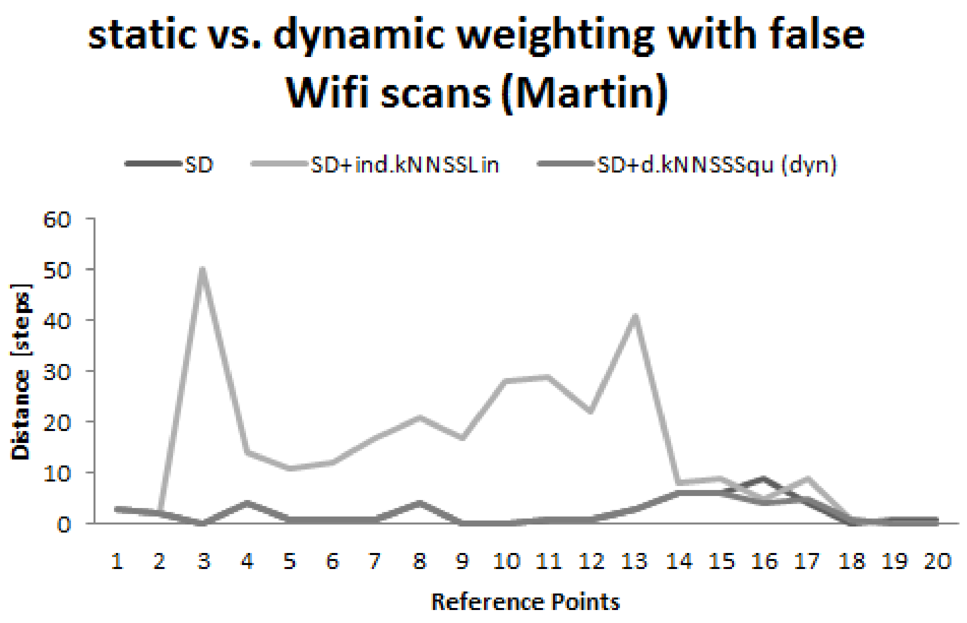


Figure 6.7 During this walk false and confusing Wifi signals have been detected. The best fitting positions of the references for fingerprinting are widely spread. The static weighting makes the localizing impossible. Dynamic weighting detects the inaccuracy of fingerprinting and provides a good quality of localizing.

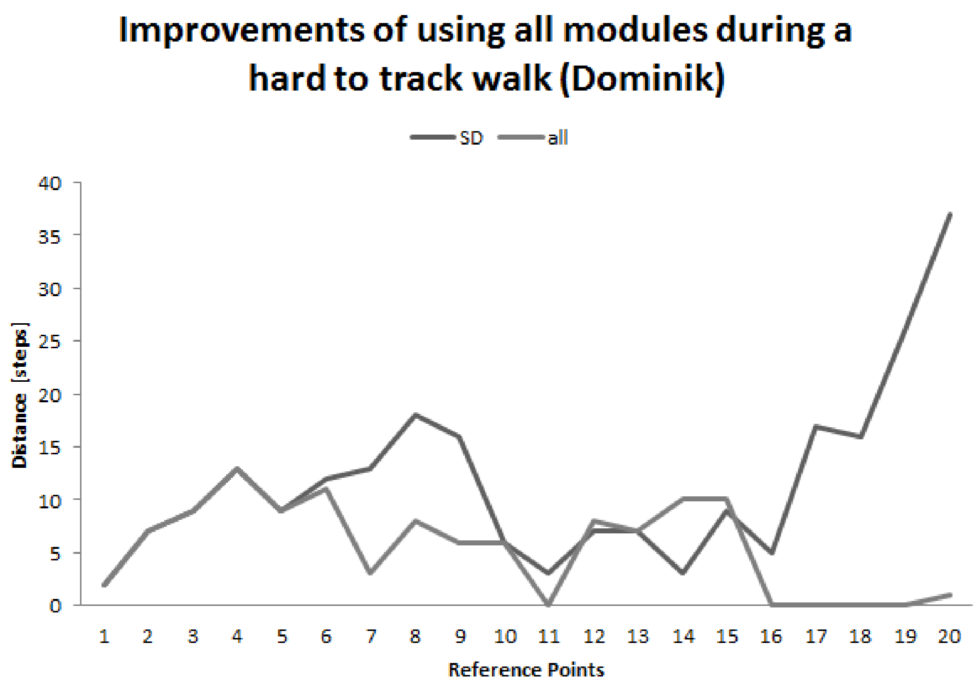


Figure 6.8 A walk that is hard to track of Dominik including walking backwards and little cycles in between. Using more modules is increasing the accuracy significantly. ‘All’ denotes the use of SD, Wifi fingerprinting, FloorColor and tagging codes.

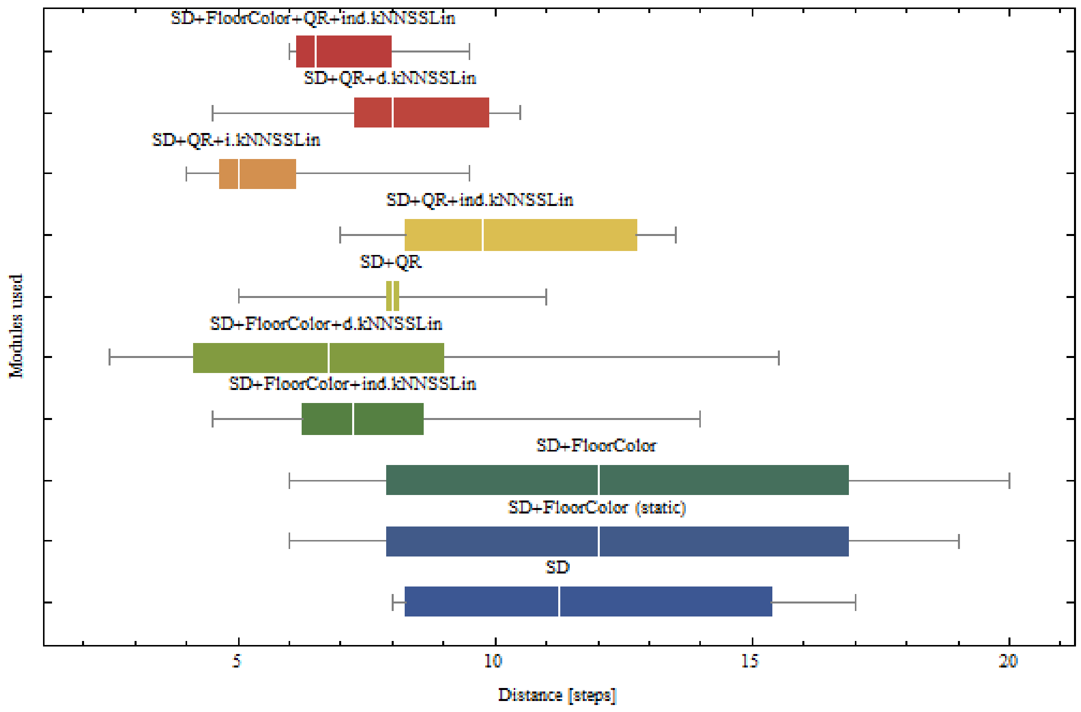
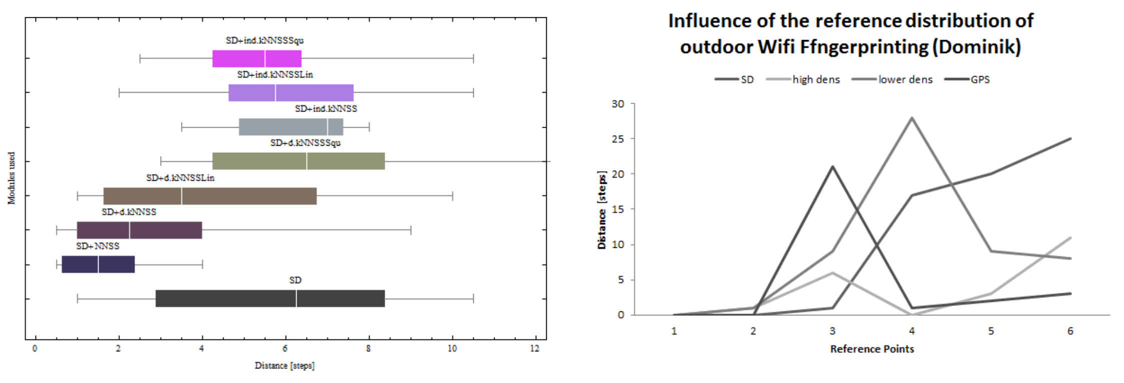


Figure 6.9 Using more localization mechanisms is providing a constantly better quality than SD only. The range of values is decreasing significantly when using more modules.



(a) (b)

Figure 6.10 Taking into account all GPS free emulations of the parking lot route (figure 6.10(a)) shows that by providing a dense reference point distribution the NNSS approach seems to be good. During one emulation the direct kNNSS with squared approximation heads inside the building and by that the highest maximum distance value is outside of this plot. Decreasing the density of the reference point distribution the values will increase fast which is presented in figure 6.10(b). For achieving the lower density of reference point distribution we only remove one fingerprint. This results in a worse quality.

Using the walk of Dominik, we start a set of emulations using different step sizes and parameters for the modules. Figure 6.9 shows boxplot diagrams of all emulations of Dominik's walk. Usually, the combination of more modules provided a better quality of the resulting position estimates. Furthermore the range of values is decreasing making the emulation more reliable and resilient to bad configurations and varying step sizes. Using all modules has always been providing a way better quality in estimating the users position than SD only.

6.2.2 Parking Lot

For reference reasons we do some walks and emulations concerning the parking lot at the west side of the computer science building seen in figure 6.1 (green route). The step size has the same influences on the overall quality of estimation as it had in the indoor experiment described above. Because the parking lot route contains a great cycle we need to configure the SD needs accurate to avoid the SD heading in the wrong direction or entering the building. We create four Wifi fingerprints each at a corner position of the parking lot. This is a dense distribution of reference points compared to the indoor environment.

Figure 6.10(a) and 6.10(b) show that the quality when using Wifi fingerprinting in our outdoor experiments depends on the distribution of reference points. Reducing the density leads to quality drops. Using GPS is usually providing a good quality as we expected outdoors.

6.3 Observations

One of the main aspects influencing the whole quality we identified is the initial step size. Only the best fitting step sizes enabled the step detection and matching approach to estimate the users position inside the room 2015. The combination of multiple localizing mechanisms made the whole system more resilient to varying step sizes and more often enabled the localization to estimate the user inside the room 2015 or at least the nearby room 2014. Besides not having located the user inside the right room this problem triggers the SD to describe a transition into a room on the other side of the floor when the users quits the room 2015.

The Wifi scans inside the room have a great distance in the signal space to each other. By that the accuracy of Wifi fingerprinting inside the room decreases. Using static weighting this leads to false estimates and position resets. We made similar observations in regions of entries and exits of the building. When standing in front of the computer science buidling the nearest neighbors in signal space have been the fingerprints outdoors even when having a real distance of 200 m. This is what we expected and therefore we defined the outdoor penalty and the check of the expected accuracy. By this we have a reduced weight of Wifi fingerprinting in those regions leading to a better overall quality of the resulting estimate when using the dynamic weighting function.

Using imperfect step sizes the SD accumulated errors and increased the inaccuracy over time. Combining mechanisms provide a more constant quality regarding those cases. Those other modules then help to reset the estimated position of the user to a more probable position. Observing those resets could improve the step size estimation dynamically in later implementations.

The static weighting of modules often provides good estimates on the users position but when having one inaccurate estimate this will influence the whole system in a bad way. Furthermore when having configured a perfect step size the static weighting decreased the quality of the resulting estimates several times. Most of these problems could be handled by the dynamic weighting. Using more mechanisms of location estimating results in a more reliable and resiliant localizing system.

6.4 Comparison

As we have shown the step detection and matching approach needs a very precise configuration of the step size. When having underestimated or overestimated step sizes the system will have an increasing error in localization. Adding more provider of information to the system as we do will increase the robustness and leads to a nearly constant error over time. Wifi fingerprinting needs a dense distribution of the reference points to achieve an accuracy of values similar to the step size. We have shown that even with a less dense structure the Wifi fingerprinting can refine the estimations of the step detection and matching resulting in a more robust and accurate localization service than both mechanisms provided on their own.

In [30] the combination of the localization approaches did not gain accuracy at first. After using an additional particle filter in their implementation for smartphone the

	Infrastr.	Config.	Resilience	Acc.	Tracking	Init
FootPath, PLATIN	++	- -	-	+	++	-
Wifi fingerprinting	o	+	-	+/-	+	+
Mobile Tagging	o	++	-	++	- -	-
GPS, pseudolites	- -	+	-	++	+	+
SD+Wifi [30]	o	+	+	+	++	+
Our System	+	++	++	+	++	+

Table 6.1 A Comparison of localization approaches taking into account the infrastructure that providers need to install additionally and the robustness of the mechanism to imperfect configurations like underestimated step sizes. Resilience denotes how the system can deal with environmental influences like people around, damaged sensors and more. The accuracy describes how accurate the estimates are and the tracking capability denotes wether the system can track a user walking around or only 'jumps' to another position. The ability to determine the starting position without help is evaluated in the auto init column.

accuracy was improved in comparison to the step detection alone. Our system does not use a particle filter but nevertheless improves the quality of indoor localization by using a weighting function as we have shown before. [31, 32] proposed how to combine Wifi localization and step detection by focusing on the human body and movement. They take into account the current posture and activity of the user to determine the quality of estimates and detections. By that they propose a human-oriented localization system that does not focus on the typical localization but discrete zones where the person does something. In contrast we do not take into account the posture but use another weighting and do not know about possible activies a person can do.

Table 6.1 compares approaches presented in this thesis and our approach. FootPath/PLATIN do need no additional infrastructure except a map of the building but have severe problems when having under- and overestimated or varying step sizes. The Wifi fingerprinting can use an existing Wifi infrastructure but needs a set of refernce points. The distribution of those reference points and the algorithm (NNSS, kNNSS, etc.) limit the accuracy. Mobile Tagging does need taggig codes installed at known positions and when scanning such a code the accuracy is good. But in between two codes the user can not be tracked. GPS and pseudolites provide an accurate localization service but the costs of infrastructure hardware and synchronization are high. The combination of step detection and Wifi fingerprinting presented in [30] combines the advantages of the two mechanisms and the ingerprinting robot decreases the infrastructure costs further because providers need to spend less time on creating fingerprints. If a provider installs a pseudolite system the Wifi fingerprinting approach will not take advantage of this. In contrast our system needs no additional infrastructure but can use already installed ones. As we have shown above the problem of step size estimation becomes less severe when having more mechanisms that intertwine. If a sensor or module is damaged or less accurate it can be disabled or weighted for resilience.

When using best fitting step sizes we achieve median accuracies during the walk of 1 to 10 m for the step detection approach only. Kothari et al[30] achieved a mean accuracy of 5.5 m for their step detection approach which is similar to our results. The mechanism they presented enabled their localization system to improve the accuracy by ∼27 %. Using only Wifi fingerprinting in addition to the step detection we have the same value of improvement. Regarding the emulations using more localization mechanisms we even achieve an improvement of more than 30 %. In comparison to the other approaches our system is extensible and more resilient and one can improve its accuracy by adding new modules.

6.5 Conclusion

In this chapter we did an evaluation of the concept of bringing location estimations together using static or dynamic weighting. Our results of about 1,500 emulations show the ability of improving the accuracy in comparison to a pure step detection and matching approach. We have given detailed information on some special cases. Furthermore we have shown that the dynamic weighting has great benefits and can increase resilience by using more mechanisms.

7

Conclusion

In this thesis we presented a way to combine several localization mechanisms aiming for increasing accuracy and resilience compared to single ones. For that, we defined and implemented multiple mechanisms and an extinsible weighting function. The evaluation showed that the approach works and is able to increase the accuracy of localizing the user inside a building.

The system's definition is extensible enabling us to introduce other mechanisms. Furthermore we can realize filters easily. By that, one can take out whole regions like restricted areas out of consideration automatically. These filters can be complex or fuzzy-queried based on the environment or the map data. We can enhance the dynamic weighting by behavioral rules for special cases. For example, when the map contains information about a strong magnet in range the weighting function can reduce the weight of mechanisms using magnetism (e.g. a compass).

As the step detection and matching has problems regarding the influence of the user's step size. The evaluation showed that we can solve these problems by adding more mechanisms reseting the position frequently or providing own localization estimates. Even when having a false step size or getting stuck other modules can reset the estimated position and reenable the step detection and matching approach.

One problem of implementing this system for smartphones is the consumption of resources. Smartphones only have limited energy resources and computational capabilities. Using a set of sensors is consuming a huge amount of energy. Doing this continuously will result in a fast decrease of the battery charge. Furthermore, complex modules and weighting rules will cause a high computational effort. Especially cheap and old smartphones may do not have sufficient processor power for this.

Taking everything into account we have presented a successful approach to increase the quality, reliability and resilience of user localization by dynamically weighting several mechanisms and bringing their estimates together.

7.1 Future Work

As mentioned before there are still problems. For these problems, one can find solutions in the future. Some areas people can make enhancements are:

Step Sizes: In our work we used a fixed step length causing the problem of estimating the right one. Furthermore the step length of a real person is not fixed but varies over time. There are approaches of dynamic step length detection like Autogait[39]. This could increase the overall performance of the system.

Energy Consumption: Using sensors and antennas continuously will strongly decrease the scarce energy resources of the mobile device. One possibility of decreasing the energy consumption could be to dynamically shut down modules. If a mechanism is providing only low weighted estimates, it may be shut down as long as it is not needed. To define such heuristics could increase the energy efficiency of the system

More Mechanisms: Of course implementing more mechanisms should be done to further increase the reliability and resiliance of the system

Tests and Special Cases: Doing more tests and evaluations may identify special cases the dynamic weighting function should react on.

User Feedback: Telling the user where her position is estimated enables the user to decide if the estimation is accurate. She can notify the system roughly about the estimates quality and eventually her real position. This can help the system to reset the estimated position to the real one and even allows a learning effect. The weighting function may learn about the sensoric and module quality over time and calculate the optimal step size backwards. By that the weighting function could optimize the weighting factors while the user is walking.

Bibliography

[1] OpenStreetMap. *OpenStrreetMap project*. Jan. 2013. URL: `http://www.openstreetmap.org`.

[2] OpenStreetMap. *OSM-Wiki, Planet.osm*. Jan. 2013. URL: `http://wiki.openstreetmap.org/wiki/Planet.osm`.

[3] OpenStreetMap. *OSM-Wiki: Slippy Map Tilenames*. Apr. 2013. URL: `http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames`.

[4] Immanuel Scholz OpenStreetMap and Dirk Stöcker. *OSM-Wiki: JOSM*. Apr. 2013. URL: `http://wiki.openstreetmap.org/wiki/JOSM`.

[5] Rat der Stadt Aachen. *Beschluss zum Bebauungsplan 901*. Ratsinformationssystem. Aug. 2008.

[6] Nvidia Inc. *Specification of Nvidia Shield*. Jan. 2013. URL: `http://shield.nvidia.com/`.

[7] Skalar International. *Specification of Polaroid iM1836*. Jan. 2013. URL: `http://www.sakar.com/products/16,cameras/101,polaroid-im1836`.

[8] International Data Corporation (IDC). *Android Marks Fourth Anniversary Since Launch with 75.0% Market Share in Third Quarter*. Press Release. Nov. 2012.

[9] Dirk Thissen. *Mobile Internet Technology*. 2012.

[10] Robert E. Beal. "Pseudolite-augmented GPS for locating wireless telephones". Patent US 6101178 (US). Aug. 2000.

[11] Klaas Bollhoefer. *Mobile Tagging mit 2D-Barcodes*. Tech. rep. Pixelpark AG, July 2007.

[12] ISO/IEC 18004. *Information Technlogy - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification*. Tech. rep. ISO/IEC, Sept. 2006.

[13] Denso Wave Inc. *QR Code Patent FAQ*. Jan. 2013. URL: `http://www.qrcode.com/en/faqpatent.html`.

[14] *Radio Frequency identification for item management*. Norm. 2000.

[15] *Near Field Communication - Interface and Protocol*. Norm. Mar. 2013.

[16] J. Laurie Snell Charles M. Grinstead. "Introduction to Probability". In: American Mathematical Society, 1997. Chap. Markov Chains (11).

[17] Michiel (Ed.) Hazewinkel. *Encyclopaedia of mathematics: an updated and annotated translation of the Soviet "Mathematical encyclopaedia" (Ed. 2001).* Dordrecht Boston Norwell, MA, U.S.A: Reidel Sold, distributed in the U.S.A., and Canada by Kluwer Academic Publishers, 2001. ISBN: 978-1-55608-010-4.

[18] Margaret L. Lial Raymond N. Greenwell Nathan P. Ritchey. "Calculus for the Life Sciences". In: Addison Wesley, 2003. Chap. Markov Chains. ISBN: 0-201-74582-8.

[19] Sebastian Jürg Göndör. *Building a Location Sensing and Positioning Service to enable Indoor Location Based Services.* 2010.

[20] Paramvir Bahl and Venkata N. Padmanabhan. "RADAR: An In-Building RF-based User Location and Tracking System". In: 2000, pp. 775–784.

[21] Paul Anthony Smith. *Tracking Progress on Path for Indoor Navigation.* Apr. 2011.

[22] Robert A. Wagner and Michael J. Fischer. "The String-to-String Correction Problem". In: *J. ACM* 21.1 (Jan. 1974), pp. 168–173. ISSN: 0004-5411. DOI: `10.1145/321796.321811`. URL: `http://doi.acm.org/10.1145/321796.321811`.

[23] J.A. Bitsch Link et al. "Indoor navigation on wheels (and on foot) using smartphones". In: *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on.* Nov. 2012, pp. 1 –10. DOI: `10.1109/IPIN.2012.6418931`.

[24] Felix Gerdsmeier. *Smarthpone-Based Multipath Indoor Navigation.* 2012.

[25] Alexandre Vervisch-Picois and Nel Samama. "Interference Mitigation in a Repeater and Pseudolite Indoor Positioning System". In: *J. Sel. Topics Signal Processing* 3.5 (2009), pp. 810–820.

[26] Rainer Mautz. *Indoor Positioning Technologies.* Vol. 86. Geodätisch-geophysikalische Arbeiten in der Schweiz. Publikationsreihe Astronomisch-geodätische Arbeiten in der Schweiz. Schweizerische Geodätische Kommission (Organ der Akademie der Naturwissenschaften Schweiz), 2012.

[27] TotalTrax Inc. *Sky-Trax System.* Feb. 2013. URL: `http://totaltraxinc.com/index.php/smart-forklift-solutions/forklift-tracking/sky-trax`.

[28] Sebastian Tilch. *CLIPS - Prinzip.* Jan. 2010. URL: `http://www.geometh-data.ethz.ch/publicat/diploma/2010_Tilch/prinzip.html`.

[29] Jaewoo Chung et al. *Indoor Location Sensing Using Geo-Magnetism.* 2011.

[30] Nisarg Kothari et al. "Robust Indoor Localization on a Commercial Smart Phone". In: *Procedia Computer Science* 10 (2012), pp. 1114–1120.

[31] Alberto Alvarez-Alvarez et al. "Human activity recognition applying computational intelligence techniques for fusing information related to WiFi positioning and body posture". In: *FUZZ-IEEE.* IEEE, 2010, pp. 1–8. ISBN: 978-1-4244-6919-2.

[32] José M Alonso et al. "Towards People Indoor Localization Combining WiFi and Human Motion Recognition". In: *XV Congreso Español sobre Tecnologías y Lógica Fuzzy.* 2010.

[33] U.S. Census Bureau. *Geographic Information Systems FAQ*. moved. Feb. 2013. URL: `http://www.movable-type.co.uk/scripts/GIS-FAQ-5.1.html`.

[34] Wikipedia. *Stochastic Matrix*. Apr. 2013. URL: `http://en.wikipedia.org/wiki/Stochastic_matrix`.

[35] Donald A. Norman. “The Design Of Everyday Things”. In: Basic Book, 2002. Chap. Knowledge in the Head and in the World.

[36] Donald A. Norman. “The Design Of Everyday Things”. In: Basic Book, 2002. Chap. The Psychopathology of Everyday Things.

[37] Google Inc. *Android SDK | Andoroid Developers*. Apr. 2013. URL: `http://developer.android.com/sdk/index.html`.

[38] Android SDK Tech Lead Xavier Ducrohet. *Android 2.2 and Developers Goodies*. Apr. 2013. URL: `http://android-developers.blogspot.de/2010/05/android-22-and-developers-goodies.html`.

[39] Dae-Ki Cho et al. “AutoGait: A mobile platform that accurately estimates the distance walked”. In: *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*. 2010, pp. 116–124. DOI: `10.1109/PERCOM.2010.5466984`.

A

Appendix

A.1 CD Contents

Thesis: This thesis in Portable Document Format (PDF).

dCollector: The source code of the dCollector Android application.

WiFiWar: The source code of the WiFiWar Android application.

EXILE: The source code of the EXILE application including the complete Javadoc source code documentation.

Datasets: The datasets of walks (recorded using dCollector), environmental information (Wifi fingerprints collected using WiFiWar, Floor Colors, etc.) and maps (OSM Files).

Evaluation: Several Excel sheets containing the collected and evaluated data of emulations using EXILE. A Mathematica Notebook for creating the BoxPlot diagrams.

Music: Special bonus tracks for the limited thesis edition. *Star Of A County Down* and *Rocky Road to Dublin (live)* performed by the Irish Folk band Ceirnin.