Information Security

WiFi Compass

WiFi Access Point Localization with Android Devices

Master Thesis

for the degree of

Diplom-Ingenieur

submitted by

Thomas Konrad, BSc & Paul Wölfel, BSc

1010619503 & 1010619517

as part of the

Information Security program at St. Pölten University of Applied Sciences

Mentor: Dipl.-Ing. (FH) Mag. Rainer Poisel, ACE

Vienna, May 31, 2012

(Authors' signatures) (Mentor's signature)

Declaration of Honor

We affirm that

- we have drafted this degree dissertation independently, have not used other than the stated sources and means and further, have not used any other illegitimate assistance.
- we have, neither domestically nor abroad, submitted this degree dissertation to an examiner for inquiry or in any other form as a test paper.
- This paper is identical with the paper, which has been assessed by the examiner.
- Hereby we grant St. Pölten University of Applied Sciences (Fachhochschule St. Pölten) the exclusive and spatially unrestricted right of use to this degree dissertation, for all kinds of uses, and retain the right to be referred to as author of this work.

Vienna, May 31, 2012

_____

(Authors' signatures)

Zusammenfassung

WiFi Compass ist eine Android-Applikation, welche entwickelt wurde, um IEEE 802.11 Wireless LAN-Access Points in Indoor-Umgebungen zu lokalisieren. Hierfür bewegt sich der Benutzer durch ein Objekt und nimmt Messungen der Signalstärken der verschiedenen Base Service Set Identifier (BSSIDs) auf. Diese Informationen werden genutzt, um mittels Trilateration die wahrscheinlichste Position des Access Points zu bestimmen. Die aktuelle Position des Benutzers kann wahlweise mittels Schritterkennung durch die in Android-Geräten verbauten Beschleunigungssensoren und Magnetfeldsensoren oder manuell anhand eines Gebäudeplans am Bildschirm bestimmt werden. Die Diplomarbeit beschreibt die der Applikation zu Grunde liegenden Algorithmen und deren Anpassungen für WiFi Compass. Für die Positionsbestimmung des Benutzers und die Berechnung der Position der Access Points werden jeweils mehrere Methoden aufgezeigt und in einem beispielhaften Szenario getestet. Jene Methoden, die sich in der Praxis als die besten herausgestellt haben, wurden in der Applikation als Standard-Algorithmen implementiert.

Abstract

WiFi Compass is an Android application, which has been developed to localize IEEE 802.11 Wireless LAN access points in indoor environments. Therefore, the user moves through a building and measures the signal strength of different Base Service Set Identifiers (BSSIDs). This information is then used to estimate the positions of the access point using trilateration. The current user position can be determined both automatically by using the built-in acceleration sensor and magnetic field sensor of the Android device or manually by setting the user position on the map. The thesis contains detailed descriptions and adjustments of the algorithms that serve as a base of WiFi Compass. For both the user tracking and the access point position estimation, several different methods were identified and tested in a real-world scenario. The methods, which have shown to perform best in practice, have been implemented as default algorithms in the application.

Table of Contents

1 Introduction

This document describes an Android application called “WiFi Compass” developed by Thomas Konrad and Paul Wölfel and its theoretical background as part of their pursuit of a Master’s degree at the University of Applied Sciences in St. Pölten. The target of the application is to calculate the position of access points in indoor environments with the use of IEEE 802.11[1] WiFi signal strength measurements. To help the user estimate their own position, the device tracks the movement of the user using different types of sensors of the device.

The research questions concerning the positioning of the Android device (the user tracking) are as follows:

> *„Which methods for Android smartphone localization and movement tracking are feasible for indoor use?“*
>
> *„Which of these methods are feasible for a standalone Android application?“*

Concerning the estimation of the access point positions once the measurement data (measuring positions and signal strength data) has been collected, the research questions are as follows:

> *„What are existing methods for calculating the position of Wi-Fi access points using position and signal strength data and how exactly do they work?“*
>
> *„How can one of these methods be implemented in an application for Android mobile devices, to which extent are the results accurate and what does the accuracy of the calculation primarily depend on?“*

In this thesis, an attempt to answer the above questions is done. Moreover, a description on how the theoretical results have been built into a ready-to-use Android application is included.

The two major parts of this work are chapters 2 and 3.

- Chapter 2: Positioning of the Android device (written by Paul Wölfel)
- Chapter 3: Access Point Trilateration (written by Thomas Konrad)
- Chapter 4.1 The Multi-Touch User Interface (written by Thomas Konrad)
- Chapter 4.2 The Model (written by Paul Wölfel)
- Chapter 4.3 Sample Use Case (written by Thomas Konrad)
- Chapter 4.4 Project Website and Source Code (written by Paul Wölfel)

The other chapters are combined work of both authors.

[1] IEEE 802.11 is a set of standards for wireless networks. See http://www.ieee802.org/11/ for details.

2 Positioning of the Android Device

The main target of this master thesis is to develop an Android application, which is able to calculate the position of access points by measuring their signal strength. To calculate the position of an access point, the received signal strength indication (RSSI) of IEEE 802.11 WiFi signals can be used. The positions of the measurements and their results are used to calculate the location of an access point. This chapter tries to find a feasible solution to determine the position of the Android device.

2.1 Basic Principles of Localization

A position definition must have a position system, in which the position is defined. The typical positioning system is a coordinate system with three dimensions, also called “frame”. These coordinates are relative to a world frame, a local frame or a body frame. A body frame would be the relative frame, like a building or a floor. In the three-dimensional coordinate system, the axis are described as x, y and z [**1, pp. 18-20**], [**2**], [**3, p. 4**]. A point in this system has an x, a y and a z coordinate, which can also be represented as a vector from the origin (0,0,0) to the point.

2.1.1 World frame

The coordinate system used to describe a position on the world is called world frame. It is defined as follows [2]:

- *“Xw is defined as the vector product Yw.Zw (It is tangential to the ground at the device's current location and roughly points East).*
- *Yw is tangential to the ground at the device's current location and points towards the magnetic North Pole.*
- *Zw points towards the sky and is perpendicular to the ground.”*

The index w is used to annotate that this is a world frame coordinate. Figure 1 shows the world frame as a graphic.

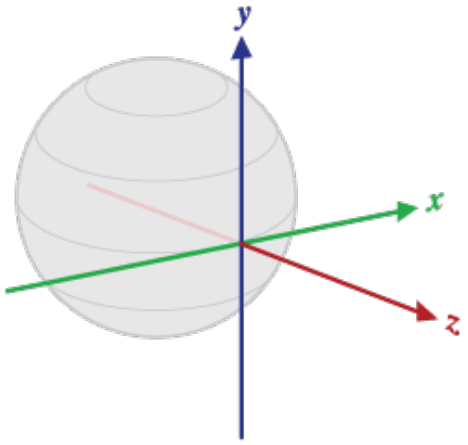

Figure 1: World frame defining x,y and z axis [2]

2.1.2 Local frame

For such uses as indoor navigation systems, the world frame would be quite large and in most cases, it is more feasible to use a relative coordinate system called “local frame” [**1, p. 19**]. The local frame defines all points with relative coordinates to the origin. This can be used to define points inside a building or on a map. The index l is used to annotate that this coordinate is a local frame coordinate.

Typically, the x_l and y_l axis are used to define the position on a map, and the z axis to define the altitude over ground. The y and x axis of the world frame can match the x and y axis of the local frame, but in some cases it is more feasible to align the axis to the building, like shown in Figure 2. To align x_l to y_w in this example, the map has to be rotated 53° counterclockwise.

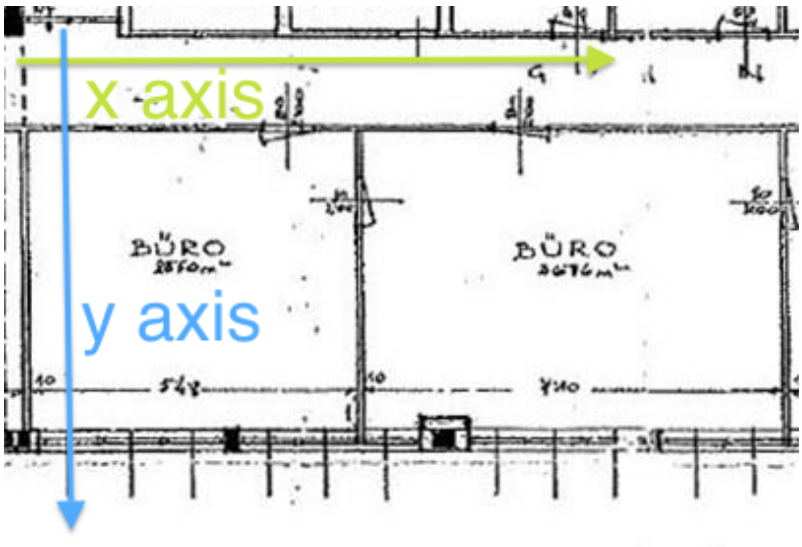

Figure 2: local frame example with a map

2.1.3 Body frame

The body frame defines the local coordinate system for a handheld device. The axes are defined in the Android Sensor API as follows [**4**]:

- "x_b: horizontal axis from the left to the right to the device.
- y_b: vertical axis from the bottom to the top of the device.
- z_b: axis from the backside to the front side of the device"

This frame is shown in Figure 3 and is used by

- the acceleration sensor,
- the gravity sensor,
- the gyroscope,
- the linear acceleration sensor, and the
- geomagnetic field sensor.

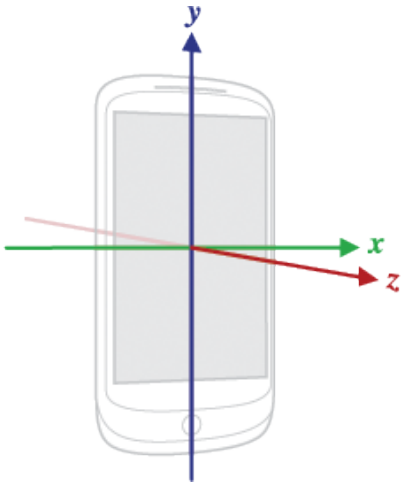

Figure 3: Body frame of Android device [4]

The gyroscope measures rotation along these axes. The rotation is positive in the counterclockwise direction, like defined in the Android Developer Guide [**5**]:

> *"Rotation is positive in the counter-clockwise direction; that is, an observer looking from some positive location on the x, y or z axis at a device positioned on the origin would report positive rotation if the device appeared to be rotating counter clockwise. This is the standard mathematical definition of positive rotation and is not the same as the definition for roll that is used by the orientation sensor."*

2.2 Methods to determine the position

The calculation of access points is depending on a correct position of the measurements. Every erroneous measurement position will also falsify the position of the access point. As a result, it is important that the positioning system provides accurate data for the calculation algorithm. There are several positioning systems available, which can be used in Android smartphones.

2.2.1 Global Positioning System (GPS)

The Global Positioning System uses a plurality of satellites and radio waves to position a device on either a two-dimensional or a three-dimensional space on the surface of the earth. [**6, p. 1**] The position is represented by latitude (X), longitude (Y) and altitude (Z) [**6, p. 7**] as also described in section 2.1.1.

The position of a device is determined by trilateration with the quasi-distance of the satellites. This quasi-distance is calculated with the time offset and position of each received radio wave signal from a GPS satellite. The time offset is calculated from the time of the receiver and the received timestamp [**7, pp. 1-2**]. The time sent by the satellite is mostly accurate, because these contain several atomic clocks, from which the time is derived. With the knowledge of the speed of light and the time offset the distance between the mobile device and the satellite could be calculated. This means an inaccurate time of the mobile device would lead to a wrong distance to the satellite and so to a wrong position. The solve this issue, the clock of the mobile device can also be synchronized with the help of GPS signals up to a few nanoseconds as described by Hahn and Powers in [**7, pp. 1-2**].

To determine the position of a device, i.e. an Android Smartphone, on the surface on the earth, there are 3 satellites required as a minimum, because the unknown quantities are three dimensional: the two-dimensional position and the error of the mobile device clock. To also calculate the altitude of a device, a fourth satellite is needed, because another axis is added to the equation [**6, p. 7**].

2.2.2 Trilateration

There are many systems, like RADAR [**8**], systems described by Lassabe et al. [**9**] or Chintalapudi et al. [**10**], which make use of IEEE 802.11 WiFi signals to determine the position of a device. Each of these systems collect signal strength data of devices or access points and combine these values with trilateration to estimate the position of a device. The signal strength is converted into a distance with a signal propagation model from the device to the access point. A circle where the radius equals distance from the measuring point is estimated. When at least 3 measurements are combined, the intersection of the circles gives the possible position of the access point [**11, p. 1**].

RADAR relies on a pre-deployment phase [**8, p. 2**], which is described in more detail in 2.2.3 Fingerprinting. This takes some time and depends on a static infrastructure, which does not change. To accommodate these limitations, several algorithms have been developed. Lassabe et al. calculate the position of a device by using the distance to access points, calculated from the signal strength of those. The distances are used to triangulate the device.

Chintalapudi et al. take a different approach [**10, p. 3**]. Signal strength measurements of all near access points to a mobile device are combined to calculate a relative position of all devices. If the absolute position of a device is also known, the absolute positions of all devices can be determined.

For devices, which are not capable of receiving and transmitting 802.11 WiFi signals, other radio frequency trilateration algorithms have been developed [**12**]. However, the authors will further concentrate on WiFi based systems in this work and not discuss other RF based systems.

2.2.3 Fingerprinting

Another approach to localize a mobile device is to use a fingerprint to identify a location. A fingerprint consists of specific data linked to point in a frame system. This data has to be collected in a pre-deployment phase as described by Bahl and Padmanabhan [**8, pp. 1-3**] or Liu and Yang [**13, pp. 1-2**]. These two systems analyze IEEE 802.11 signal strength and signal noise ratio to get specific characteristics of a point. In the real-time phase, where mobile device are located, this characteristics are compared to measurements of the client. The location of the device will be the best match.

Bitsch et al. use Fingerprinting and step detection to estimate the position on a path with an Android device [**14, pp. 2-7**]. The initial position of the user must be known by the device or entered by the user and the user defines the destination. Footpath automatically calculates the best route and estimates the steps taken on this path. Each step and its direction is being recorded and matched against the planned path as shown in Figure 4. The position with the highest probability is the calculated position of the user. The matching against the path and the definition of the probability are further discussed by Bitsch et al. [**14, pp. 3-7**], but will not be discussed further by the authors of this work, because the path is not known in unknown locations and therefore path matching is no applicable for WiFi Compass.

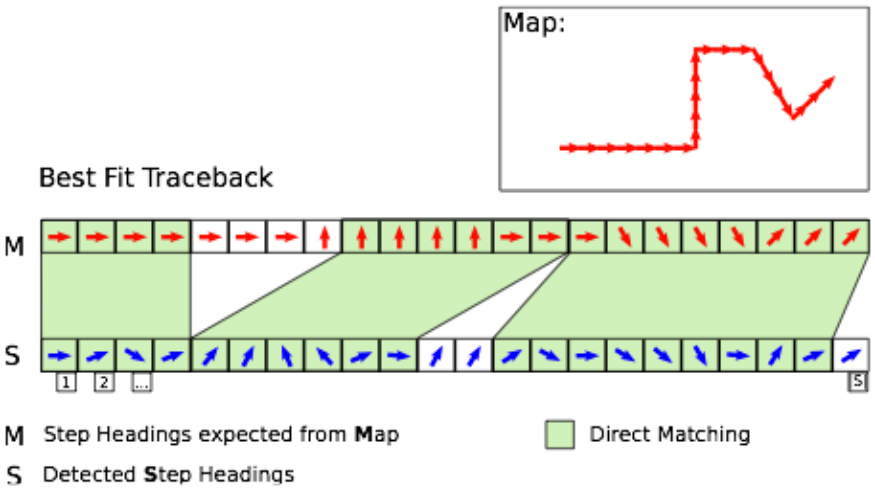

Figure 4: "Matching sequences of detected steps onto sequences of expected steps using Best Fit. We use sequence alignment to find the best match between the sequences. Unmatched parts correspond to overestimated and underestimated step lengths." [14, p. 4]

2.2.4 Proximity

Proximity can be used in networks to locate a device, if the location of fixed nodes is known [**15, pp. 1-2**]. In Proximity Systems, a device is "in-range" of another one, if the receiver can demodulate and decode a packet sent by the transmitter [**16, pp. 2,3**]. The information whether a mobile device is "in-range" or "out-of-range" can be used to calculate its location [**16, p. 5**], [**15, p. 2**].

2.2.5 Inertial Navigation Systems

Inertial Navigation Systems are used in airplanes and are based on accelerometers and gyroscopes as defined by Lerman et al. in US Patent 3260485 [**17, p. 11**]. These systems are based on Newtons laws of motion, which state that every object, which is in movement, will continue uniformly in a straight line unless disturbed by an external force acting on the object [**18, pp. 2-3**]. If the initial speed and all forces acting on the object is known, its movement can be estimated. Furthermore, if the initial location of an object in a frame is known, the position can be tracked over time.

2.3 Feasibility Check for WiFi Compass

This master thesis tries to assess which indoor localization methods are feasible to be used in an Android application. To be feasible for this specific use case, an algorithm has to meet the following conditions, which are derived from the research question:

1. **Indoor Use**: The algorithm can be used indoor.
2. **Android Application**: The software must be suitable to run on low-performance systems, such as Android mobile phones or tablets.
3. **Unknown Locations**: The algorithm must be suitable for unknown locations.

As stated in the first research question, the algorithm used in this research should be feasible for indoor use. “Indoor” in our research refers to the inside of an office building, where the tests of the software and sensors are also conducted. This does not exclude the use in other building types, such as storage facilities or private housing, or even outdoor use, but the application’s main targets are office buildings. This requirement is defined as the first condition, which should apply to the algorithm.

The second condition states that the computation should be suitable for an Android mobile device such as a mobile phone or tablet. This is derived from the second research question, which requires the algorithm to be feasible for a standalone Android application. The application should be standalone; so no other central server or special hardware should be required to calculate the position of a user. Some algorithms [**10, p. 7**] require a central entity, which consolidates the data collected and calculates the position of a device.

As WiFi Compass tries to develop a solution, which should be suitable for unknown locations, a central unit or specific infrastructure would be contradictory. As a result, all computations must be done on the Android device and the algorithms should be feasible as an Android application. This implicitly states that the location calculation should not take longer than a few seconds. If the calculation would take longer, it would negatively affect usability and the time effort which has to be invested to find an access point. These requirements are defined as the second and third condition for a suitable algorithm.

To ensure good usability, the algorithm should also require no or only a minimum of configuration, before being used. If an algorithm must be extensively calibrated, the time effort and the knowledge requirement for localization of an access point grows and makes the application usable for a smaller group of people. This should be considered in the evaluation of a feasible algorithm to locate and track the user.

2.3.1 Global Positioning System (GPS)

GPS was developed for outdoors use and cannot be used indoors, because the GPS signal strength is too low to go through a building [**19, p. 6**] [**20, pp. 21-22**]. Although there are several approaches to improve signal reception, an accuracy of 20-25 meters is not enough for our application [**21, p. 9**].

Indoor GPS addresses the issue that the GPS signals from satellites are too weak for indoor reception by adding indoor pseudo-satellites. These “pseudolites” are devices which send GPS

signals in an indoor environment, to enable devices with this signals to triangulate their position [**22, pp. 7-13**].

However, this requires special hardware to be in place, which does not fit the requirements of WiFi Compass. As a result of the second condition, GPS is not applicable for our approach.

2.3.2 Trilateration

Trilateration based localization or tracking systems are often based on a “pre-deployment” phase, like RADAR [**8, p. 2**]. The feasibility condition number 3 for WiFi Compass requires the algorithm to work in unknown locations. This excludes algorithms which require knowledge about the infrastructure, which RADAR does. As a result, Trilateration algorithms, which are based on the infrastructure devices or a pre-deployment phase, are not feasible for WiFi Compass.

The approach of Chintalapudi et al. [**10, p. 3**] overcomes the requirement for a pre-deployment phase by building a relative map of all devices, including mobile devices, and fixed devices like access points. This even rules out the requirement that the location must be known when a measurement is done. The location of the measurement can be calculated if enough data is available. The algorithm developed by Chintalapudi et al. is designed to be used by several mobile devices and a central server [**10, p. 7**]. This conflicts with the second condition of feasibility for WiFi Compass and prohibits the use of the algorithm in its original design.

In the future, this could be adopted to combine the client and server part in one Android application, in order to eliminate the requirement of a dedicated server. Another approach would be a peer-to-peer architecture, where data is exchanged between mobile devices, and combined to localize and track the user. These approaches are not taken in the initial version of WiFi compass, because of the complexity and the amount of work required to adopt this algorithm to the application. This does not rule out the possibility that it is done in a future version of WiFi Compass.

2.3.3 Fingerprinting

Fingerprinting is a two-phase approach to localize a device based on radio frequency signals. In the first phase, also called “pre-deployment” or “offline”, data is collected which is then used in the second phase, called “real-time”, to localize a device [**8, p. 2**]. WiFi Compass requires an algorithm, which is able to localize a user without a “pre-deployment” phase. As a result, Fingerprinting is not a feasible approach for WiFi Compass.

2.3.4 Proximity

The Proximity approach to localize a device is based on the data which device is close to one or more reference devices [**16, p. 5**]. Because the location of these reference devices is known, the location of the device can be estimated. However, this requires an existing and well-known infrastructure, which is not a feasible approach for WiFi Compass and ruled out by the second condition (Android Application).

2.3.5 Inertial Navigation Systems

Inertial Navigation Systems are based on Newton's Laws of physics and determine the position of an object with the knowledge of the initial position and the forces applied to the object [**18, pp. 2-3**]. If the initial position is not known, only the relative position to the origin can be determined. With this limitation, an Inertial Navigation System cannot be used solely to provide a device localization and tracking system. The INS can track the device movement over time, but needs the initial position provided by another system.

INS are not limited to indoor or outdoor use, because the force applied to a device can be measured on this device and does not need an external source of information. As long as the laws of physics apply, an Initial Navigation System can be used. This fulfills the first feasibility condition (Indoor Use) and the third condition (Unknown Locations) of WiFi Compass.

Step Detection is one Inertial Navigation System, which are already available as Android applications. The Android application "Footpath" developed by Bitsch et al. [**14, p. 3**] uses Steps, detected with an accelerometer and a compass, to follow a path in a building. Figure 5 shows the raw data processed by the step detection algorithm of Footpath and the detected steps. The project is published as open sources software[2] and available as proof for a working step detection system. As a result, the second feasibility condition (Android Application) applies to Step Detection as an Inertial Navigation System.

[2] Open Source Project Footpath on GitHub: https://github.com/COMSYS/FootPath

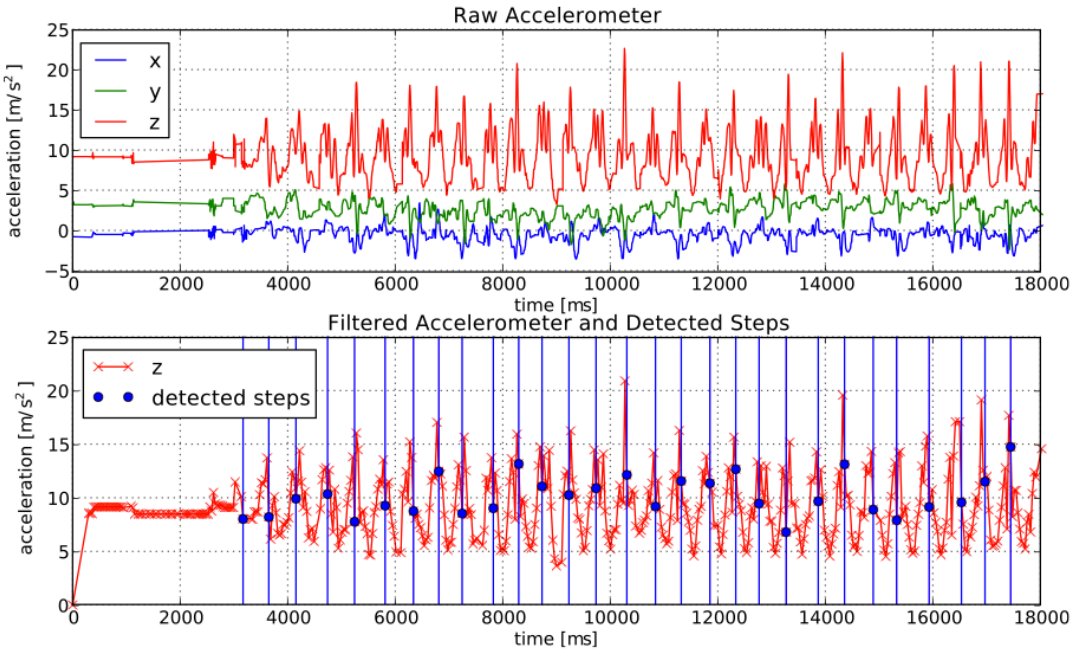

Figure 5: step detection by Footpath [14, p. 3]

To solve the research problem an algorithm must be able to localize an Android smartphone and track the movement solely with an Android application. Step Detection could be implemented as an Android application and track the movement. To initially localize the device, an external source must be used or only relative positions are available.

This is not an issue to WiFi Compass, because if the location is completely unknown, relative positions from the starting point of all measurements and the access points are enough to find an access point. The coordinates of an object do not have to be linked to a global coordinate system, like GPS, because the main target for WiFi Compass are indoor environments. These environments have physical boundaries and only require positions which are inside this local frame. The starting point of measurements could be the origin of a two-dimensional coordinate system (x:0, y:0) and all absolute positions could be determined with the knowledge of the real position of the origin. If the location is well known and a map can be used to visualize the location, the user can initially define the position.

With this caveat, a step detection algorithm could be used to localize the user and is implemented in the first version of WiFi compass.

2.4 Realization in WiFi Compass

WiFi Compass depends on accurate position data of the measurements of IEEE 802.11 radio frequency signals. Without correct locations of measurements, the data cannot be combined to calculate the location of an IEEE 802.11 access point. This chapter describes how the required locating and tracking of the Android device are accomplished.

2.4.1 Basic User Interface to locate and track a user

WiFi Compass tries to provide usability and visibility of the tasks to collect data and the results. All elements that are needed to localize an access point are placed on a multi-touch user interface. The basic frame, which is presented as a grid, is the coordinate system also referred to as “local frame”. The local frame has the coordinates (x:0,y:0) at the top-left corner. All coordinates are relative to this point. On this grid the user icon, measurements, known access points, and detected access points are drawn, as shown in Figure 6.

The user icon with the compass visualizes the current location of the device in the local frame. Access points defined by the user are drawn as a green access point icon and calculated ones as red access point icon. If the icons are clicked, information about the access point and its relative position are shown. Measurements are presented as blue footprints. If clicked, all SSIDs and BSSIDs with the associated channel and additional information are presented.

Two buttons, which allow the user to start a single scan or step detection and continuous scanning for WiFi networks, are located on the top of the screen.

Figure 6: WiFi Compass Scan User Interface

If the location where the scan is carried out, is indoor and not known a grid will be utilized as a background. The grid helps navigating and provides a way to read the distance from the display. Fetching of maps may be possible in public buildings with Open Street Map, as described by Bitsch et al in Footpath [**14, p. 2**]. The authors decided not to implement Open Street Map and

focus on the location tracking and calculation of access points. If the location is known and a map exists, it can be loaded as a background image to help associating points on the map to real locations. This can be really helpful, because the initial position has to be defined by the user. The user has to drag the icon on the map to the current position as shown in Figure 7. From this point on the location can be determined by the Step Detection Algorithm as described in chapter 2.4.2.

Indoor maps are often not aligned to the magnetic north so walls are vertical and horizontal aligned to the sheet of paper. To define the magnetic north of the map, an icon shows the adjustment angle. To adjust the angle, the device itself can be rotated as shown in Figure 7. This is needed because the compass detects the direction of the user movement.

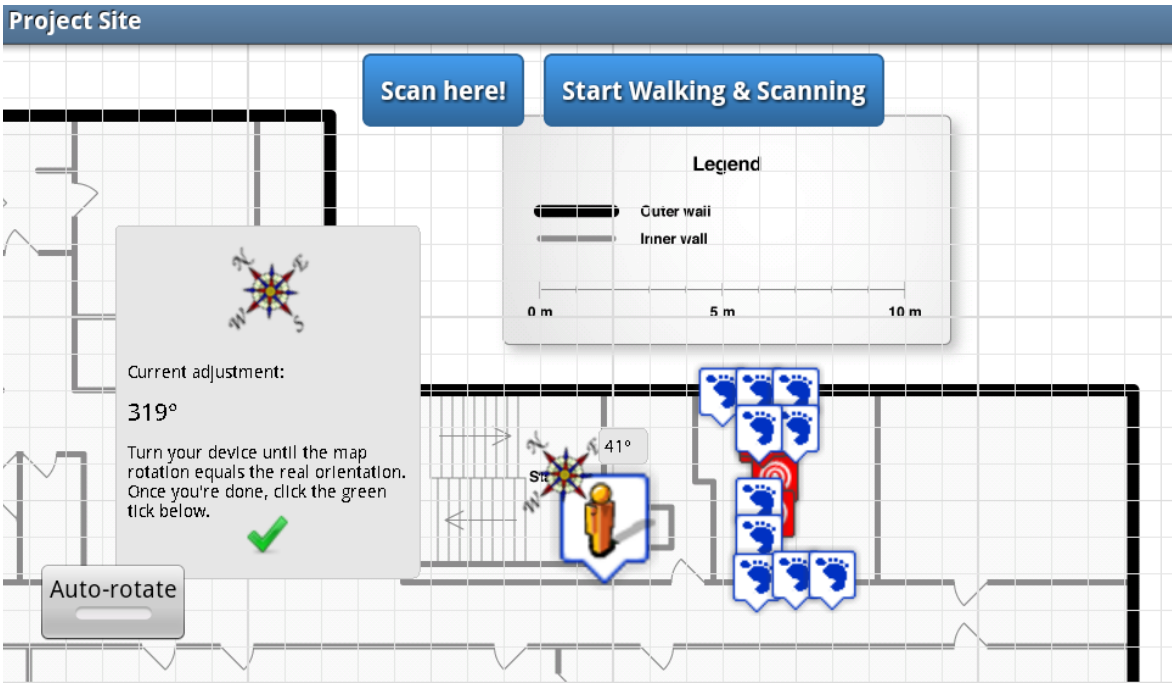

Figure 7: WiFi Compass User Interface with map loaded

Step Detection needs to know how many pixels equal one meter, because the user icon position is always defined in pixels. The algorithm uses the proportion (grid spacing) to calculate the distance of a step. The default grid spacing for one meter is $g = 30$. The user can change this value as shown in Figure 8 and Figure 9.

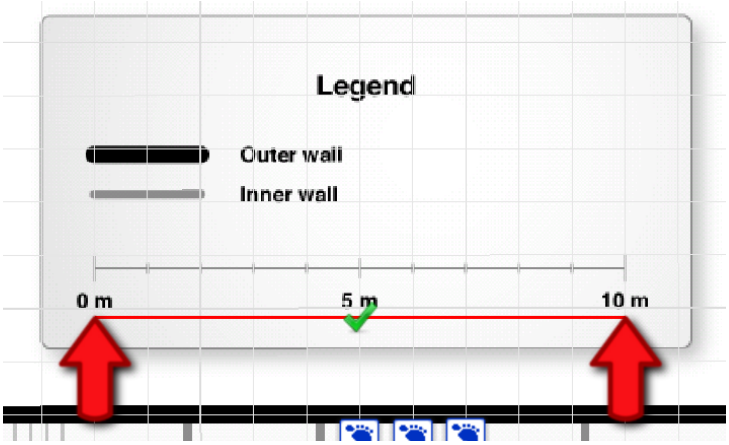

Figure 8: Distance selection

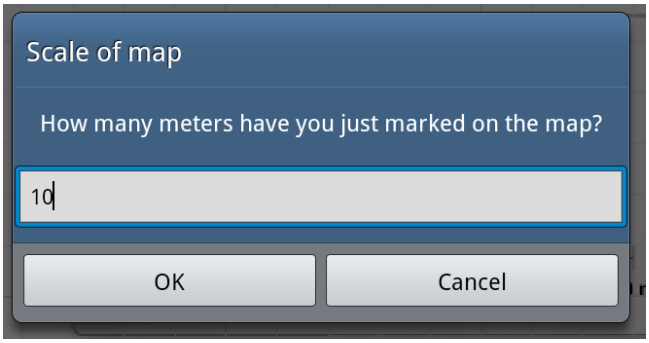

Figure 9: Defining proportion of pixels to meters

With the initial position, the magnetic north and the proportion of pixels to meters set, the step detection algorithm can be used to track the movement of the user.

2.4.2 Step Detection Algorithm

The Inertial Navigation System implemented by WiFi Compass is based on step detection with the accelerometer and the direction of the compass. These sensor values can be obtained by `SensorManager` **[2]**.

The step detection implementation is based on the step detection algorithm implemented by Paul Smith and Jó Ágila Bitsch in FootPath[3]. This Implementation provides a `StepDetection` class which accesses the accelerometer and the compass to detect a step. The sensor data is sampled at the fastest possible rate provided by the sensor. These values are lowpass-filtered to only include major movements as shown in Listing 1. The collected data is analyzed 30 times

[3] Open Source Project Footpath on GitHub: https://github.com/COMSYS/FootPath/

per second; this results in a sample time of approximately 33 ms. The `StepDetection` class from FootPath was rewritten to separate the algorithm (`StepDetector`) from the Sensor listener (`StepDetection`). This is necessary because the algorithm is also used in auto calibration with persisted data.

```
1       public synchronized void addSensorValues(long timestamp,float values[]) {
2               // simple lowpass filter
3               lastAcc[0]+=a*(values[0]-lastAcc[0]);
4               lastAcc[1]+=a*(values[1]-lastAcc[1]);
5               lastAcc[2]+=a*(values[2]-lastAcc[2]);
6               lastUpdateTimestamp=timestamp;
7       }
```

Listing 1: Lowpass filter

Steps can be detected by their characteristic change of acceleration [**14, p. 3**]. If a user moves forward, the device moves up and down in their hand. Experiments have shown that the amount of movement depends on the user and how they concentrate on holding the device still. One step is detected if the z-axis acceleration drops by at least $\Delta p = -0{,}7ms^{-2}$ within 5 samples as diagrammed in Listing 2. This drop must be within a window $w = 5$ samples, or 166ms. After one step is detected, a timeout $t = 400ms$ is applied to filter movements after a step. Figure 10 shows the z-axis of an accelerometer and two detected steps. The parameters p, t, the lowpass filter l and the step size $s = 0{,}7m$ can be calibrated and are discussed in section 2.4.3. The initial values $\Delta p = -0{,}7ms^{-2}, t = 400ms, l = 0{,}3$ are empiric values which proved as a good baseline. This is also discussed in further detail in section 2.4.3.

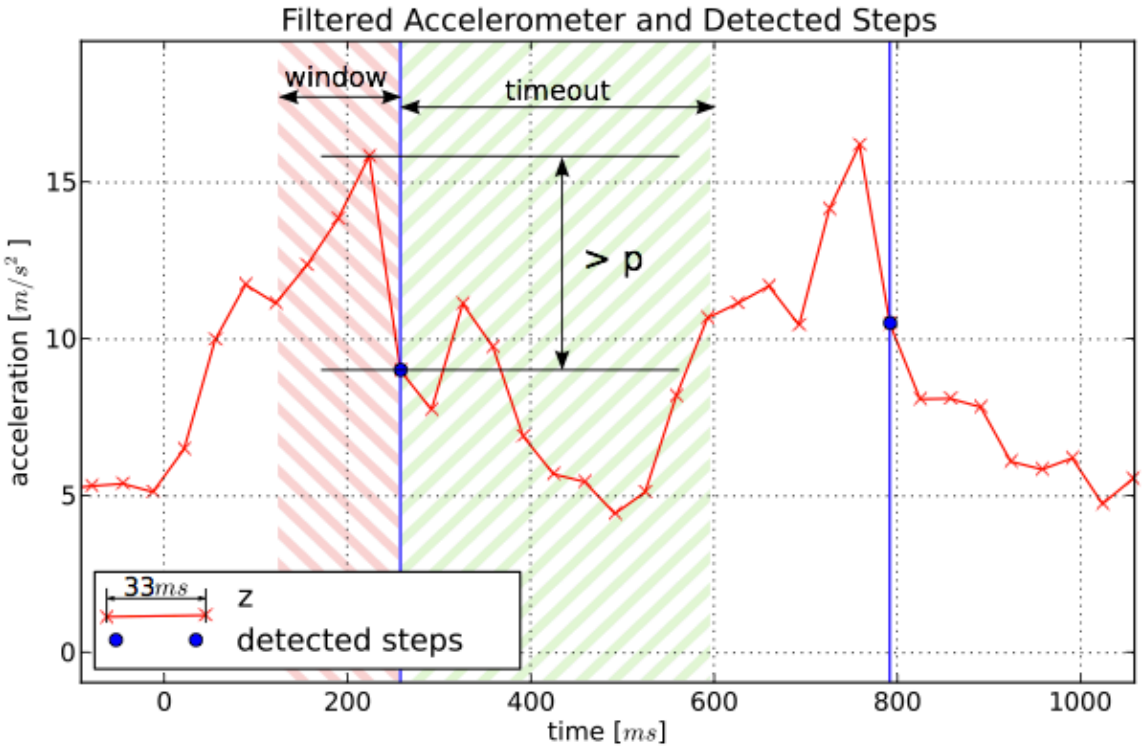

Figure 10: Step Detection algorithm implemented by FootPath [14, p. 2]

```
1  for (int t = 1; t <= WINDOW; t++) {
2      if ((values_history[(vhPointer - 1 - t + vhSize + vhSize) % vhSize] -
   values_history[(vhPointer - 1 + vhSize) % vhSize] > peakSize)) {
3              return true;
4      }
5  }
```

Listing 2: Detection of a step

The algorithm to detect a step is implemented as the `StepDetector` class. This class holds all sensor values, filters the new added values and checks if a step is found. The class is used by `StepDetection`, which reads the compass and accelerometer sensor data and passes it on to the `StepDetector`. Auto Calibration in the `CalibrationActivity` also makes use of the `StepDetector` to process saved data with different lowpass filters l and peak values p.

When a step is detected, a method of the interface `StepTrigger` with the current azimuth of the compass a_c is called by `StepDetection`. The implementing class `StepDetectionProvider` calculates the direction aligned to the magnetic north of the map a_m and adjusts the position with the step size s and grid spacing g as shown in Formula 1 and Figure 11.

$$P_{n+1}(x,y) = P_n(x + \sin(a_c + a_m) * s * g, y + \cos(a_c + a_m) * s * g)$$

Formula 1: Position tracking with step detection

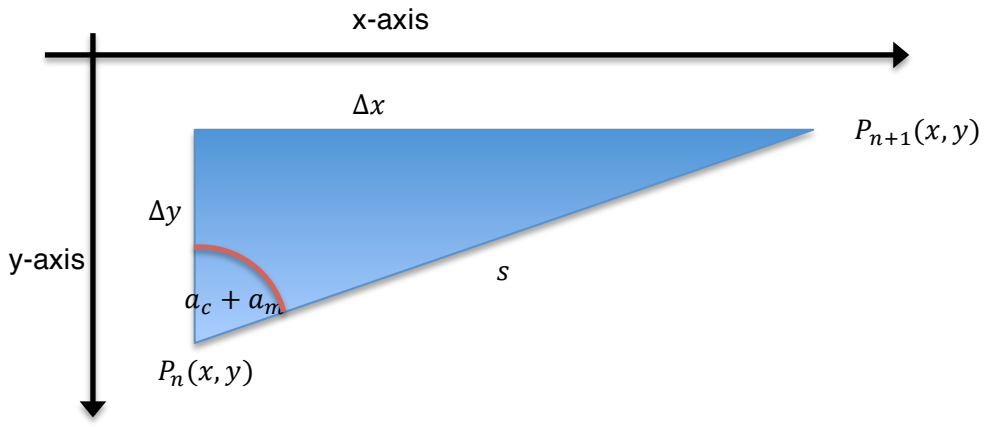

Figure 11: Position change by a detected step

Step Detection is one provider of location information for the `LocationService`. This Service provides methods to get and modify the current position and information about the current local frame, as well as the angle to the relative north (a_m) and the grid spacing (g). `StepDetectionProvider` does not set the absolute position, because it only tracks

movement. However, it reads the current position and adjusts the position as shown in Figure 18. The initial position is defined as the center of the map and can be changed by dragging the user icon around. This movement of the icon is monitored by the `ManualLocationProvider`, which updates the `LocationService`. This architecture can be adopted in future versions to add further location providers, like GPS or trilateration. The UML diagram in Figure 12 shows all relevant classes, which are used to provide location tracking and Step Detection in WiFi Compass, as well as their relations.

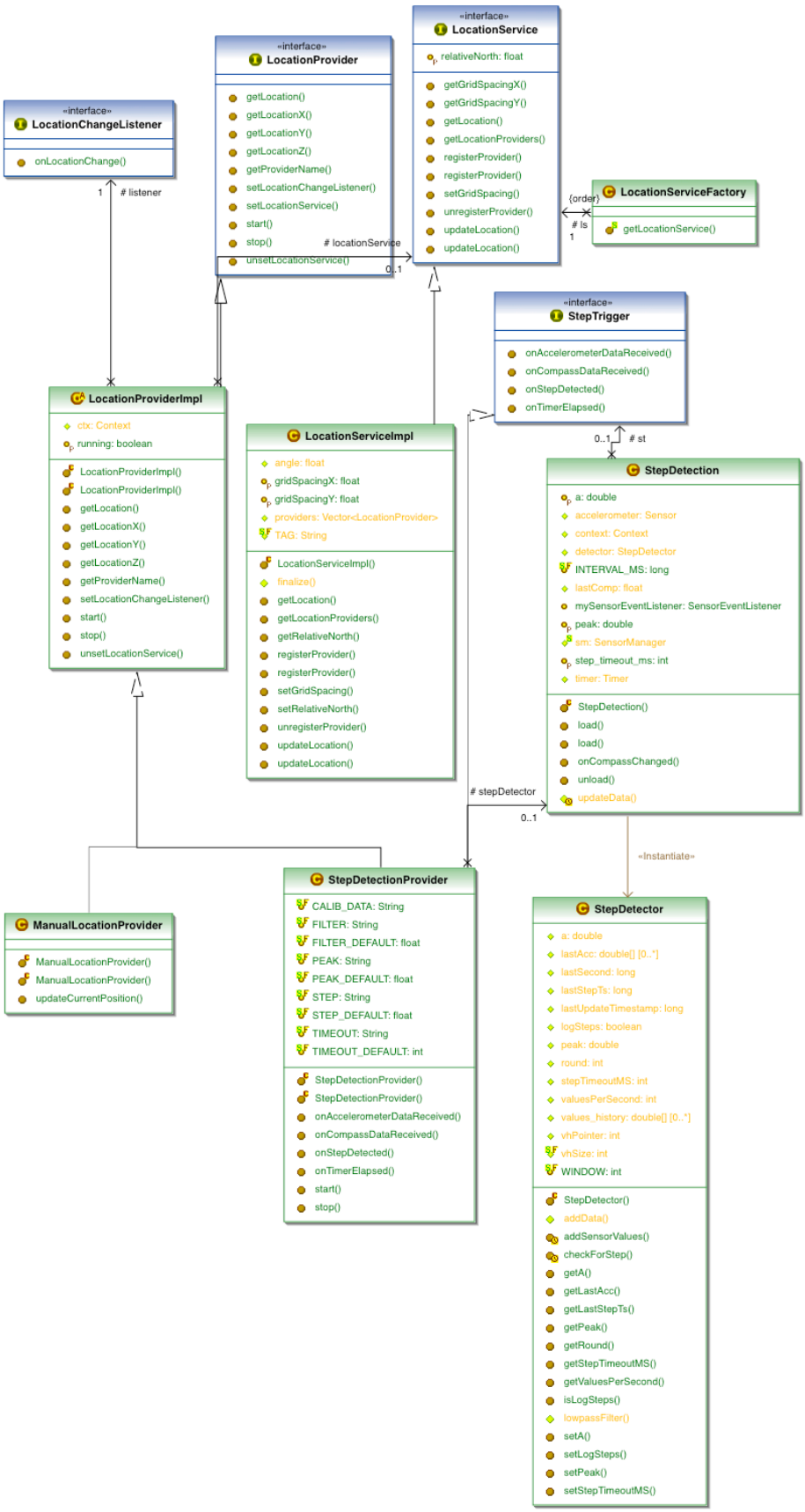

Figure 12: UML diagram of Location Tracking Classes

2.4.3 Parameter Adjustment

Bitsch et al. stated, that the parameters $p = 2\ m/s^2, t = 333\ ms$ are robust against different body heights and walking styles [**14, p. 3**]. In the first tests, the authors noticed that this may be the case, but these parameters may not be robust against different devices. To provide a stable way to define this and not require the user to have knowledge about the correct parameters, the authors decided to create an auto configuration algorithm, which tries to match real steps against the data measured. The calibration process starts with the user walking around and tapping the graph area each time a step is done (this is shown in Figure 14). The timestamps of the steps and all sensor values received are recorded in the database.

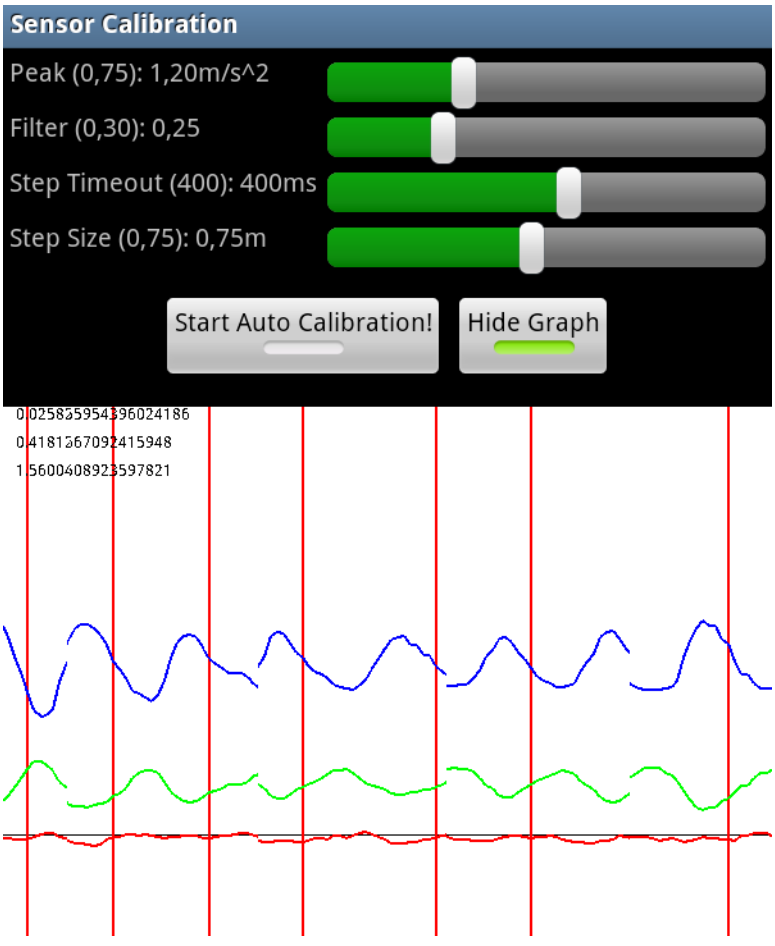

Figure 13: collecting step sensor data

The algorithm searches for the best matching parameter values with $\{p \in \mathbb{R} | 0{,}2 < p < 3; p_{n+1} = p_n + 0{,}1\}, \{l \in \mathbb{R} | 0{,}05 < l < 0{,}8; l_{n+1} = l_n + 0{,}05\}$. The step timeout does not change,

because this would take about ten times longer. Moreover, the parameter is not device dependent. As Bitsch et al. stated, this parameter is robust against user walking styles and body heights [**14, p. 3**]. With each parameter duple, `StepDetector` tries to find steps t_d and matches them against the time stamps t_r, which the user provided by tapping on the graph area. This time stamps can't be exactly the same, because the algorithm searches peaks in negative acceleration. The user taps on the device when they think they are doing the step. One step is not only a short movement, it takes some time where the body moves up and down. To compensate inaccuracy of the timestamps recorded, the step detected by `StepDetector` is matched against a window w_d as shown in Formula 3. The user can adjust the window size in order to narrow the search algorithm if too many results are found. This is shown in Figure 14.

For each parameter duple, a score is calculated to compare the performance of the values. A reward and punishment method is used to measure performance:

- Add one point to the score if a step is detected and a matching user-defined step is found.
- Subtract one point from the score if a user-defined step is not detected.
- Subtract two points from the score if a step is detected and the user defined no step.

For each duple, a weighted percentage score ($P(step)$) is calculated, which describes the success of these parameters. The count of steps not detected n and the count of steps wrongly detected multiplied by two ($f * 2$) is subtracted from the count of correct detected steps c. The resulting value is divided through the step count s. The calculation is shown in Formula 2. The best possible score is be 100%; negative scores are also possible.

$$P(step) = \frac{c - n - f * 2}{s}$$

Formula 2: weighted percentage calculation of a step

The parameter duple with the best score is then used. If multiple solutions with the same score are found, the first one found is used. As a result, parameter duples with lower peak and lower filter values are preferred. In WiFi compass, the results of the algorithm are presented to the user and automatically saved.

$$|t_r - t_d| < w_d/2$$

Formula 3: matching detected against user defined steps

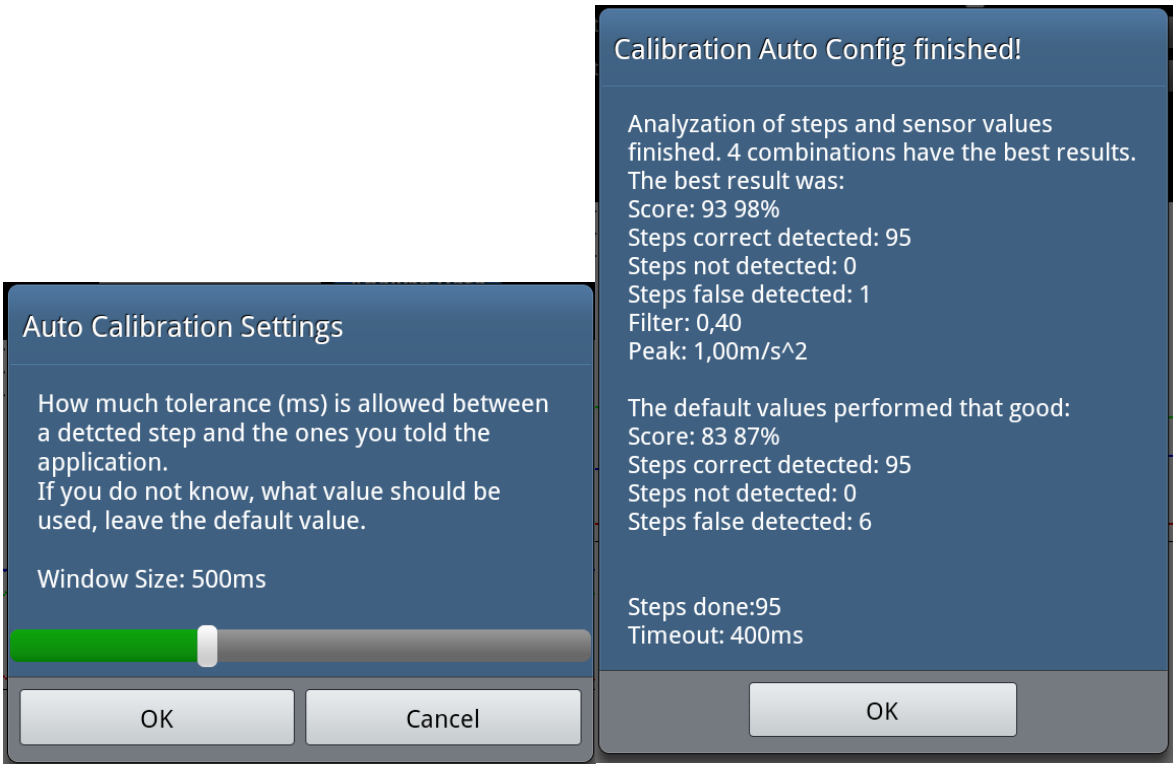

Figure 14: Sensor auto calibration

The auto calibration algorithm has been tested on six different device types by eight different users. The detailed results are documented in appendix B: Sensor Calibration Test Results. Each test run consists of following information:

- Test number.
- Device: The name of the device the test was done with.
- User: Numeric ID of the user, who did the test.
- Steps: Count of steps done during this test.
- Timeout: Step timeout t between two steps.
- Window: Tolerance Window w_d, defining the window in which a step must be detected.
- Score: Best score found.
- Pct: Percentage describing the success of the parameters $P(step) = \frac{c-n-f*2}{s}$
- Found: Count of steps c correctly found by the algorithm with these parameters.
- Not found: Count of steps not detected n by the algorithm.
- False: Count of steps detected by the algorithm, but not declared by the user (f).
- Filter: Lowpass filter value l with the best test score.
- Peak: Minimum drop of acceleration Δp with the best test score.
- Def score: Score of the default parameters.
- Def pct: Percentage describing the success of the default parameters $P(step) = \frac{c-n-f*2}{s}$
- Def found: Count of steps c correctly found by the algorithm with default parameters.
- Def not found: Count of steps not detected n by the algorithm with default parameters.

- Def false: Count of steps the detected by the algorithm with default parameters, but not declared by the user (f).

The test runs 1 to 15 do not include the results of the step detection algorithm with the default parameters, because the values have not been logged. The tests conducted show that the parameters should be adjusted to each device and user before being used. This will provide a very accurate step detection algorithm with a success score of over 90%. The default values proved to detect steps very well, but also detect some false positives.

2.4.4 Conclusion

To benchmark the performance of the step detection algorithm implemented by WiFi compass, real world tests have been conducted with a Samsung Galaxy Tab 7" and a Samsung Galaxy Tab 8.9". These devices were used during the development of the Android application. The parameters had been adjusted to fit the devices and the user on both devices. Section 2.4.3 "Parameter Adjustment" showed that the steps are detected correctly in over 90% at test cases. To benchmark whether the navigation with steps and the use of the compass works in a real world scenario, tests have been conducted.

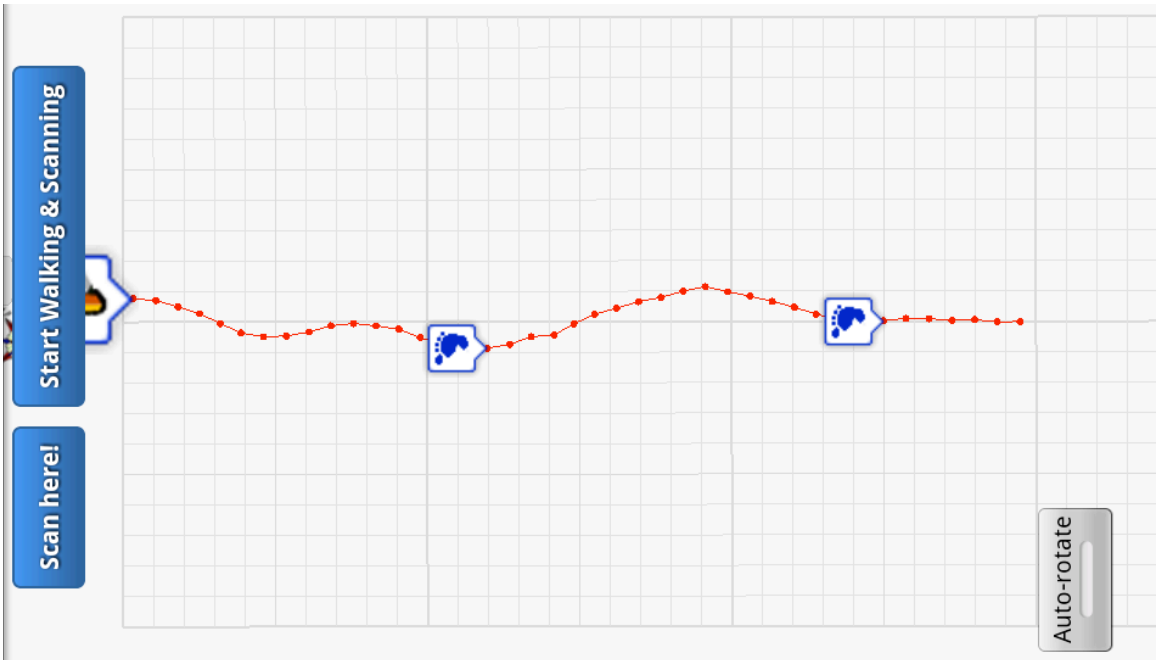

Figure 15: straight walk of 30m in a office building

The first test was performed in the basement of an office building. The target was to measure the deviation of the detected path from the original path. Before the test started, the magnetic north of the map had been calibrated as shown in Figure 7. A walk of about 30 m in a straight line in a corridor in the office building was done as shown in Figure 15. The red line with the red

dots shows the detected steps and their path. The grid on the map equals 1 m x 1 m and is used to help navigating if no map is loaded, like in this case. The path is followed by the step detection with a maximum deviation of 1 m. This is shown in Figure 16. The deviation is a result of disturbances of the magnetic field from electric sources or obstacles, which deflect the magnetic field. This is very common in office buildings and the amount on error depends on the environment. In the specific test case, the compass error did not exceed 45°. The end position after 30 m differs 1 m from the real position resulting in an error value of 3 %. This value proves that an Inertial Navigation System based on accelerometer and magnetic field sensors can be used to navigate along a straight path.

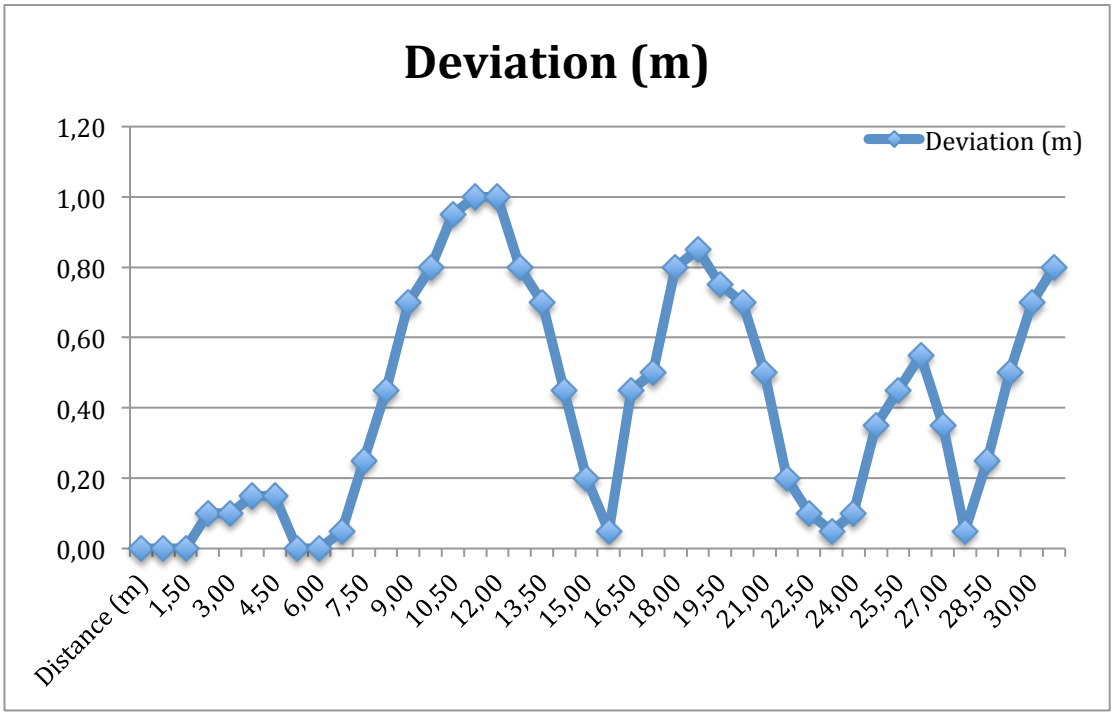

Figure 16: Deviation in a straight walk

The second navigation test was realized in a flat, because at the time the test was conducted, the office building also used in section 3.5 was not available for tests. Figure 17 shows a map of the environment.

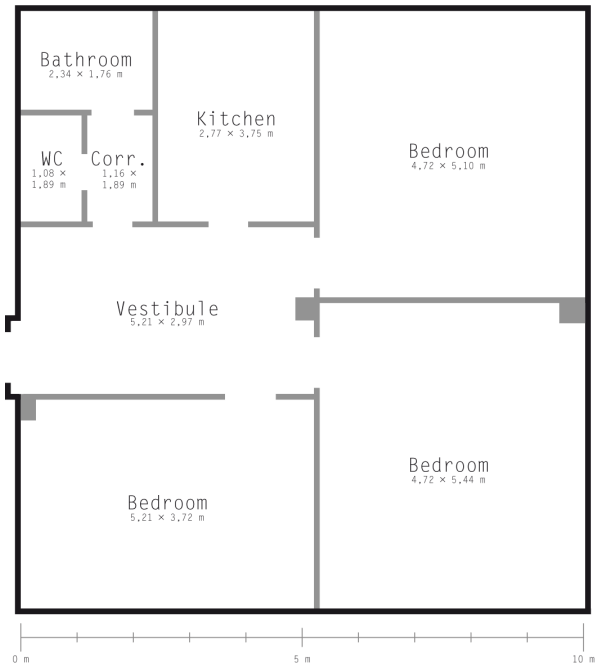

Figure 17: Test Environment for Step Detection Algorithm

A five-minute walk was done in the environment and the tracked steps were logged. Figure 18 shows the steps detected from WiFi Compass as a red line with dots. The test was done with a Samsung Galaxy Tab 8.9". To provide best possible results, the sensors were calibrated with auto calibration and the step length was set to 45 cm, which was estimated with a short walk.

The walk started in the vestibule and went through the three bedrooms and the kitchen. The WiFi scan results were logged and are shown as blue footprints in Figure 18. The real walk path is marked as green line. The author tried to choose a path which is easy to draw on a map afterwards, but there were obstacles in the way, which prevented a rectangular walk through the flat. The tracked path differs from the real walked path, but the deviation does not exceed 1 m. As a result, the WiFi Compass tracking algorithm provides a good position estimation, which can be manually corrected at any time by the user.

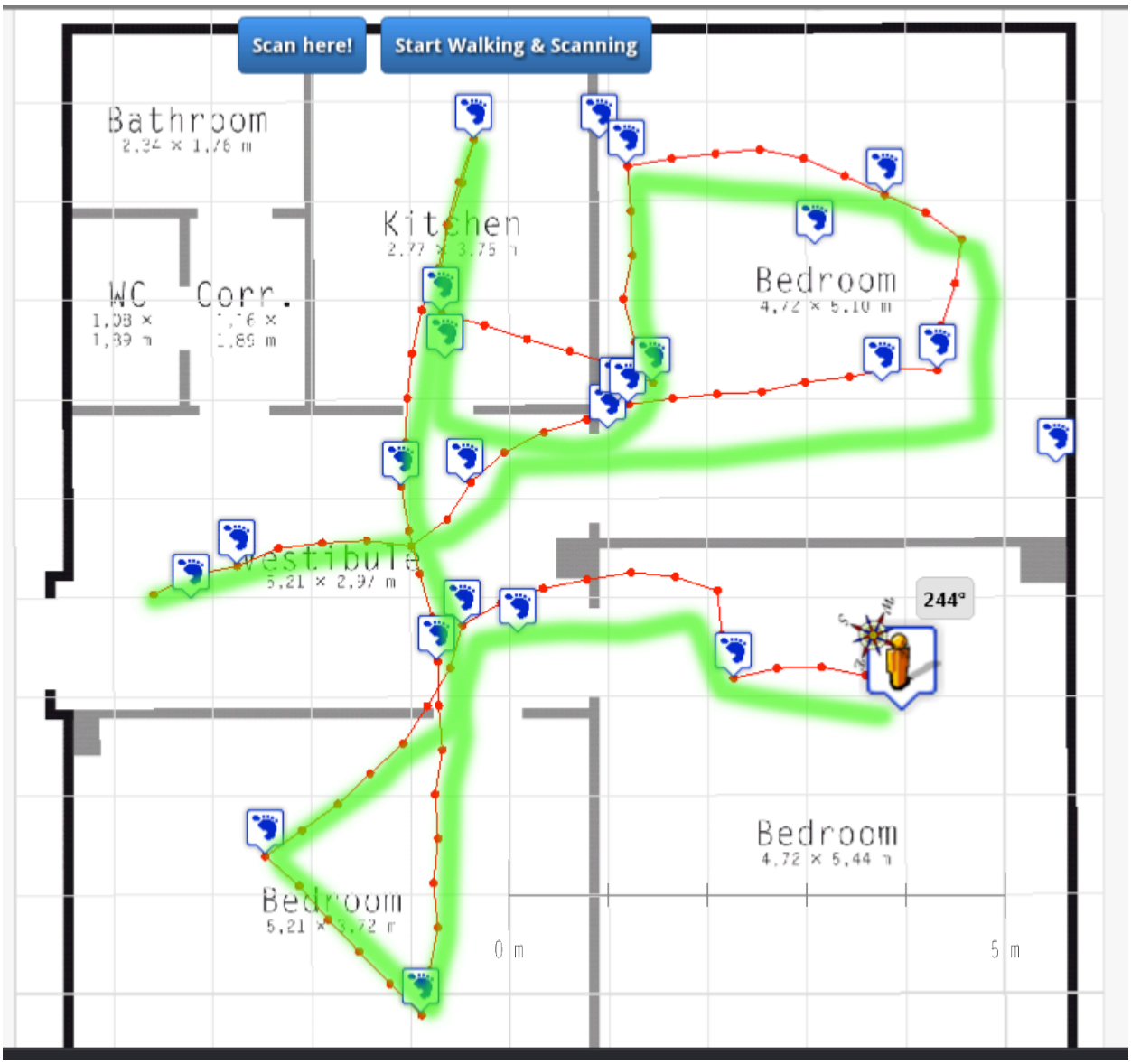

Figure 18: Real walk path compared to estimated path

3 Access Point Trilateration

This chapter provides detailed information about possible algorithms to calculate the position of the access point once the measurement data has been collected. First, an overview of related work is given. Then, various algorithms are selected that match the requirements of WiFi Compass. These requirements are

- that no extra hardware but a WiFi-capable device is needed,
- that the algorithm runs on low-performance mobile devices, and
- that the algorithm does not need much pre-configuration.

The selected algorithms are then described and tested in a real-world scenario.

3.1 Related Work

Schuhmann et al. [**23**] describe the Weighted Centroid algorithm in their work. They use the algorithm to locate users in the building; this requires the positions of the access points to be known. However, the algorithm can be adapted to locate access points when measuring points are given. They also describe how the algorithm can be improved to give more accurate results. The algorithm is also described and tested in this work.

Subramanian et al. [**24**] describe an approach that uses a “steerable beam directional antenna mounted on a moving vehicle to localize roadside WiFi access points (APs), located outdoors or inside buildings.” “The idea is to estimate the angle of arrival of frames transmitted from the AP using signal strength information on different directional beams of the antenna – as the beam continuously rotates while the vehicle is moving. This information together with the GPS locations of the vehicle are used in a triangulation approach to localize the APs.” This approach requires extra hardware (the directional antenna) and will therefore not be used in WiFi Compass.

Hansson et al. [**1**] describe the Advanced Trilateration approach in their work. The algorithm tries to estimate the distance of the access point from each measuring point. It does so for every measuring point. It then takes these values and tries to estimate the position where the access point is most likely located. It requires the knowledge of the transmitting power of each access point at 1 m and the average attenuation of the environment. In this work, the algorithm is described, optimized and tested.

Han et al. [**25**] describe a novel approach in their work. It is called the Local Signal Strength Gradient algorithm. This algorithm takes every measuring point and tries to estimate the direction towards which the access point is most likely located. It does so by taking the surrounding measuring points into account and finding out where the RSSI value increases most. This directional information of every measuring point is then processed to give the location where the access point is most likely located. As this approach does not required any

knowledge about the transmission power of the access points or the attenuation of the environment, it is described in this work and tested in a real-world scenario.

To sum up, the three candidate algorithms are

- the Weighted Centroid algorithm,
- the Advanced Trilateration algorithm, and
- the Local Signal Strength Gradient algorithm.

In the last section of this chapter, all three algorithms will be compared in a real-world scenario.

3.2 The Weighted Centroid Algorithm

The Weighted Centroid algorithm is the simplest of all mechanisms presented in this work. It works by calculating the geometric centroid of all the measuring points weighting each point's x and y coordinates by the signal strength measured [**25, p. 2**].

3.2.1 The Algorithm

In order to give higher RSSI values proportionally higher priority than lower values, each RSSI value in the measurement data set is processed the following way ($RSSIP$ means prioritized RSSI):

$$RSSIP_i = (P_{ref} \cdot 10^{\frac{RSSI_i}{20}})^g$$

Formula 4: Prioritization of the RSSI values in Weighted Centroid [23, p. 7]

where P_{ref} is the reference power, in this case 1 mW. With this formula, the RSSI value [dBm] is converted back to milliwatts [mW] and raised to the g-th power, where g is a constant that defines how big the weight difference between the highest and the lowest value should be. The higher g is, the less importance is given to very low RSSI values, or, in other words, measuring data sets taken far away from the access point have little importance at high g values. For an ideal selection of a g value see section 3.5.1.

Given a set of measuring points with the data x, y and $RSSIP$, the formula used for the calculation of the access point position looks as follows:

$$Pos_{AP} = \left(\sum_{i=1}^{N} w_i x_i , \sum_{i=1}^{N} w_i y_i \right)$$

Formula 5: The Weighted Centroid algorithm [25, p. 2]

where w_i is the weight by which each measuring data set is prioritized. The weight w at each point is given by

$$w_i = \frac{RSSIP_i}{\sum RSSIP}$$

Formula 6: Calculation of the weight factor in Weighted Centroid

In [**25, p. 2**], each RSSI is weighted by the signal-to-noise ratio (SNR). This value cannot be used in this case, as the SNR data cannot be retrieved by the Android operating system (see Table 1 for the fields returned by `WifiManager`[4] in Android).

Fields [of the `ScanResult` class in the Android Operating System]		
public String	**BSSID**	The mac address of the access point interface for the specific SSID.
public String	**SSID**	The network name.
public String	**capabilities**	Describes the authentication, key management, and encryption schemes supported by the access point.
public int	**frequency**	The frequency in MHz of the channel over which the client is communicating with the access point.
public int	**level**	The detected signal level in dBm.

Table 1: Member variables of the `ScanResult` class in the Android Operating System [26]

Therefore, the signal strength RSSI is used instead of the signal-to-noise ratio to weight the measuring points (see Formula 4).

3.2.2 MATLAB Implementation

Listing 3 shows a MATLAB implementation of the algorithm described above. A constant value of 1.3 is used for g because this has proven to be a good value in indoor environments (this is further discussed in section 3.2.3).

```
1  function [posX, posY] = weighted_centroid(x, y, rssi)
2      g = 1.3;
3      rssi = (10.^(rssi/20)).^g;
4      x = x .* rssi ./ sum(rssi);
5      y = y .* rssi ./ sum(rssi);
6
7      posX = sum(x);
8      posY = sum(y);
```

Listing 3: MATLAB implementation of the Weighted Centroid algorithm

[4] See http://developer.android.com/reference/android/net/wifi/WifiManager.html

The parameters x, y and $rssi$ are $n \times 1$ matrices. Operators with a preceding dot („.“) mean an element-by-element operation. Thus, if the variable right of the operator is a scalar, each element in the matrix left of the operator is processed with the scalar.

3.2.3 Challenges

The algorithm described above is pretty simple and easy to implement. In addition, it proved to be very fast, even on mobile devices (see section 3.5.2 for details). There are, however, two major challenges when it comes to using the algorithm in practice. These challenges are

- an appropriate selection of a constant g, and
- the more or less even distribution of measuring points around the access point.

Both things must be given in order for the algorithm to give accurate results.

The constant g

As described in section 3.2.1, the constant g defines how much importance is given to measuring points with low signal strength values. Thus, a high value g means that more importance is given to measuring points that are close to the access point. In [**23, p. 8**], values between 0 and 5 are used for g. A value of **1.3** for g has proven to give accurate results in the test scenario of section 3.5.

Distribution of the measuring points

The Weighted Centroid algorithm only gives accurate results when the measuring points are more or less evenly distributed *around* the access points in consideration. This is because the geometric centroid of a set of points can only be within the area spanned by the points. In other words, the x value of the centroid can never be lower than the lowest x of all points. For details about how the distribution of the measuring points affects the accuracy of the positioning algorithms see section 3.5.3.

3.3 The Advanced Trilateration Algorithm

A method described in [**1, p. 62 et sqq.**] uses advanced trilateration to locate access points. In short, each measuring data set is taken, the distance to the access point is estimated and a circle is drawn around the measuring point with this distance as the radius. Then, a Gaussian probability distribution is used to distribute probability values inside and outside this circle, the points on the circle itself being the locations with the highest probability that the access point is located here. This is done for each and every measuring data set captured. Then, the area is rasterized. The probability values are multiplied on each point on the resulting grid, and the point with the highest probability is selected. This is where the access point is most likely located.

3.3.1 The Algorithm

In order for the algorithm to work correctly, two constants need to be defined [**1, p. 64**]:

- C, which which describes the expected RSSI at a distance of one meter from the access point, and
- n, which describes the average attenuation of the environment.

First of all, the entire area is rasterized, e.g. with a grid distance of 0.5 meters. Optionally, the area can be expanded in order to be able to calculate the positions of access points that are located outside that area. Then all those points on the map are looped through.

For each and every point on the map $p_{i,j}$, we loop through all the measuring points previously recorded. Each measuring point $m(k)$ consists of the x-coordinate $m(k)_x$, the y-coordinate $m(k)_y$ and the RSSI $m(k)_R$ measured at the point. The distance $d(k)$ between the measuring point and the access point is given by

$$d(k) = 10^{(C - m(k)_R)/10n}$$

Formula 7: The distance between measuring point and access point [1, p. 71]

The perimeter of the circle with radius $d(k)$ around the measuring point $(m_x(k), m_y(k))$ is denoted $C(k)$. This is the circle on which the access point is most likely located. For each measuring point $(m_x(k), m_y(k))$, the distance $\Delta d_{x,y}(k)$ of the current point on the map (x, y) to the circle $C(k)$ is given by

$$\Delta d_{x,y}(k) = \sqrt{(m_x(k) - x)^2 + \left(m_y(k) - y\right)^2} - d(k)$$

Formula 8: The distance between a given point on the map and the circle C [1, p. 71]

This distance is an indirect indicator for the probability that the access point is located on the current point on the map $p(k)_{i,j}$. In order to calculate the probability $p(k)_{i,j}$ that the access point is located at the current point (i, j) on the map, the Gaussian probability distribution is used:

$$p(k)_{i,j} = \frac{1}{\sqrt{2\pi\sigma(d)}} \cdot e^{\frac{-\Delta d_{i,j}(k)^2}{2\sigma(d)^2}}$$

Formula 9: Probability that the access point is located on a given point on the map [1, p. 71]

In Figure 19, the probability distribution for the location of the access point seen from one single measurement point is shown. The measurement point is at (0, 0). It can be seen that the highest probability is at the estimated distance of 10 m (the dark red part of the surface).

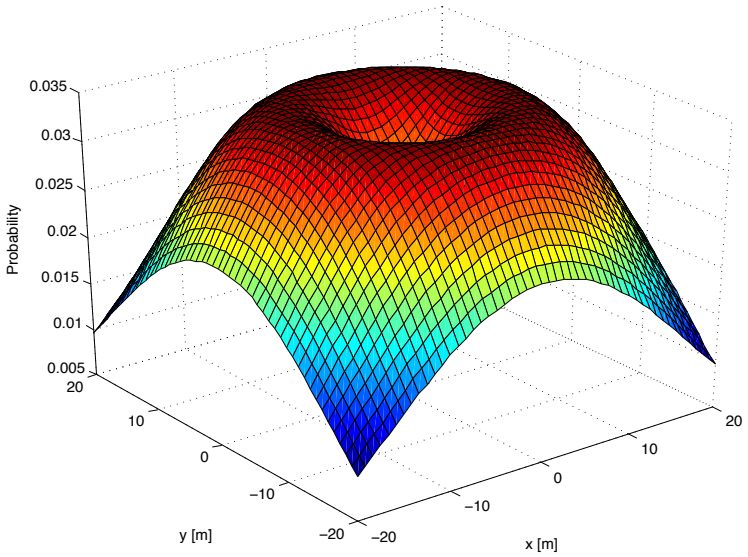

Figure 19: Probability distribution on a 2D map seen from one measuring point with an estimated access point distance of 10 m [1, p. 70]

$\sigma(d)$ is the standard deviation for the Gaussian distribution. This standard deviation is not constant but depends on the measured RSSI value. This is due to the fact that an error in the measured RSSI value has a much greater impact for lower signal strengths than it has for higher signal strengths. For example, the distance between a -70 dBm measurement and a -71 dBm measurement is roughly 0.8 meters. However, the distance between a measurement of -90 dBm and -91 dBm is about 2.7 meters [**1, p. 70 et seqq.**]. The standard deviation is calculated by

$$\sigma(d) = 0.64d + 5$$

Formula 10: The standard deviation [1, p. 71]

The relationship between the estimated distance and the probability distribution based on the RSSI values can be seen in Figure 20.

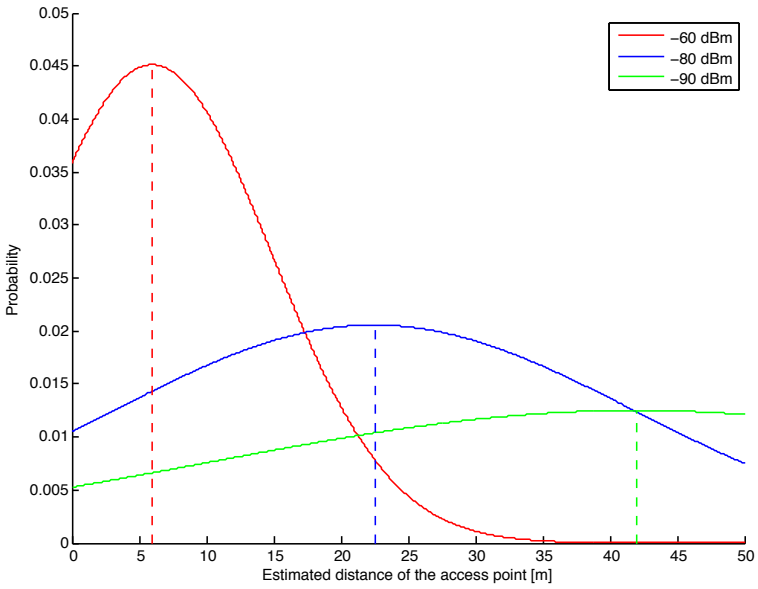

Figure 20: The Gaussian distribution for three different RSSI measurements [1, p. 71]

The final probability for each position on the map, taking into account *all* measuring points, can be calculated by

$$\forall g_{i,j} \in G: g_{i.j} = \sum_{k=1}^{N} p_{i.j}(k)$$

Formula 11: The probability for each position on the map [1, p. 72]

The point on the map with the highest probability is the estimated position of the access point. Listing 4 shows a MATLAB implementation of the probability function for a given point x on the map (in this case, x is a two-value array which holds both the x and the y value). Note that a change of sign is done with the probability value (see line 42 of Listing 4) because the optimization tool described in section 3.3.3 is designed only for finding local minima instead of local maxima.

```
1  % The probability function. Returns an inverted probability for the AP
2  % to be located at x (x contains x and y coordinates)
3  function p = prob(x)
4
5      % Gives us access to the measuring data
6      global xdata
7      global ydata
8      global zdata
9      global C
10     global n
11
12     % We start with probability 0 and then add up the probability values
13     % for each measuring point
```

```
14      p = 0;
15
16      for i=1:size(xdata)
17          % Get the current measuring point data
18          currx = xdata(i);
19          curry = ydata(i);
20          currz = zdata(i);
21
22          % Calculate estimated circle circumference d
23          d = 10^((C - currz)/(10 * n));
24
25          % Calculate the distance to the circle
26          deltaD = sqrt((currx - x(1))^2 + (curry - x(2))^2) - d;
27
28          % The standard deviation for Gaussian distribution depends
29          % on the estimated distance
30
31          sigma = 0.64 * d + 5;
32          % Calculate the probability
33          pTemp = 1 / sqrt(2 * pi * sigma^2) *...
34                  exp(-(deltaD^2) / (2 * sigma^2));
35
36          % Add the probability value of the current measuring point
37          p = p + pTemp;
38      end
39
40      % We have to invert the value because the optimization tool can only
41      % find local minima, not maxima
42      p = -p;
43  end
```

Listing 4: A MATLAB implementation of the probability function in Advanced Trilateration

A complete MATLAB implementation of the algorithm described above can be found at 6.1.1.

3.3.2 Challenges

The Advanced Trilateration algorithm is way more complex than the Weighted Centroid algorithm. Challenges are

- the appropriate selection of a constants C for each access point,
- the appropriate selection of a constant n for the entire environment, and
- the performance of the algorithm on slow devices.

As C denotes the signal strength of an access point at a distance of 1 m, using a constant value for C for a number of different access points may not give accurate results. Moreover, the value

is impossible to estimate in unknown environments. The same applies for n. However, n is easier to estimate because

1. it is, to a greater or lesser extent, constant in the entire environment and
2. it can be roughly estimated by the thickness and number of walls.

In the next section (3.3.3), the algorithm is optimized in order to face these challenges.

3.3.3 Optimization Attempts

Both the selection of appropriate values C and n and the performance are problems when using the Advanced Trilateration algorithm. In this section, the Algorithm is optimized in order to give better results.

The constants C and n

In order to calculate the distance between the measuring points and the access point, Formula 12 can be used. C, n and the signal strength in dBm (R) must be given.

$$d = 10^{(C-R)/10n}$$

Formula 12: The distance between measuring point and access point [1, p. 71]

To calculate the signal strength at a given point (x, y) for given C and n values, the formula can be transformed as depicted in Formula 13.

$$R = C - 5 \cdot n \cdot log_{10}((x_{est} - x)^2 + (y_{est} - y)^2)$$

Formula 13: Calculation of expected RSSI for given coordinates, C, n, and estimated AP position

(x_{est}, y_{est}) are the coordinates of the estimated access point position. The position must be roughly estimated because otherwise the best-fitting C and n values cannot be calculated. When the values x, y and R for given C, n and (x_{est}, y_{est}) are drawn into a 3-dimensional coordinate system, the figure looks as Figure 21.

In this algorithm, the rough estimation of the access point location is calculated using the Weighted Centroid algorithm (for details about the algorithm see section 3.2).

For every measurement point given, a function like in Formula 13 is formulated. Now there are usually more equations than unknowns, so the equation system is solved using the method of least squares (for details about how this method works see section 3.7). Figure 21 shows an example of such a calculation. The rainbow-colored plane is the best-fitting plane for the given set of measuring points (the magenta-colored points). In this example, the access point is estimated to have a signal strength of -36 dBm at a distance of 1 m. The environment is estimated to have an average attenuation of 1.37.

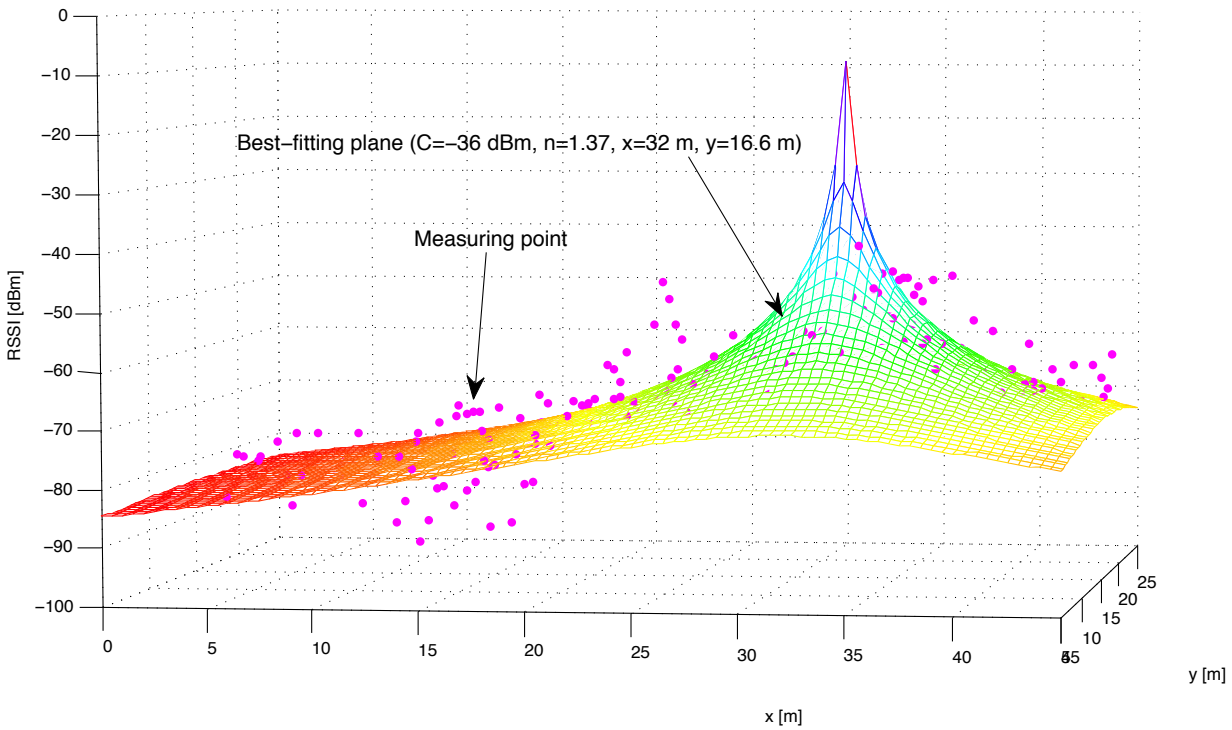

Figure 21: Calculation of best-fitting C and n

In the test environment (see section 3.5), it has been found that the results using this algorithm are not accurate. This is because the estimation of the access point position is often inaccurate. So when the estimated access point position is a few meters off the real position, C and n values are very likely to be estimated too low.

However, and experiment has shown that constantly adding a value of 14 to C and a value of 3.2 to n *gives way more accurate results than the non-optimized Advanced Trilateration algorithm*: **The average error decreases from 3.87 meters to 2 meters, the maximum error from 21.97 meters to 4.7 meters, and the median error from 3.01 meters to 1.62 meters** (see section 3.5.2 for details).

Speed

When it comes to implementing the Advanced Trilateration algorithm on mobile devices, speed is an important aspect. The algorithm is very slow because it needs to calculate the probability for the access point to be calculated on every point of the map, dividing it into, for example, 25 cm slices. For an environment of e.g. 45 x 25 meters, 150 measuring points and 20 access points this would mean **54 million** runs through the MATLAB code of Listing 5.

```
1  % Calculate estimated circle circumference d
2  d = 10^((C - currz)/(10 * n));
3
4  % Calculate the distance to the circle
```

```
5  deltaD = sqrt((currx - x(1))^2 + (curry - x(2))^2) - d;
6
7  % The standard deviation for Gaussian distribution depends
8  % on the estimated distance
9
10 sigma = 0.64 * d + 5;
11 % Calculate the probability
12 pTemp = 1 / sqrt(2 * pi * sigma^2) *...
13         exp(-(deltaD^2) / (2 * sigma^2));
```

Listing 5: CPU-intensive part of the MATLAB implementation of the Advanced Trilateration algorithm

The Advanced Trilateration algorithm tries to find the x and y coordinates of the point where the probability p for the access point to be located is the highest. An exhaustive search of the entire area is cost-intensive.

There are, however, algorithms called *optimization tools*, which can quickly find a local minimum or maximum of any given function very quickly. In order for these tools to work, there must not be any local minima between the initial guess and the global minimum. This further implies that the initial guess must already be close to the real minimum.

If this condition is met, the optimization tool finds the global minimum a lot faster than an exhaustive search would do. Fortunately, the probability distribution of the access point location in the Advanced Trilateration algorithm proves to be quite continuous, or in other words, there are usually not many local minima between the initial guess and the global minimum (see Figure 22 for an example probability distribution). As an initial guess, both the measuring point with the highest signal strength or the result of the Weighted Centroid algorithm can be used. In the case of WiFi Compass, the Weighted Centroid calculation serves as the initial guess for the optimization tool.

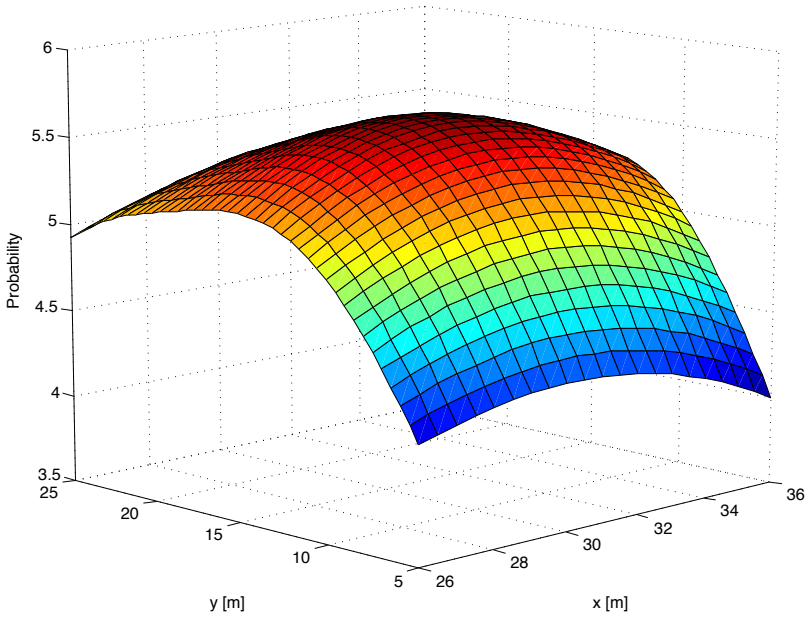

Figure 22: The calculated probability distribution for an access point location using several measuring points

In the MATLAB implementation of the Advanced Trilateration algorithm, the probability function looks as in Listing 4 (note that the probability is multiplied with -1 because the `fminfunc` optimization tool can only find minima in functions.). This is the function where the result p needs to be minimized. In MATLAB, this works as depicted in Listing 6.

```
1 % Use the optimization tool 'fminunc' to search the local minimum
2 % starting from the Weighted Centroid estimation
3 x0 = [wcX, wcY];
4 options = optimset('Display', 'Off', 'LargeScale','Off');
5 [xres, fval] = fminunc(@prob, x0, options);
```

Listing 6: Usage of the `fminunc` optimization tool in MATLAB

`x0` is the variable that holds the initial guess for the coordinates of the highest probability. In the last line, `fminunc` is called with the function, the initial guess and some options as parameters. The variable `xres` is an array with two entries: The x and y coordinates of the local minimum. `fval` holds the value that the function returns at that point (in this case, the probability).

This way, the algorithm is about **190 times faster** than without the optimization. However, the average error increases about 20 cm. This is because in some occasions, the algorithm finds local minima, which are not equal to the global minimum of the entire area in consideration.

3.4 The Local Signal Strength Gradient Algorithm

The Local Signal Strength Gradient (LSSG) algorithm is a novel approach that localizes access points using directional information derived from local signal strength variations [**25, p. 1**]. In

short, the algorithm loops through all the measuring points, takes all the surrounding measuring points within a certain window size with the current measuring point as the center, and estimates the direction towards which the access point is located. In order to achieve this, it takes the RSSI values of the points and draws the “arrow” towards the direction where the RSSI value increases most. This is done for each measuring point. Once this is done, it takes all the arrows and calculates the position with the least squared angle error or, in other words, towards which all the arrows point. This is where the access point is most likely located.

3.4.1 The Algorithm

The algorithm consists of two major steps [**25, p. 4**]. Those steps are

1) the calculation of the “arrows”, or rather the direction towards which the access point is estimated to be located, seen from each measuring point, and
2) the calculation of the point that minimizes the sum-squared angular error of all arrows previously calculated.

The arrows

First of all, the algorithm loops through all the measuring points $(m_x(k), m_y(k))$ and tries to estimate the direction towards which the access point is most likely located [**25, p. 4**]. To achieve this, the x and y values of all surrounding points within a given window size w (e.g., 3 meters) are taken and put into an $n \times 3$ matrix A, where the third column is filled with ones. See Formula 14 for an example.

$$A = \begin{pmatrix} 4 & 6 & 1 \\ 2 & 6 & 1 \\ 0 & 6 & 1 \\ 4 & 4 & 1 \\ 3 & 8 & 1 \end{pmatrix}$$

Formula 14: Example for matrix A

The matrix $RSSI$ represents all corresponding RSSI values (see Formula 15).

$$RSSI = \begin{pmatrix} -52.1600 \\ -51.0206 \\ -50.5630 \\ -50.0515 \\ -52.1600 \end{pmatrix}$$

Formula 15: Example for matrix $RSSI$

In this example, the first measuring point in this case is: x: 4 m, y: 6 m, $RSSI$ at that point: -52,16 dBm. Now, this data is mapped into a 3-dimensional coordinate system, where the $RSSI$ value represents the z coordinate. An example is shown in Figure 23.

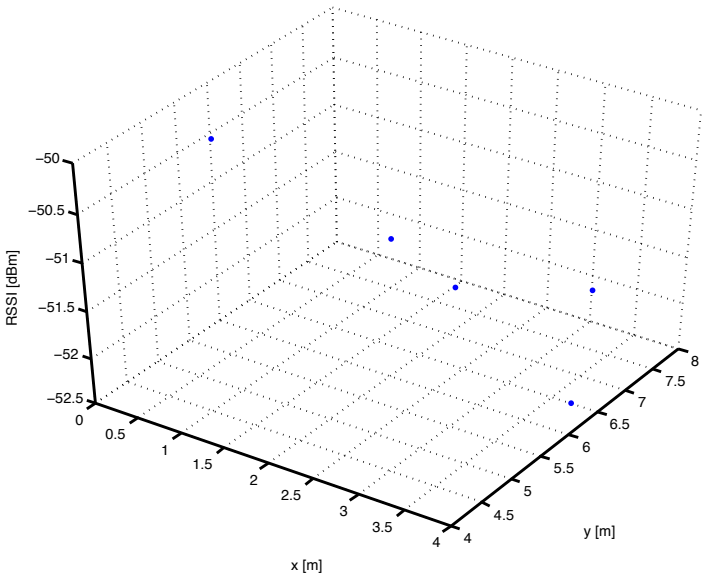

Figure 23: The x, y and RSSI values mapped into a 3-dimensional coordinate system

Now, using the least squares method by dividing the two matrices A and $RSSI$, a plane C is fit into these points (see Formula 16) [**25, p. 4**]. In other words, the plane that minimizes the sum-squared error (the sum of the squared distances of the points from the plane) is calculated. For details about the method of least squares see section 3.7.

$$C = {}^{A}/_{RSSI}$$

Formula 16: Calculation of the best-fitting plane C

The result of the calculation in this example is shown in Formula 17.

$$C = \begin{pmatrix} -0.2746 \\ -0.5958 \\ -46.9024 \end{pmatrix}$$

Formula 17: Example result for matrix C

This result defines the best-fit plane defined like in Formula 18.

$$z = -0.2746 \cdot x - 0.5958 \cdot y - 46.9024$$

Formula 18: Example result for best-fitting plane

The result looks as in Figure 24 (the gray plane equals the plane defined in Formula 18):

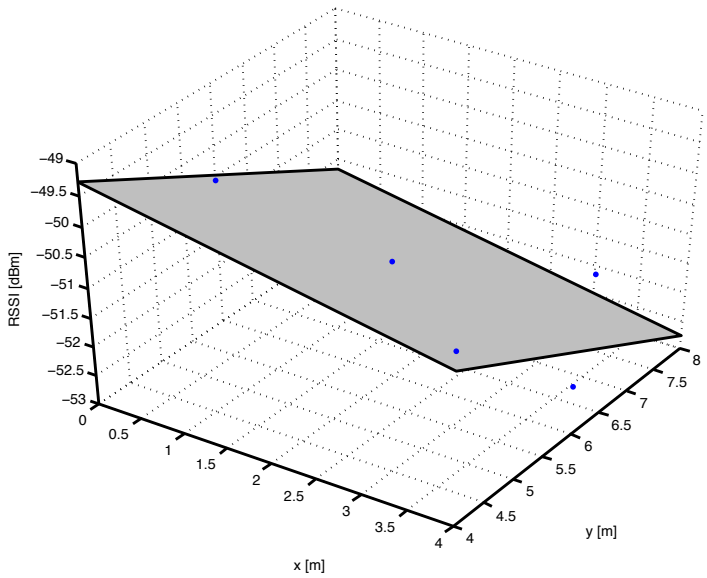

Figure 24: The best-fitting plane for x, y and RSSI

Now, the gradient at the center point (in this case the center point is located at $(2, 6)$) is taken and projected onto the x-y-plane. This is simply done by taking the x and y coordinates of the center point as the starting point of the arrow and the x and y factors of the best-fitting plane as the end point of the arrow.

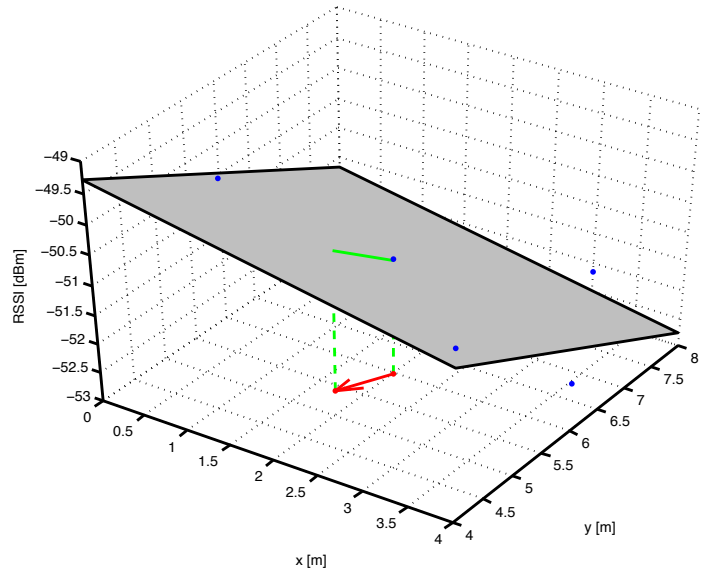

Figure 25: The calculation of the “arrow” [25, p. 3]

In Figure 25, the red arrow at the bottom represents the calculated arrow. It points towards the direction where the access point is most likely located, seen from the current center point.

Weighting measurements by their signal strength

During this work, it has been found out that weighting each measuring point by its signal strength (just like the Weighted Centroid algorithm described in section 3.2.1) improves the accuracy of the algorithm. Formula 19 shows how the signal strength values are processed.

$$RSSIP_i = (P_{ref} \cdot 10^{\frac{RSSI_i}{20}})^g$$

Formula 19: Prioritization of the RSSI values in LSSG

The weight is then calculated like in Formula 20. The result w_i is then multiplied with the squared error of the given point on the map (see Formula 24 later in this section).

$$w_i = \frac{RSSIP_i}{\sum RSSIP}$$

Formula 20: Calculation of the weight factor in LSSG

However, in the case of the Local Signal Strength algorithm, the g value is *negative*. This means that measuring points taken further away from the access points are given *higher* importance. This can be explained by the fact that the closer the measuring points are to the access point, the higher local variations in the environment attenuation are. This seems to increase the error of close “arrows”. A value of -0.4 for g has been found optimal for the test environment (for details see section 3.5.1).

The estimated access point location

Now that the estimated direction of the access point seen from each measuring point is known, the approximated position of the access point can be calculated [**25, p. 4**]. As with the Advanced Trilateration algorithm (see chapter 3.3), the map is rasterized (e.g., with a grid distance of 0.5 meters) and optionally expanded. All these points on the map are then looped through.

At each of those points $p(i)$ on the map, we go through all the arrows a previously calculated. Each arrow consists of the measuring point position $a_{px}(k), a_{py}(k)$ and the direction of the arrow $a_{ax}(k), a_{ay}(k)$. For each of those arrows and the given point, the sum of the squared angular error is computed.

Therefore, both the angle of the current arrow $\alpha_a(k)$ and the angle of the point seen from the arrow $\alpha_{a(k)p_{i,j}}$ need to be calculated (see Formula 21 and Formula 22).

$$\alpha_{a(k)} = atan2(a_{ay}(k), a_{ax}(k))$$

Formula 21: The angle of the current arrow

$$\alpha_{a(k)p(i)} = atan2(p_y(i) - a_{py}(k), p_x(i) - a_{px}(k))$$

Formula 22: The angle of the point seen from the arrow

Given these two angles, we can now calculate the angle difference, or rather the angular error $\Delta\alpha_{a(k)p(i)}$ of the current point on the map seen from the current arrow (see Formula 23).

$$\Delta\alpha_{a(k)p(i)} = |\alpha_{a(k)} - \alpha_{a(k)p(i)}|$$

Formula 23: The calculation of the angular error

Note that in order for the angular error to be correct, it needs to be wrapped to values between 0 and $+\pi$. This is because the angular error can theoretically be any number, so angular errors of 370 ° are theoretically possible. First, the resulting angle needs to be wrapped to values between 0 and 2π. Then, if the angle is greater than π, the angle needs to be subtracted from 2π ($\Delta\alpha = 2\pi - \Delta\alpha$). This gives 15 ° for -15 °, 165 ° for 195 °, 15 ° for 375 ° etc.

Once the angular errors of all arrows are calculated for a given point $p(i)$, these errors are squared, multiplied with the weight w_i (see Formula 20) and summed up like in Formula 24.

$$E(p(i)) = \sum_{k=1}^{N} (\Delta\alpha_{a(k)p(i)})^2 \cdot w_i$$

Formula 24: The sum-squared angular error [25, p. 4]

The access point is now most likely to be located at the point $p(i)$ where the sum-squared angular error E is minimum.

3.4.2 Challenges

When it comes to using the algorithm in practice, the two major challenges are

- the selection of an appropriate parameter g, and
- the performance of the algorithm, especially on slow mobile devices.

In the following sections, an optimization attempt is done concerning these two challenges.

The parameter g

A test was conducted in order to find out the optimal g value for the test environment (for details about the g parameter see section 3.4.1). The survey showed that, at least for the test environment, **-0.4 is a good value**. The survey can be found in section 3.5.1.

Performance

For the Advanced Trilateration algorithm, an optimization tool can be used to dramatically increase performance (see section 3.3.3). This works because the probability distribution is usually more or less continuous. So if the initial guess is not too far away from the global minimum, the local minimum found by the optimization tool is very likely to be equal to the global minimum.

However, this is not possible with the Local Signal Strength algorithm. The problem here is that the probability distribution can be very discontinuous, or in other words, there are lots of peaks and holes in the probability plane. Figure 26 shows a sample probability distribution for an access point with relatively few measuring points. Note that the probability values are inverted. This is because the optimization was implemented in MATLAB, and the `fminunc` optimization tool can only find local minima (not maxima).

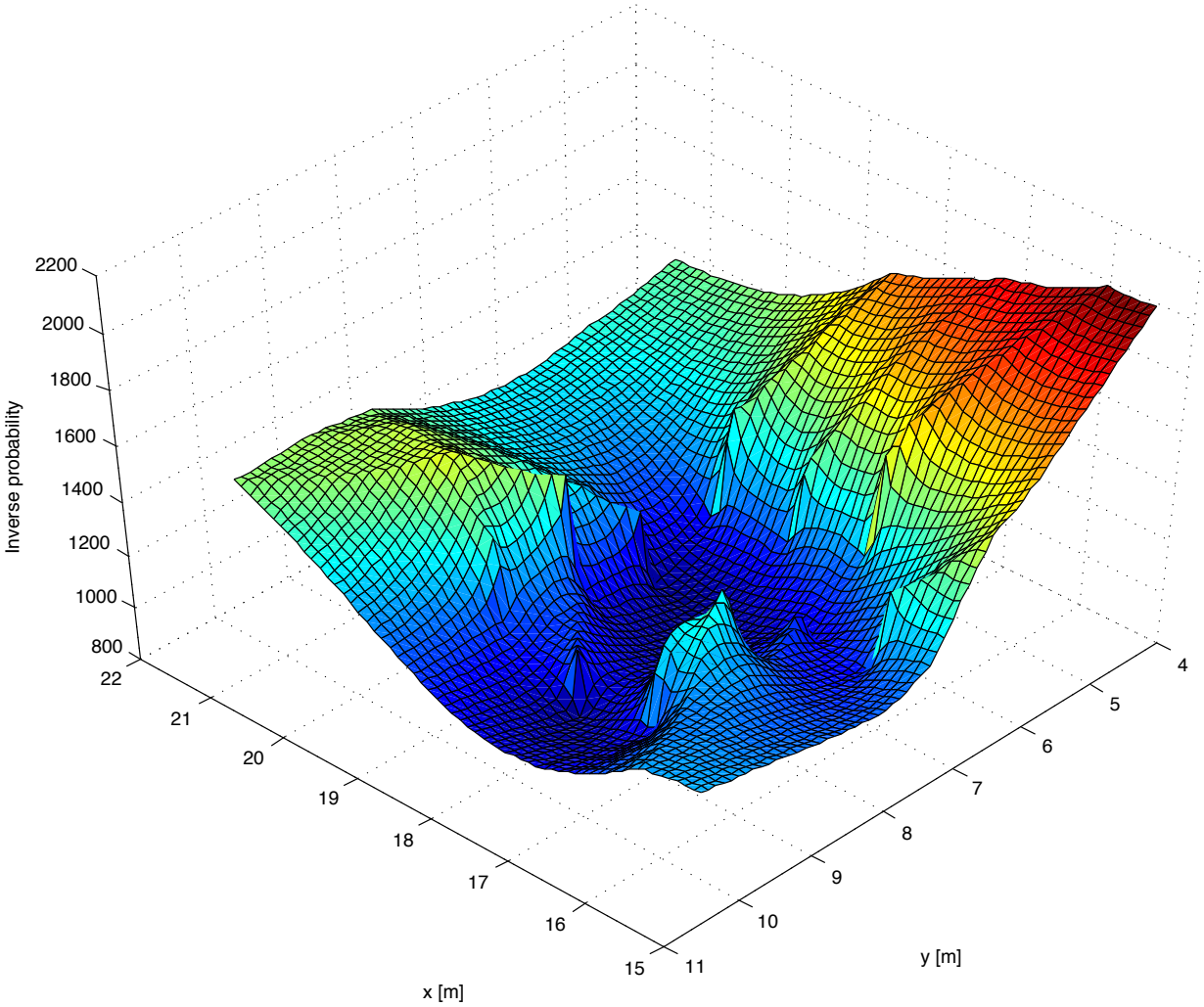

Figure 26: Sample probability distribution of the Local Signal Strength Gradient algorithm

An optimization with common optimization tools is therefore not feasible for the Local Signal Strength Gradient algorithm because its usage would lead to significantly higher error.

3.5 Ground Truth

In the course of this work, measurements were taken in a real-world scenario. The survey was done in an office building with quite thin inner walls (see Figure 27). Four access points with a total of 37 unique BSSIDs were taken into account. At some measuring points, there was direct line of sight to access points, and at other points, no one of the access point could be seen.

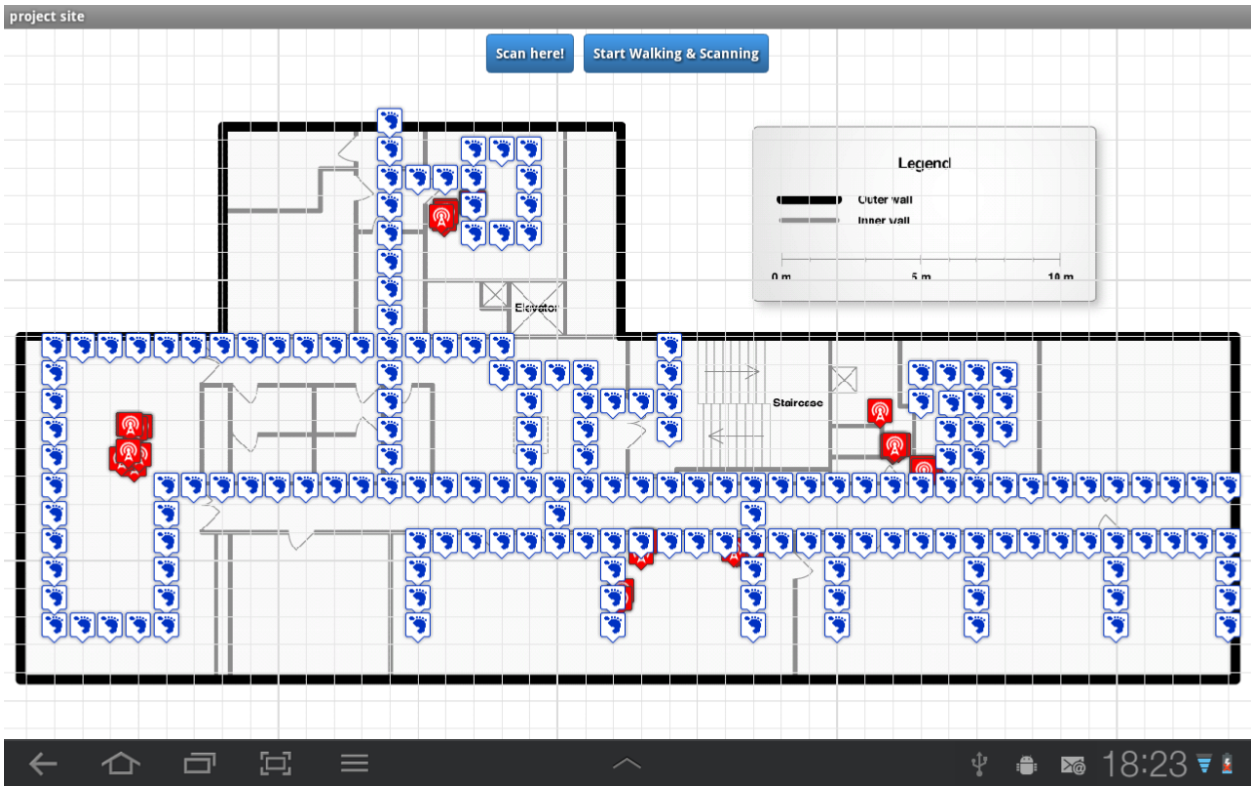

Figure 27: The test environment

In Figure 27, the blue symbols are measuring points and the red symbols are the calculated positions of BSSIDs (note that several BSSIDs are usually configured on one access point). The little arrow at the bottom of each symbol points to the actual location (*not* the center of the symbol). Some of the BSSIDs in Figure 27 have been excluded for the statistics in this chapter, as they were located on different floors.

The survey was done in order to

1. provide good starting points for parameters used in the algorithms presented in this work, and to
2. compare the performance of the algorithms altogether.

Table 2 shows some facts about the data collected[5].

Description	Value
Size of the building	~45 × 20 m
Amount of unique BSSIDs within the floor	37
Amount of access points	4
Amount of measuring points	180
Amount of data sets (position, BSSID and RSSI)	~4,090
Average amount of BSSIDs captured per measuring point	~23
RSSI value range	-94 dBm to -9 dBm

Table 2: Facts about the real-world data collected

3.5.1 Parameter Optimization for the Test Environment

This section is all about finding optimal parameters for the three algorithms described in chapters 3.2, 3.3, and 3.4. Note that though the parameter values proposed here are not supposed to be optimal for all environments, they rather aim to provide a starting point for indoor environments. This is possible through a high amount of data collected in the test scenario described at the beginning of section 3.3.

The parameter optimization was conducted the following way (the steps were repeated for each algorithm):

- A parameter range and interval were defined for each parameter (e.g., range -3 to +2, interval 0.2 for parameter g).
- *All* the BSSID positions of the environment described in the beginning of section 3.5 were calculated for each parameter value (e.g., -3, -2.8, -2.6, etc. for parameter g). If there were two parameters, each and every combination of the parameters was used.
- The calculated positions were compared to the real positions of the corresponding access point.
- The average error of all BSSID positions was computed.
- The data (parameter value(s), average error) was stored in a database.
- The parameters corresponding to the minimum error were identified.

In the following three sections, the optimal parameters for *the given environment* are presented.

[5] The data was captured with a Samsung Galaxy Tab 8.9.

Weighted Centroid

The only parameter used in the Weighted Centroid algorithm is the constant g. For details about the impact of this parameter see section 3.2.1. In the test, a range of 0 to 5 and an interval of 0.1 were used for the parameter (see Table 3).

Parameter	Range	Interval
g	0 through 5	0.1

Table 3: Parameter ranges and intervals for the Weighted Centroid algorithm

All the BSSID positions of the test environment were calculated and the average error was computed. The results can be seen in Figure 28.

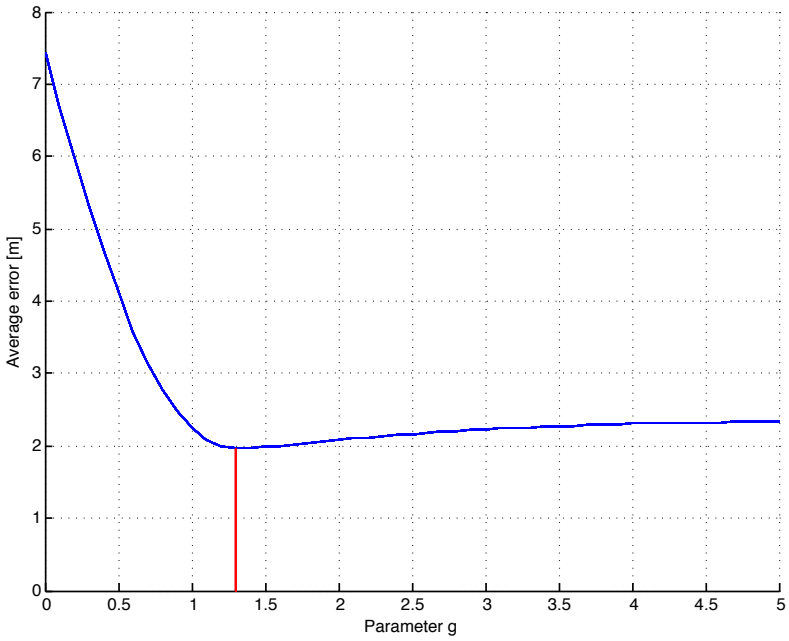

Figure 28: Optimal parameter value g in the test environment (Weighted Centroid)

One can see that the optimal value for the test environment is $g = 1.3$ (see the red line on Figure 28). For values higher than 1.3, the error does not increase significantly. Thus, a g value of 1.3 or slightly higher is recommended for indoor environments. The minimum error for the test environment was recorded at $g = 1.3$ with an **average error of 1.97 m**.

Advanced Trilateration

In the Advanced Trilateration algorithm, the parameters C and n are used. For details about their impact see section 3.3.1. In the test, each and every combination of the values depicted in Table 4 was used.

Parameter	Range	Interval
C	-50 dBm through -24dBm	1 dBm
n	2.1 through 5	0.1

Table 4: Parameter ranges and intervals for Advanced Trilateration algorithm

Again, all combinations were taken and the calculated positions were compared with the real positions. Then, the average error was computed. The results can be seen in Figure 29 (note that because there are two parameters, the results were plotted in a three-dimensional graph).

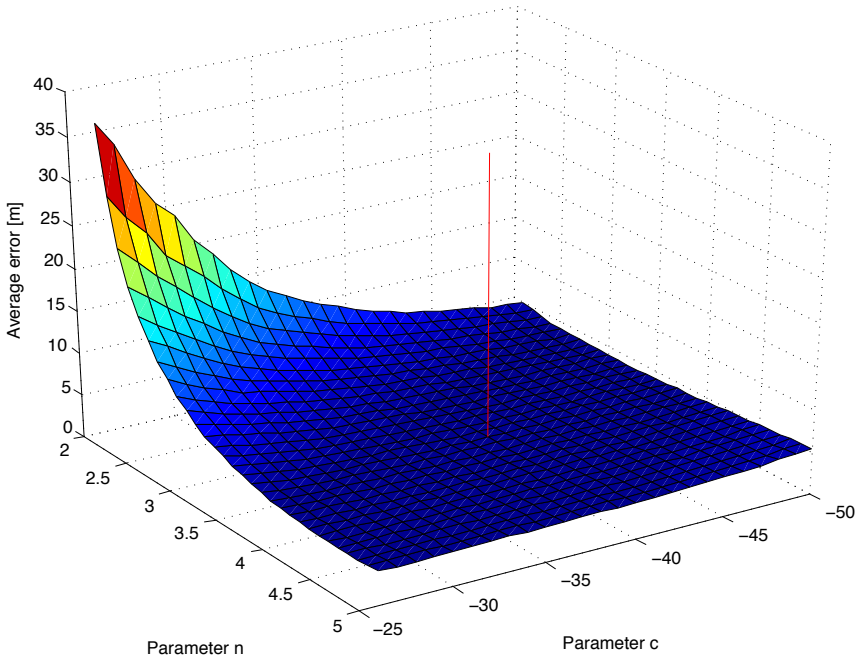

Figure 29: Optimal parameter values C and n in the test environment (Advanced Trilateration)

The dark blue area in Figure 29 indicates the best parameter combination of C and n for the Advanced Trilateration algorithm in the test environment (the red vertical line indicates the best parameter combination). The minimum error for the test environment was recorded at $C = -39$ and $n = 3.7$ with an **average error of 3.87 meters**.

Local Signal Strength Gradient

The Local Signal Strength Gradient algorithm uses two parameters, the window size and the constant g. For the test, ranges and intervals were used as depicted in Table 5.

Parameter	Range	Interval
$\boldsymbol{Window\ size}$	2 m through 4 m	1 m[6]
$\boldsymbol{g}$	-3 through +2	0.2

Table 5: Parameter ranges and intervals for Local Signal Strength Gradient algorithm

As with all the other algorithms, the average error was calculated for each and every combination of the parameters in Table 5. There are two parameters to be considered, so the results in Figure 30 are shown in a three-dimensional graph.

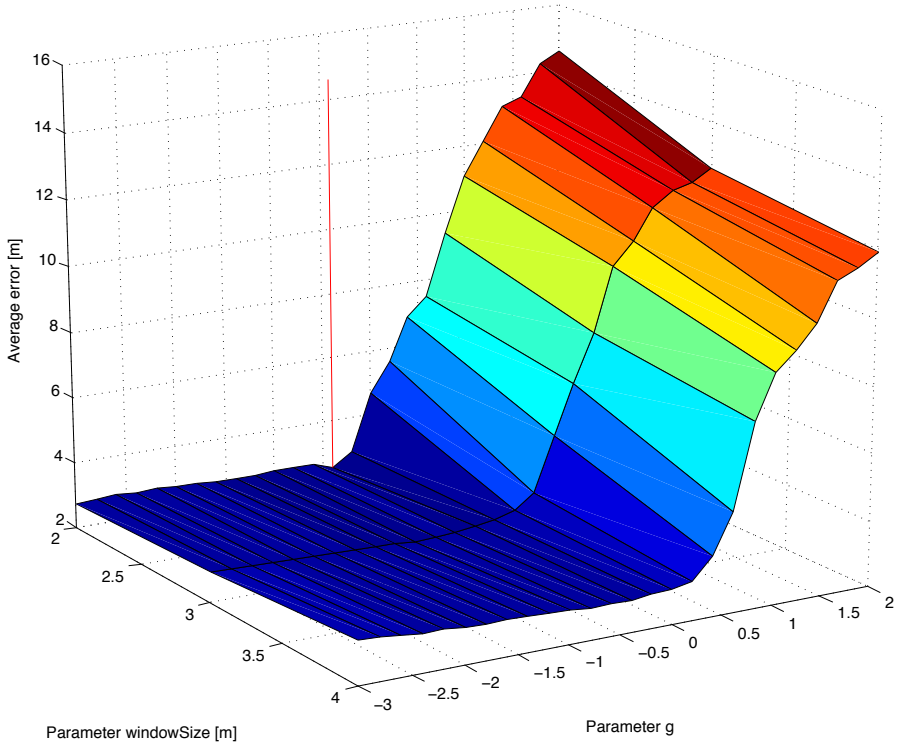

Figure 30: Optimal parameter values C and n in the test environment (Local Signal Strength Gradient)

The dark blue area in Figure 30 indicates the best parameter combination of the window size and g for the Local Signal Strength Gradient algorithm in the test environment (the red vertical line indicates the spot). The minimum error for the test environment was recorded at $windowSize = 2$ and $g = -0.4$ with an **average error of 2.52 meters**.

[6] The window size interval was 1 m because in the test environment, all measuring point positions were snapped to a grid of 1m. This can be seen in Figure 27. Thus, the results using a window size of e.g. 2.5 meters wouldn't differ from the results using a window size of 2 meters.

3.5.2 Performance Comparison

This section aims to compare the real-world performance of all three trilateration algorithms presented in this work. The data is taken from the test environment described in the beginning of section 3.5. Figure 31 shows the results of the survey. The yellow stars are the real positions of the access points. The small dots are the calculated positions of the BSSIDs (note that in the test scenario, there are several BSSIDs per access point). The dashed line connects the estimated position of the BSSID with its real position.

In this scenario, Weighted Centroid (blue dots) performs best, closely followed by the optimized Advanced Trilateration algorithm (orange dots). The non-optimized Advanced Trilateration algorithm (red dots) has a couple of outliers. On the top third of Figure 31, the Advanced Trilateration algorithm even has an outlier that is almost 22 meters away from its real position. The Local Signal Strength Gradient algorithm (green dots) has a good overall performance with a couple of outliers.

Table 6 shows different performance values for the discussed algorithms. These have been measured in the test environment using the best parameters for each algorithm in terms of its average error (for the ideal choice of parameters see section 3.5.1).

Performance indicator	Weighted Centroid	Advanced Trilateration	Advanced Trilateration (optimized)	Local Signal Strength Gradient
Average error	1.97 m	3.87 m	2 m	2.52 m
Other performance indicators				
Median error	1.60 m	3.01 m	1.62 m	1.91 m
Maximum error	4.02 m	21.97 m	4.7 m	5.87 m
Average time to calculate one BSSID[7]	~0.2 ms	~4.7 s	~25 ms	~42 s

Table 6: Performance comparison of the discussed algorithms

In this scenario, the **Weighted Centroid algorithm performs best in every way**. It has the lowest average error, the lowest median error, the lowest maximum error and the average time to calculate one BSSID significantly outruns all the other algorithms.

[7] For the comparison, the average calculation time of one BSSID in the test environment was computed. The test was conducted with a MATLAB implementation of the algorithms and run on a MacBook with an Intel Core 2 Duo 2.4 GHz processor and 4 GB of RAM.

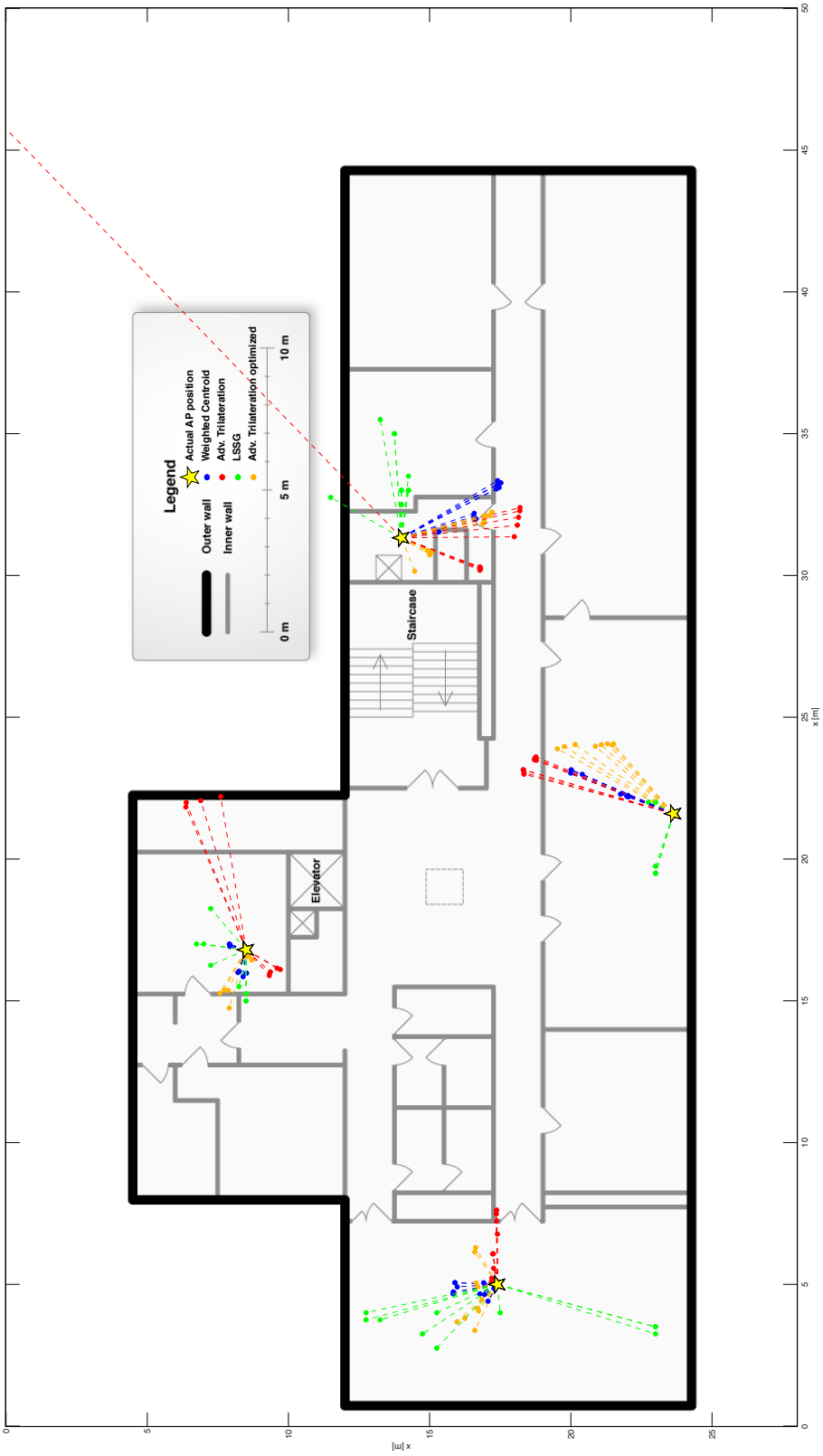

Figure 31: Comparison of performance of the described algorithms (best parameters)

3.5.3 Performance Comparison: Special Measuring Circumstances

A second test was conducted where measuring points were taken away. The aim was to provide an overview of how the algorithms perform when the measuring points do not entirely surround the actual access point. To achieve this, measurement points that were at a certain distance from the access point or closer were not taken into account. Figure 32 illustrates this. For AP1 and AP4 (green and yellow stars and corresponding lines and shadings), the minimal distance to the measuring points taken into account was 10 m. For AP2 and AP3 (blue and red stars and corresponding lines and shadings), the minimal distance was 6 meters.

The results are very inaccurate. Figure 33 shows the results graphically. None of the algorithms got close to any actual position. There are lots of outliers, especially with the Local Signal Strength Gradient algorithm. The Weighted Centroid algorithm is again the best of all algorithms, although the graphical centroid of all measuring points cannot even lie outside the measuring points themselves. Table 7 gives a numeric summary of the results.

Performance indicator	Weighted Centroid	Advanced Trilateration	Advanced Trilateration (optimized)	Local Signal Strength Gradient
Average error	12.4 m	13.35 m	13.14 m	14.2 m
Other performance indicators				
Median error	13.73 m	14.97 m	15.01 m	11.86 m
Maximum error	19.8 m	22.62 m	21.14 m	39.9 m

Table 7: Performance comparison of the discussed algorithms under special circumstances

The numbers show that the Weighted Centroid algorithm does not only have the lowest average error, but also the lowest median and maximum error. Thus, **even in scenarios where the actual access point lies outside the range of the measuring points, Weighted Centroid performs best**.

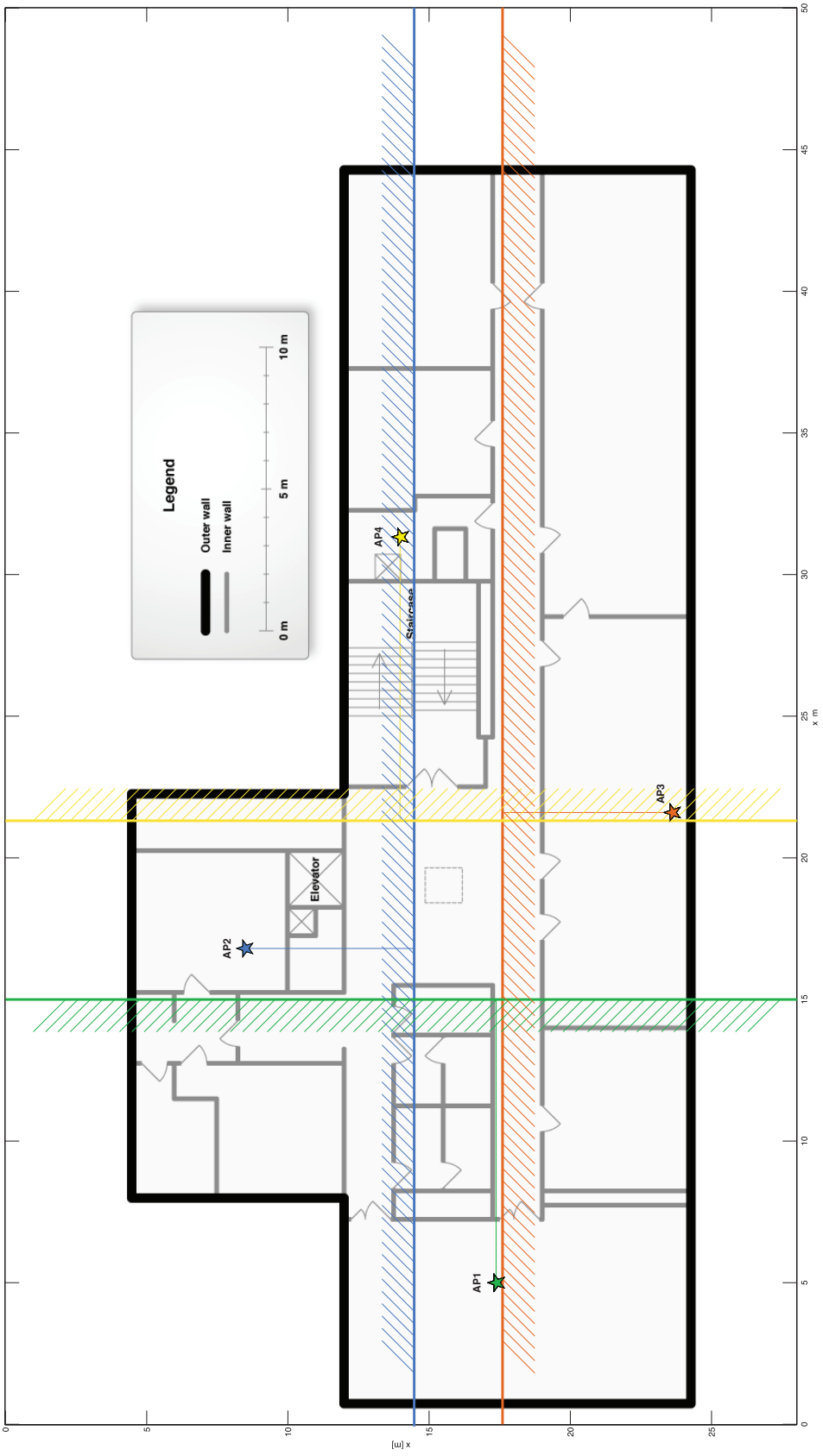

Figure 32: Measuring points from the shading towards the particular access point were not taken into account for the performance comparison with special circumstances

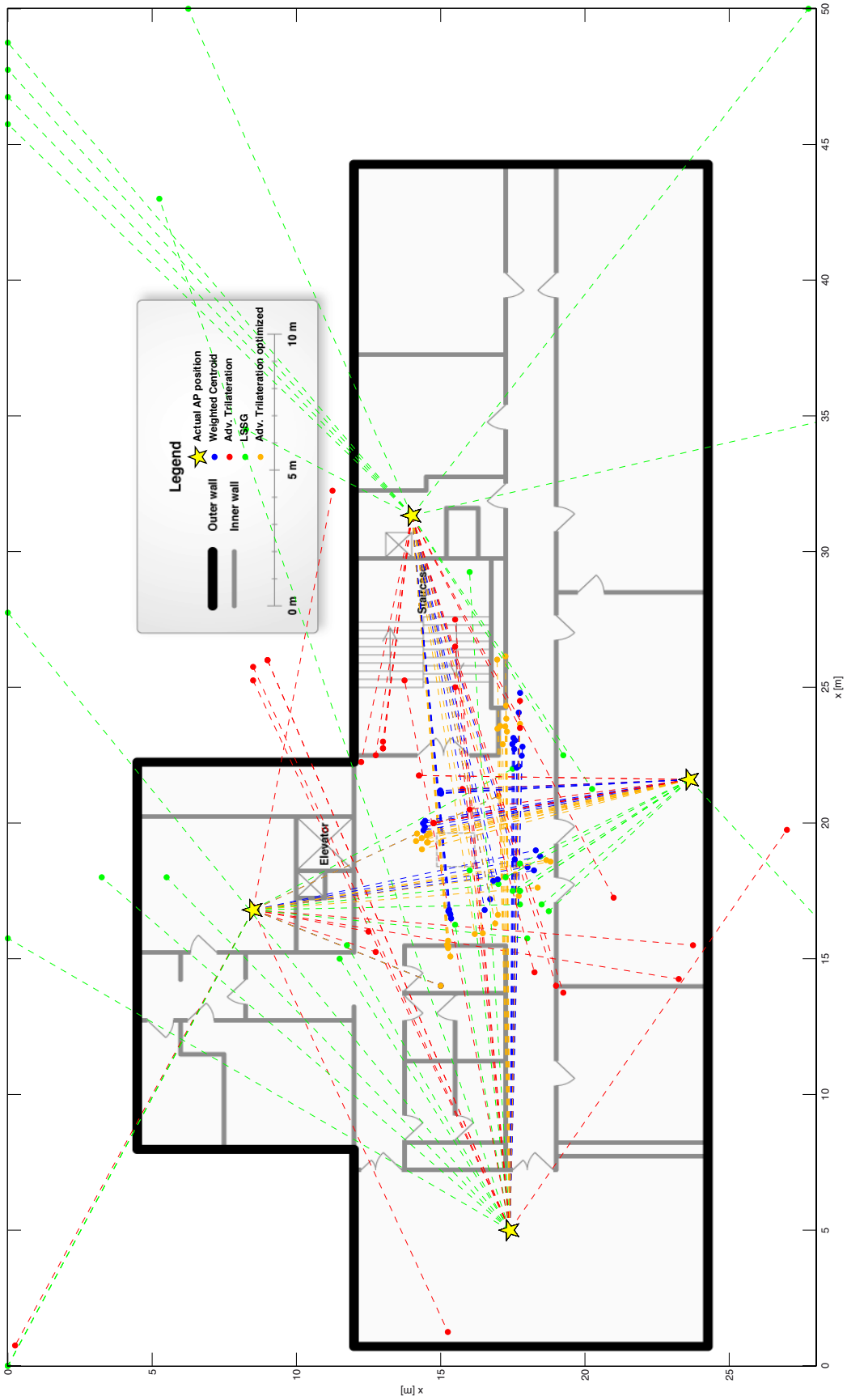

Figure 33: Comparison of performance of the described algorithms (best parameters)

3.5.4 Conclusion

In all tests, the **Weighed Centroid algorithm performed best** in the given indoor scenario. Weighted Centroid is best in the following aspects:

- **Accuracy**. It has the lowest average, median and maximum error in both test scenarios.
- **Speed**. Weighted Centroid is by far the fastest of all algorithms. It is about 125 times faster than the optimized Advanced Trilateration algorithm, about 24.000 times faster than the non-optimized Advanced Trilateration algorithm and over 200.000 times faster than the Local Signal Strength Gradient algorithm.
- **Ease of implementation**. The MATLAB implementation of the Weighted Centroid algorithm has only eight lines of code. The MATLAB implementations of the other algorithms are substantially longer.
- **Measurement point distribution.** The Weighted Centroid algorithm returns results with only two measuring points. The other algorithms need more measuring points to give results at all. Moreover, the Local Signal Strength Gradient algorithms requires most of the measuring points to have a maximum distance of its window size (2—3 meters) so that the arrows can be calculated.

The Weighted Centroid has therefore been implemented in WiFi Compass and is used as the default algorithm.

3.6 Realization in WiFi Compass

Both the Weighted Centroid algorithm and the Local Signal Strength Gradient algorithm have been implemented in WiFi Compass. The following paragraphs describe the implemented classes.

`AccessPointTrilateration`: This is the super class of all implemented algorithms. The `ProjectSite`, which contains all collected measurements, is passed to the constructor as an argument.

- `calculateAccessPointPosition()`: This abstract method takes the access point as an argument and calculates the position of the access point.
- `calculateAllAndGetDrawables()`: This method calculates all positions and creates an `AccessPointDrawable` object for each access point. An array of these `AccessPointDrawable`s is then returned.
- `parseMeasurementData()`: This protected method parses the measurement points in the `ProjectSite` and creates corresponding `MeasurementDataSet` objects (see below). This is done to reduce the amount of data stored in memory during calculation, because the `ProjectSite` contains way more data than needed (such as access point capabilities).

`MeasurementDataSet`: This class is used to hold the measurement data. A `ProjectSite` object is passed to the constructor of `AccessPointTrilateration`. The `parseMeasurementData()` method then parses out the measurement points and their corresponding BSSIDs and `MeasurementDataSet` objects.

- `x`: The x coordinate of the measurement.
- `y`: The y coordinate of the measurement.
- `RSSI`: The received signal strength of the BSSID at the given point.

`LocalSignalStrengthGradientTrilateration`: This class holds an implementation of the Local Signal Strength Gradient algorithm (see section 3.4 for details).

- `areAllMeasurementsCollinear()`: This method checks whether a given set of points is collinear, or in other words, whether or not all of those lie on one line or not.
- `arePointsCollinear()`: Just like `areAllMeasurementsCollinear()`, but only for a set of three points. This method is used by `areAllMeasurementsCollinear()`.
- `getMeasurementDataWithinWindow()`: Returns all measurement points that are within a given window size seen from the current measurement point.
- Class `GradientArrow`: This class holds an arrow needed for the Local Signal Strength Gradient algorithm. An arrow holds the coordinates of a measurement point on the map and the direction towards which the access point is most likely to be located, seen from that measurement point.
- `calculateAccessPointPosition()`: The implementation of the abstract method of the superclass.
- `calculateAllAndGetDrawables()`: The implementation of the abstract method of the superclass.

`WeightedCentroidTrilateration`: This is an implementation of the Weighted Centroid algorithm (see section 3.2 for details).

- `calculateAccessPointPosition()`: The implementation of the abstract method of the superclass.
- `calculateAllAndGetDrawables()`: The implementation of the abstract method of the superclass.

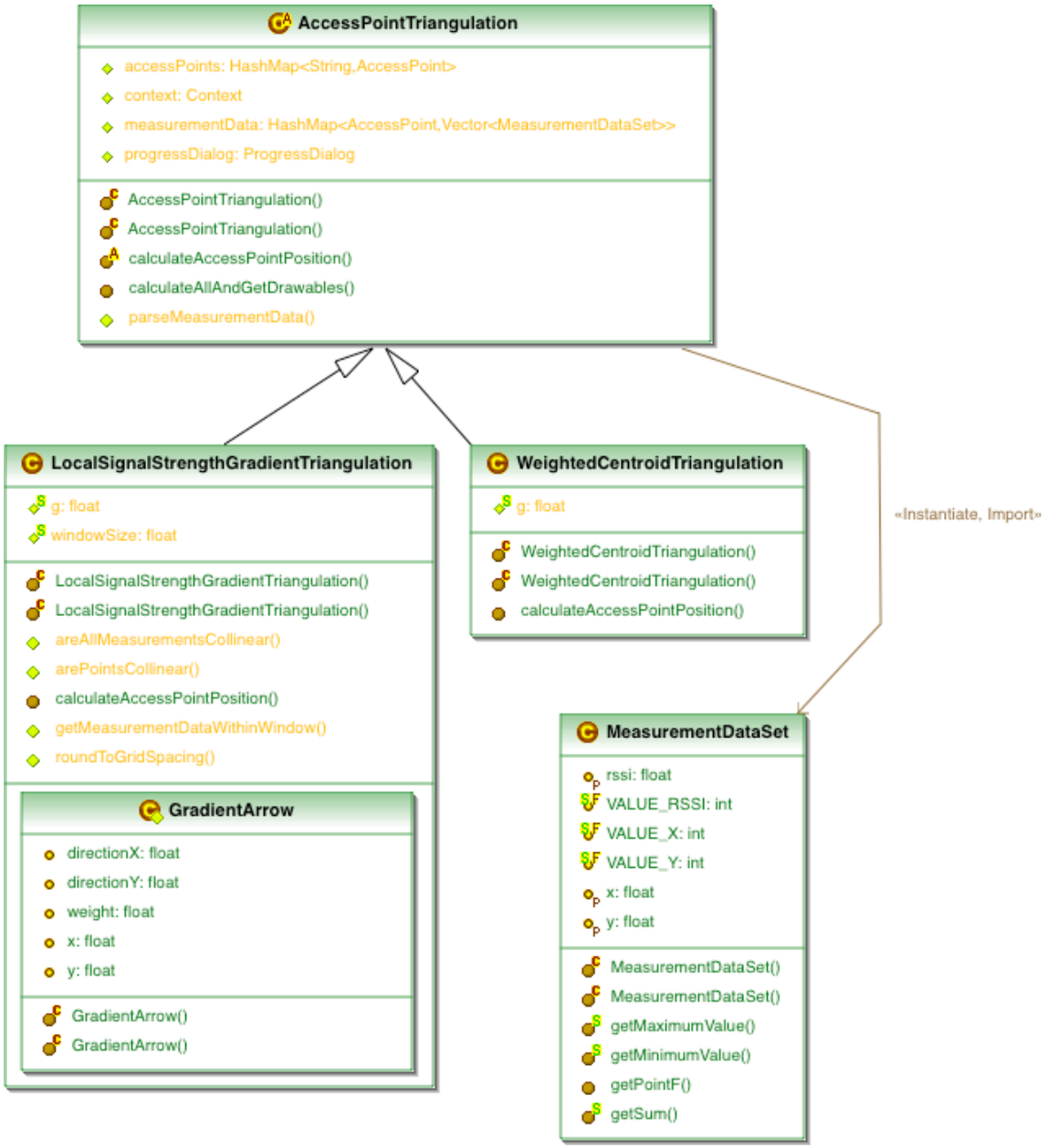

Figure 34: Class diagram of the realization of trilateration algorithms in WiFi Compass

Figure 34 shows the class diagram of the implementation. Note that the Weighted Centroid algorithm is used as default in WiFi Compass as it has proven to be the best algorithm in the test environment (see section 3.5 for details on how the test was conducted). The implementation of the Local Signal Strength Gradient algorithm is kept in the source code for reference.

3.7 Solving an Over-determined Equation System using the Method of Least Squares

In this work, both the Advanced Trilateration algorithm and the Local Signal Strength Gradient algorithm try to solve equations where the number of equations is bigger than the number of unknowns. In both algorithms, the method of least squares is used to calculate the best-fitting plane for a set of points in a 3-dimensional coordinate system. In this section, the method of least squares is explained in more detail. All the examples and descriptions in this section are taken from [**27**].

The problem

Imagine having equations that outnumber the unknowns. An example can be seen in Figure 35.

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 \\ y_3 &= a_0 + a_1 x_3 + a_2 x_3^2 \\ y_4 &= a_0 + a_1 x_4 + a_2 x_4^2 \\ y_5 &= a_0 + a_1 x_5 + a_2 x_5^2 \\ y_6 &= a_0 + a_1 x_6 + a_2 x_6^2 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \\ 1 & x_6 & x_6^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{X a = y} \quad .$$

Figure 35: An over-determined equation system [27]

In that example, six equations with three unknowns (a_0, a_1 and a_2) are given. Usually, there is no solution that fits all six equations [**27**]. In every equation, there will be a deviation v_i (see Figure 36 for its calculation).

$$v_i = y_i - \left(a_0 + a_1 x_i + a_2 x_i^2\right)$$

Figure 36: The deviation for a set of solutions [27]

The solution

Now the solution of the equation system is calculated so that the sum of the squared deviation for all equations is minimum (see Figure 37).

$$S = \sum_{i=1}^{6} v_i^2 = \sum_{i=1}^{6} \left[y_i - \left(a_0 + a_1 x_i + a_2 x_i^2\right) \right]^2 \Rightarrow Minimum \; .$$

Figure 37: Minimization of the sum-squared deviation of all equations [27]

In mathematics, in order to find the minimum of a function, the partial derivative is set to 0 (see Figure 38).

$$\frac{\partial S}{\partial a_0} = 0 \quad \Rightarrow \quad -2\sum_{i=1}^{6}\left[y_i - \left(a_0 + a_1 x_i + a_2 x_i^2\right)\right] = 0 \quad ,$$

$$\frac{\partial S}{\partial a_1} = 0 \quad \Rightarrow \quad -2\sum_{i=1}^{6}\left[y_i - \left(a_0 + a_1 x_i + a_2 x_i^2\right)\right] x_i = 0 \quad ,$$

$$\frac{\partial S}{\partial a_2} = 0 \quad \Rightarrow \quad -2\sum_{i=1}^{6}\left[y_i - \left(a_0 + a_1 x_i + a_2 x_i^2\right)\right] x_i^2 = 0 \quad .$$

Figure 38: Partial derivative of the sum-squared deviation functions [27]

The so-called “normal equations” are parts of the equation system to calculate a_0, a_1 and a_2 (see Figure 39). Note that there are no indices below the sigma signs. This is because the equation system can be solved for any number of points (in this case, the number of points is $n = 6$).

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 \\ \sum x_i & \sum x_i^2 & \sum x_i^3 \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \end{bmatrix} .$$

Figure 39 “Normal equations” system to calculate a_0, a_1 and a_2 [27]

Once the equation system in Figure 39 is solved, a_0, a_1 and a_2 are the values that minimize the sum-squared deviation (or error) of the function $y = a_0 + a_1 x + a_2 x^2$ for any given set of points $p_i = (x_i i, y_i)$.

4 Combination of Position Detection and Access Point Trilateration

This chapter gives an overview on how Step Detection (chapter 2) and Access Point Trilateration (chapter 3) have been combined into one single software product. The outcome is an Android application that allows a user to effectively locate access points within a building.

4.1 The Multi-Touch User Interface

This section describes how the user interface of the Project Site Activity was built. For other user interface components see section 4.3.

The Site Map is based on the “Android-multitouch-controller” library [**28**]. The library has been used to implement the drag, rotate and pinch-zoom functionality of the site map. The following list cites some of the library’s functionality from [**28**]:

- “It filters out "event noise" on Synaptics touch screens (G1, MyTouch, Nexus One) -- for example, when you have two touch points down and lift just one finger, each of the ordinates X and Y can be lifted in separate touch events, meaning you get a spurious motion event (or several events) consisting of a sudden fast snap of the touch point to the other axis before the correct single touch event is generated.”
- “It simplifies the somewhat messy and inconsistent MotionEvent touch point API -- this API has grown from handling single touch points, potentially with packaged event history (Android 1.6 and earlier) to multiple indistinguised touch points (Android 2.0) to the potential for handling multiple touch points that are kept distinct even if lower-indexed touchpoints are raised (and thus each point has an index and generates its own indexed touch-up/touch-down event). All this means there are a lot of API quirks you have to be aware of. This MultiTouch Controller class simplifies getting access to these events for applications that just want event positions and up/down status.”
- “The controller also supports pinch-zoom, including tracking the transformation between screen coordinates and object coordinates. It correctly centers the pinch operation about the center of the pinch (not about the center of the screen, as with most of the "Google Experience" apps that added their own pinch-zoom capability in Android-2.x).”
- “The controller was recently updated to support pinch-rotate, allowing you to physically twist objects using two touch points on the screen. In fact all of rotate, scale and translate can be simultaneously adjusted based on relative movements of the first two touch points.”
- “The controller makes it very easy to work with a canvas of separate objects (e.g. a stack of photos), each of which can be separately dragged with a single touch point or scaled with a pinch operation.”

Using this library, the site map of WiFi compass can be zoomed, rotated and moved with multi-touch gestures (see Figure 40).

Figure 40: Multi-touch functionality of the site map

The software does so by first instantiating an object of the `MultiTouchView` class (see Figure 41). This class is derived from the Android-internal `android.view.View` class and can therefore be added to any Android UI layout. The class moreover implements the `MultiTouchObjectCanvas` interface which forces the View to implement methods such as `getDraggableObjectAtPoint()`. This and other methods are used by the `MultiTouchController` to, for example, detect whether there is a multi-touch object below the point that the user touches on the Screen. Another method of the interface is `setPositionAndScale()`, which sets the position and scale values of a given object on the canvas.

The MultiTouchView then instantiates an object of MultiTouchController and passes itself as a parameter to the constructor of the latter (see Listing 7). The generic (the type of objects handled by the MultiTouchController) is the MultiTouchDrawable class. This is the class of the objects that are drawn on the canvas.

```
1 private MultiTouchController<MultiTouchDrawable> multiTouchController = new
  MultiTouchController<MultiTouchDrawable>(this);
```

Listing 7: The instantiation of the MultiTouchController object inside MultiTouchView

The View then passes each touch event to the controller (see Listing 8). That is where the move, pinch-zoom and rotate actions are detected and reacted upon.

```
2 /** Pass touch events to the MT controller */
3 @Override
4 public boolean onTouchEvent(MotionEvent event) {
5     boolean handled = multiTouchController.onTouchEvent(event);
6     invalidate();
7     return handled;
8 }
```

Listing 8: The MultiTouchView passes each touch event to the MultiTouchController

The MultiTouchView has a method called onDraw(). It is called internally by Android every time the invalidate() method is invoked on the View (see Listing 9). The method loops through all drawable objects and invokes their draw() method.

```
1  @Override
2  protected void onDraw(Canvas canvas) {
3      super.onDraw(canvas);
4      int n = drawables.size();
5      // Logger.d("drawing " + n + " drawables");
6
7      for (int i = 0; i < n; i++)
8            drawables.get(i).draw(canvas);
9      if (mShowDebugInfo)
10           drawMultitouchDebugMarks(canvas);
11 }
```

Listing 9: The onDraw() method of MultiTouchView

“Invalidate” in this case means that graphical objects on the canvas have changed, for example, their size or scale. It is called, among others, in the setPositionAndScale() method of the View (see line 7 of Listing 10).

```
1 /** Set the position and scale of the dragged/stretched image. */
2 public boolean setPositionAndScale(MultiTouchDrawable drawable,
3           PositionAndScale newImgPosAndScale, PointInfo touchPoint) {
4     currTouchPoint.set(touchPoint);
```

```
5       boolean ok = drawable.setPos(newImgPosAndScale, true);
6       if (ok)
7               invalidate();
8       return ok;
9   }
```

Listing 10: The `setPositionAndScale()` method of `MultiTouchView`

`MultiTouchDrawable` objects can be stacked hierarchically. This means that one object can contain several others, which are dragged along with the superior object. They are also zoomed and rotated depending on whether they return `true` or `false` in their `isScalable()` and `isRotateable()` methods. However, sub objects always stay at the same relative position to their superior object, not matter whether they are draggable, scalable and rotatable or not. If they are neither scalable nor rotatable, they simply stay the same size and un-rotated relative to the screen, no matter what the scale and rotation level of the superior object is.

The object-oriented design of `MultiTouchDrawable` and it s subclasses follows the composite design pattern. The `MultiTouchDrawable` class has several methods and member variables (see Figure 41). The most important ones are the following:

- `addSubDrawable()`: Adds a subsidiary object. The sub object always moves along with its super object (and stays at the same relative position to it), no matter how the latter is scaled or rotated. Depending on whether the sub object is scalable and rotatable, the sub object also adapts the scale and rotation properties of its super object.
- `draw()`: This is where the graphics are painted. The canvas that the object is painted on is passed to the `draw()` method as an argument. The canvas is then translated, rotated and scaled depending on the properties of the object. Then the object paints, for example, bitmaps, lines, rectangles and text on the transformed canvas.
- `drawSubdrawables()`: This method is usually called at the end of the `draw()` method. After the object has been painted on the canvas, its sub objects need to be painted. This happens after the super object because otherwise the super object would paint over its sub objects.
- `getGridSpacingX()` / `getGridSpacingY()`: These methods return the amount of pixels that correspond to one meter. These values are calculated depending on how the user sets the scale of the map.
- `isDragable()`: Returns whether or not the user can drag the object over the screen or not. If the object is located inside a super object, the object is still dragged along with its super object. But the user cannot drag the object by hand. This is the case, for example, with measuring points on the map. They are located at a fixed position. They are moved along with the map, but the user cannot change the relative positions to the map retrospectively.
- `isRotatable()`: Returns whether or not the object is rotatable. If it is not, the object is not rotated, even if its super object is. This means that the object keeps the same rotation relative to the screen.

- `isScalable()`: Returns whether or not the object is scalable. If it is, the object scales along with its super object. If it is not, it keeps the same scale relative to the entire screen even if the super object is scaled.

There are several classes derived from `MultiTouchDrawable`. The most important ones can be seen in Figure 41.

- `SiteMapDrawable`: This is the class that holds the map image, the grid and sub objects such as `UserDrawable`s, `MeasuringPointDrawable`s, and `AccessPointDrawable`s. It is draggable, scalable and rotatable and is the root object of all graphics objects of the `ProjectSiteActivity` in WiFi Compass.
- `UserDrawable`: The user icon. The user can be dragged over the map so that its relative position can be changed. However, the object is neither scalable nor rotatable, which means that it always keeps the same rotation and scale factor relative to the screen, no matter what is the scaling and rotation of the `SiteMapDrawable`.
- `MeasuringPointDrawable`: This class holds the measuring point icon. It is drawn when a measurement is done. Its relative position to the `SiteMapDrawable` cannot be changed, nor can it be scaled or rotated. This can be seen on Figure 40 (the small blue icons).
- `AccessPointDrawable`: An object of this class is added as soon as the access point positions have been calculated. As with `MeasuringPointDrawable`s, its relative position is fixed and it can neither be scaled or rotated.

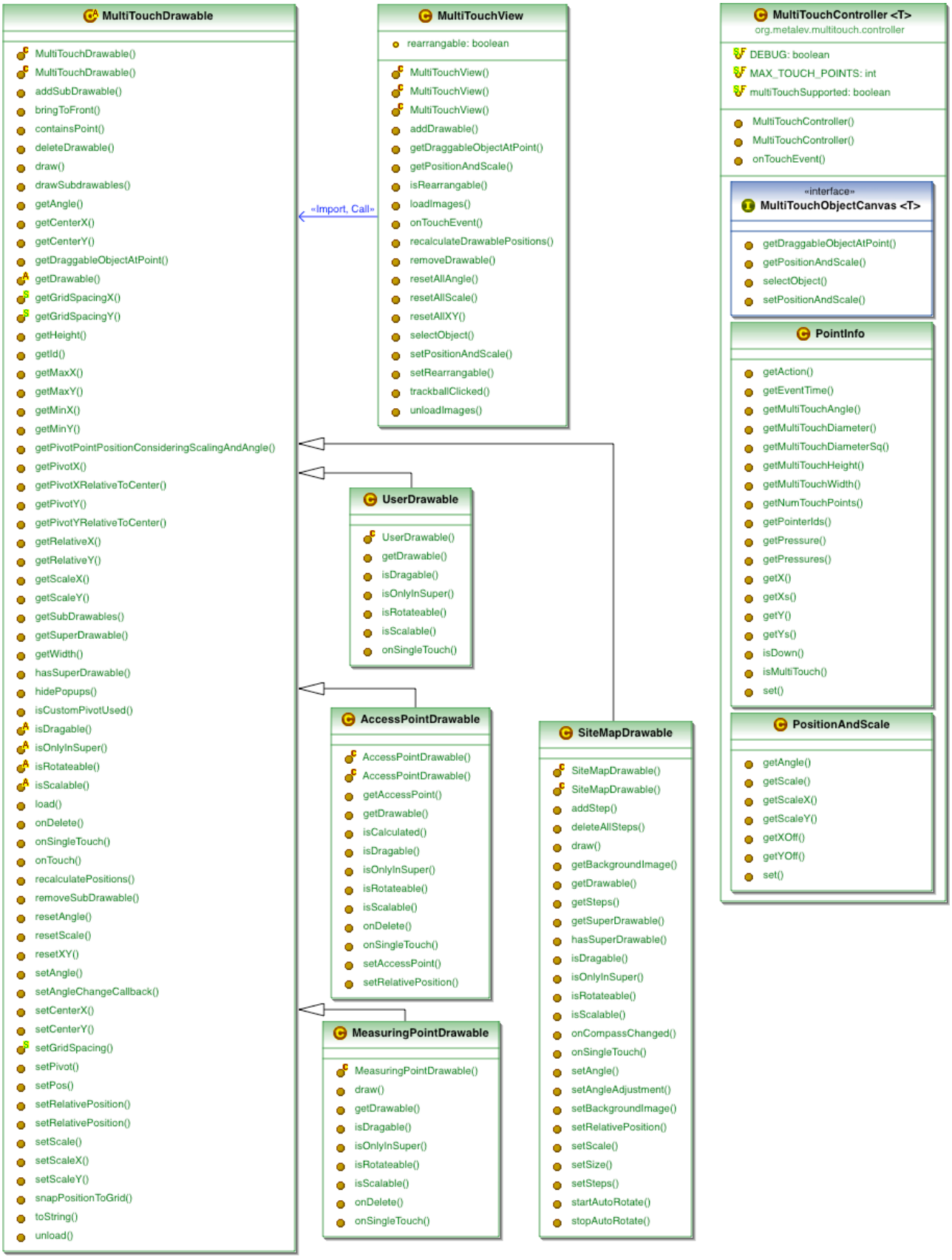

Figure 41: Cutout of the class diagram of view classes

The `MultiTouchController` (see Figure 41) gets all touch events of the `MultiTouchView`. Here, the touch events are detected and handled. When, for example, the view is tapped with one finger, the `MultiTouchController` gets the touch event and asks the `MultiTouchView` via the `getDraggableObjectAtPoint()` method whether there is an object below the finger (see Listing 11).

```
1  private void multiTouchController() {
2      [SNIP]
3      switch (mMode) {
4      case MODE_NOTHING:
5            // Not doing anything currently
6            selectedObject = objectCanvas.getDraggableObjectAtPoint(mCurrPt);
7      [SNIP]
```

Listing 11: The call of the `getDraggableObjectAtPoint()` method in the `MultiTouchController`

If so, the controller detects a finger movement and sets the `mMode` member variable to `MODE_DRAG`. When the user continues dragging their finger over the screen, the `performDragOrPinch()` private method is called. This is where the position, scale and rotation of the currently selected object are actually performed (see Listing 12). The new position and scale values are set on the object by calling the `setPositionandScale()` method of the view and passing the current object and the position and scale data (see Line 24 of Listing 12).

```
1  /** Drag/stretch/rotate the selected object using the current touch position(s)
2   * relative to the anchor position(s). */
3  private void performDragOrPinch() {
4      // Don't do anything if we're not dragging anything
5      if (selectedObject == null)
6            return;
7
8      // Calc new position of dragged object
9      float currScale = !mCurrXform.updateScale ?
10                 1.0f : mCurrXform.scale == 0.0f ? 1.0f : mCurrXform.scale;
11     extractCurrPtInfo();
12     float newPosX = mCurrPtX - startPosX * currScale;
13     float newPosY = mCurrPtY - startPosY * currScale;
14     float newScale = startScaleOverPinchDiam * mCurrPtDiam;
15     float newScaleX = startScaleXOverPinchWidth * mCurrPtWidth;
16     float newScaleY = startScaleYOverPinchHeight * mCurrPtHeight;
17     float newAngle = startAngleMinusPinchAngle + mCurrPtAng;
18
19     // Set the new obj coords, scale, and angle as appropriate
20     // (notifying the subclass of the change).
21     mCurrXform.set(newPosX, newPosY, newScale, newScaleX, newScaleY, newAngle);
22
23     boolean success = objectCanvas
24                       .setPositionAndScale(selectedObject, mCurrXform, mCurrPt);
25     if (!success)
26           ; // If we could't set those params, do nothing currently
```

```
27 }
```

Listing 12: The `performDragOrPinch()` method in `MultiTouchController`

The pinch zoom gesture is handled in a similar manner. The `anchorAtThisPositionAndScale()` private method of the `MultiTouchController` is used to calculate the new scale factors of the object depending on the distance of the fingers on the screen. The next time `setPositionAndScale()` is called, the new values are passed to the affected object via the View.

4.2 The Model

The model of WiFi Compass was developed step by step in a prototyping process. The initial idea was to develop an Android application, which can be used like a WiFi survey software. This is also the reason why WiFi scans are divided into projects and project sites. One project can correspond to a building, or location, where the scans are conducted. A building or building block can have several stories, which must be divided into different maps. WiFi compass does not support multiple floors in one map; it is designed to work with separate project sites for each floor. A project only holds a title and a description, and is referred to by the project sites. All project sites assigned to one project are shown in the project page and can be selected as show in Figure 42. The corresponding class is `Project`, which is shown in the UML Diagram in Figure 43.

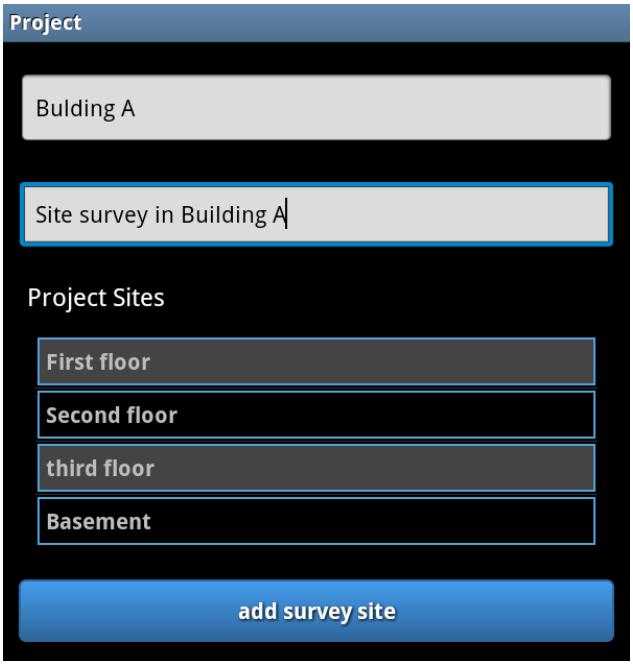

Figure 42: WiFi Compass projects & project sites

The central class, in which the most important information is linked together, is `ProjectSite`. As mentioned above, a project site typically corresponds to a floor in a building. A site has a title

and a description. The description was planned and is implemented in the model, but not implemented in the UI, because in the prototyping phase the authors realized that it is not needed. To visualize a floor, a map can be used. This map can be loaded from a file on the SD card of the Android device. Currently, Android supports JPEG, GIF, PNG, and BMP files. From Android 4.0 onwards, WEBP is also supported [**29**]. The bitmap image is also used in the model, but persisted as a byte array. Because Android handles conversion between the formats and decoding / encoding of images, this is a very flexible solution to handle different file formats for map images.

The boundaries of a map are defined with its width and height in pixels. All calculations in the application are done with pixels. To create a relation to real world distances, a grid spacing can be defined, which defines how many pixels correspond to one meter. The default value is 30, but this can be changed after the map image is loaded. The user does not have to know how many pixels correspond to one meter; it is possible to mark a distance on the map and tell the device how many meters this is. For more details, see section 2.4.1 Basic User Interface to locate and track a user, Figure 8: Distance selection and Figure 9: Defining proportion of pixels to meters, and section 4.1 The Multi-Touch User Interface.

To make use of step detection, the geographic north of the map must be known by the application. This can be defined by the user and is saved as a deviation angle in radiant (a_m). A deviation angle of 0 would correlate with the top of the map heading directly northbound; $\pi/2$ correlates with the device pointing westbound. Normally, $\pi/2$ would be applied to a circle clockwise and therefore correlate with an east alignment. However, the deviation angle defines the direction where the user has to move and therefore is defined as the inverted angle. The calculation is described in chapter 2.4.2 Step Detection Algorithm. As a result, $\pi/2$ correlates with the map's top heading westbound.

When a project site is loaded, the grid spacing and the deviation angle are set on `LocationService`, so all `LocationProviders` can access these parameters for calculations. Both parameters are needed for step detection, because it uses the compass to get the heading of the user and calculates the distance of one step in pixels.

The project site holds a reference to all WiFi scan results of this site. A `WiFiScanResult` consists of the location, the time when the results were received, and a list of BSSIDs. Each `BssidResult` contains the SSID, the frequency of the 802.11 channel, the signal level (RSSI) and the capabilities of the network. These `WifiScanResults` and `BssidResults` are used to calculate the location of the access points.

Access points either found or defined on the map are also referenced to from the `ProjectSite`. If an access point is calculated with an algorithm as defined in chapter 3, it contains all information received in a WiFi scan. The flag `calculated` indicates whether this is an access point of which the position is estimated by the algorithm. The user interface provides the possibility to add known access points for reference purposes.

The `Location`, which is used by `ProjectSite`, `AccessPoint`, and `WifiScanResult`, contains the x and y coordinates and the timestamp when this location was defined. The `provider` String can be used to identify the source of information. The accuracy was planed in the early prototyping stages, but is not used now.

The `SensorData` class of WiFi Compass is used to hold all measurements for auto calibrating the application for step detection as described in chapter 2.4 Realization in WiFi Compass. It contains up to four float values returned by the sensor, its type and name, and a timestamp of the reception of the sensor values.

The WiFi Compass model is shown in Figure 43: WiFi Compass data model, which contains all model classes as well as their relations.

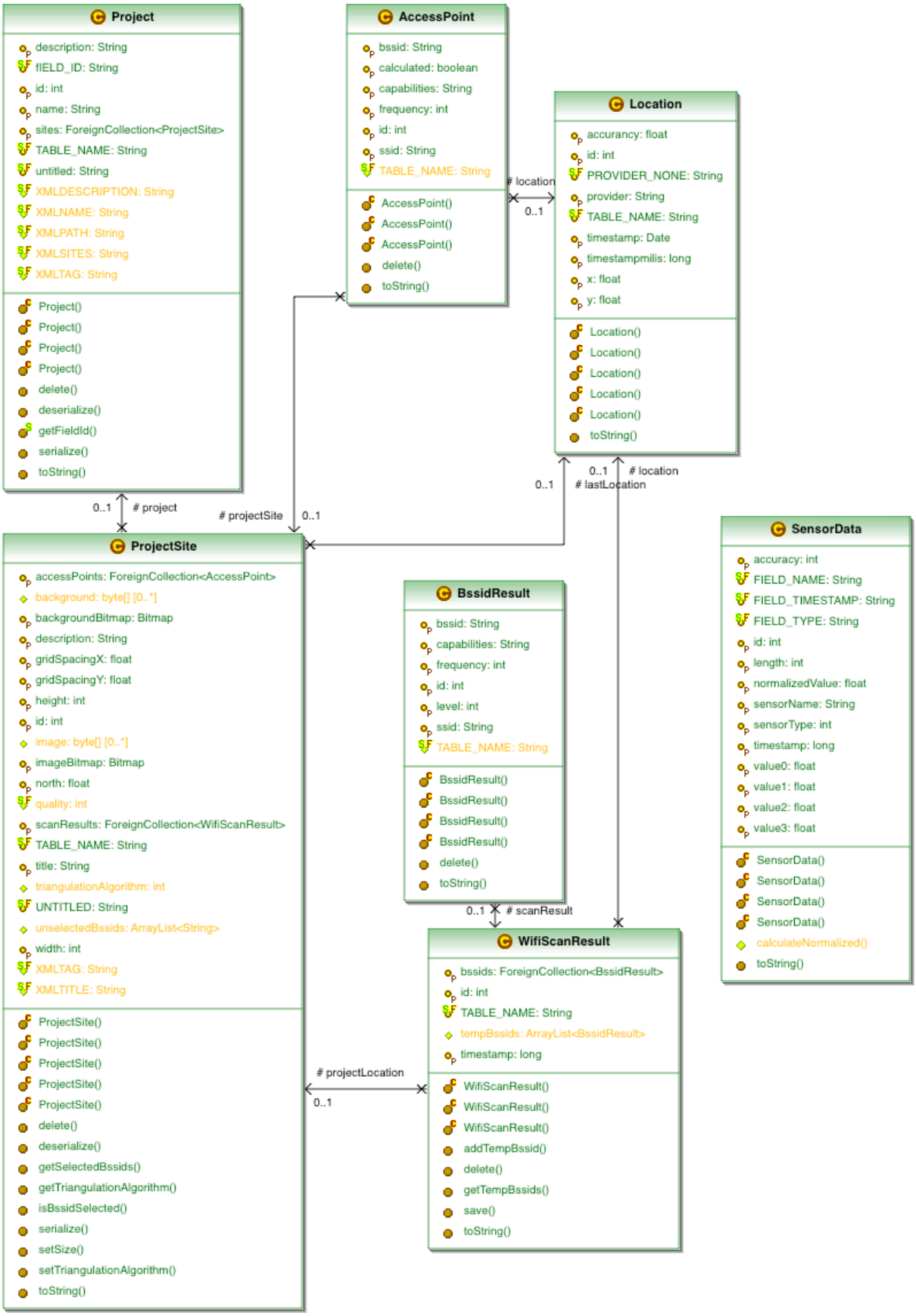

Figure 43: WiFi Compass data model

Persistence

The WiFi Compass model must be persisted, so that the data can be used several times and is available after the application has been paused or stopped. This chapter describes how this is implemented in WiFi Compass.

All data that must be persisted is either saved in an SQLite database or as a Shared Preference. Shared Preferences are used for settings, which are independent from the projects. This includes the calibration values for step detection and the interval of WiFi scans conducted during walking through a building. Shared Preferences are handled by the Android Operating System and can be accessed in different Activities in an application [**30**].

The classes from the model as described in the previous chapter are persisted in an SQLite database. To ease access to the persisted model and remove the need of writing SQL, the authors have decided to use an abstraction layer. In the prototyping phase, different abstraction layer libraries have been tested, but OrmLite[8] has proven as the most suitable one for WiFi Compass. OrmLite is a Framework, which is able to generate Data Defnition Language (DDL) and Data Query Language (DQL) for SQLite and JDBC databases [**31**]. The SQLite part is specially targeted for Android and uses the native Android system calls [**32**].

The `DatabaseHandler` class controls the database access, which is responsible for providing the algorithms to create or update the database. All persisted classes are either found dynamically or defined by hand. The authors chose to define these classes by hand, because they are not subject to change very often. Moreover, upgrading between different versions is also handled manually. So every time the database changes, the tables can either be dropped and recreated, or DDL alter statements are executed to modify the table structure. Beginning from database revision 16, the data is no longer purged. Instead, alter statements are executed. The revision number is an internal number, which is used to identify the state of the database model.

The static class `OpenHelperManager` of OrmLite is used to get an instance of `DatabaseHelper` as show in Listing 13, lines 12 and 13. The `OpenHelperManager` manages the `ConnectionSource` of the database and serializes the access to the database, so that the data stays in a consistent state [**33**]. When a database helper is no longer needed, it can be released by calling `releaseHelper` as shown in Listing 13 line 5.

[8] OrmLite Homepage: http://ormlite.com/

```
1  @Override
2  protected void onDestroy() {
3      super.onDestroy();
4      if (databaseHelper != null) {
5          OpenHelperManager.releaseHelper();
6          databaseHelper = null;
7      }
8  }
9
10 private DBHelper getHelper() {
11     if (databaseHelper == null) {
12         databaseHelper =
13             OpenHelperManager.getHelper(this, DatabaseHelper.class);
14     }
15     return databaseHelper;
16 }
```

Listing 13: OrmLite mainting database access [33]

Instances of objects can be accessed by a `Dao` Class, which is retrieved from the database handler. The `Dao` class provides methods to query for objects, create, update and delete objects. The model class `WifiScanResults` makes use of these functions to save a new object as shown in Listing 14. Line 2 retrieves a `Dao` object to create a `Location` object (line 3). On line 5, the object itself is created. The `Location` is created first, because the `WifiScanResult` directly references the location object. If the location object does not have a unique ID assigned by the database, the reference cannot be saved. Line 11 creates all `BssidResults`, which have been saved only in memory. This operation must be done after the `WifiScanResult` is created because the `BssidResult` holds the reference to the `WifiScanResult`. This is the case because the relationship between `WifiScanResult` and `BssidResult` is a 1-to-n relation.

```
1      public void save(DatabaseHelper databaseHelper) throws SQLException{
2             Dao<Location, Integer> locationDao =
   databaseHelper.getDao(Location.class);
3             locationDao.createIfNotExists(this.getLocation());
4
5             Dao<WifiScanResult, Integer> scanResultDao =
   databaseHelper.getDao(WifiScanResult.class);
6             scanResultDao.createIfNotExists(this);
7
8             Dao<BssidResult, Integer> bssidResultDao =
   databaseHelper.getDao(BssidResult.class);
9
10            for (BssidResult br : this.getTempBssids()) {
11                  bssidResultDao.create(br);
12            }
13
14     }
```

Listing 14: Creating a new WifiScanResult

Creating objects in the database must always be done either with the use of `dao.create` or by first setting the Dao on the object with `modelInstance.setDao` and `modelInstance.create`. This is the case because the object itself does not know which database it belongs to. The relation to the database is created with the `Dao` object. In WiFi compass, create operations are always done by `dao.create`. Modifying objects is sometimes done from the object, because the objects also support deleting their dependent objects to keep the database clean. The model classes overwrite the delete method of `BaseDaoEnabled` to accomplish this task as shown in Listing 15. The `setDao` method is not required to update or delete objects, because when an object is loaded, the Dao is automatically set.

This mechanism can be improved, if all objects use the default database instance. The model classes could implement the create, read, update and delete methods from a database object super class and therefore would remove the requirement of a Dao object. If a different database should be used, providing a name or `ConnectionSource` in the method function calls could do this. This approach is realized in Grails [**34**] and provides a more flexible way to access database objects. However, this is not feasible for WiFi Compass, because Groovy is not supported on Android, which is the basement for Grails. Also, Grails is a platform to build web applications.

```
1      @Override
2      public int delete() throws SQLException {
3             if(lastLocation!=null&&lastLocation.getId()!=0) lastLocation.delete();
4             for(AccessPoint ap: accessPoints){
5                    ap.delete();
6             }
7             for(WifiScanResult sr: scanResults){
8                    sr.delete();
9             }
10
11            return super.delete();
12     }
```

Listing 15: Deleting Project Site, which contains sub objects

4.3 Sample Use Case

In this section, a sample use case of the WiFi Compass Android application is provided. The steps done are

- the sensor calibration,
- the creation of a new project,
- adding a new project site,
- setting a background image (the map),
- setting the scale of the map,
- setting the magnetic north (for step detection),
- the collection of measurement data using step detection,
- the selection of BSSIDs to consider in the calculation,
- and finally the calculation of the access point positions.

The use case is done with a fresh installation of WiFi Compass on a Samsung Galaxy Tab 8.9. When the application is started for the first time, it suggests starting the calibration of the sensors. This is shown in Figure 44.

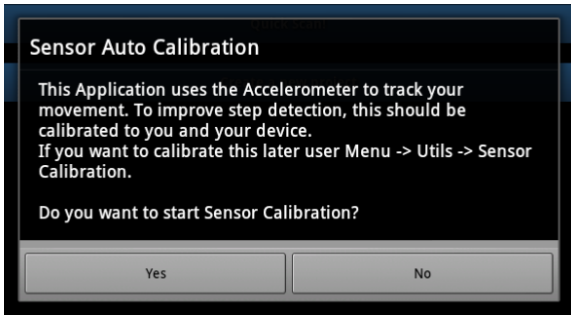

Figure 44: Dialog that suggests the calibration of the sensors when the application is started for the first time

When the sensor calibration activity is started, an explanation dialog shows how to execute the calibration (see Figure 45).

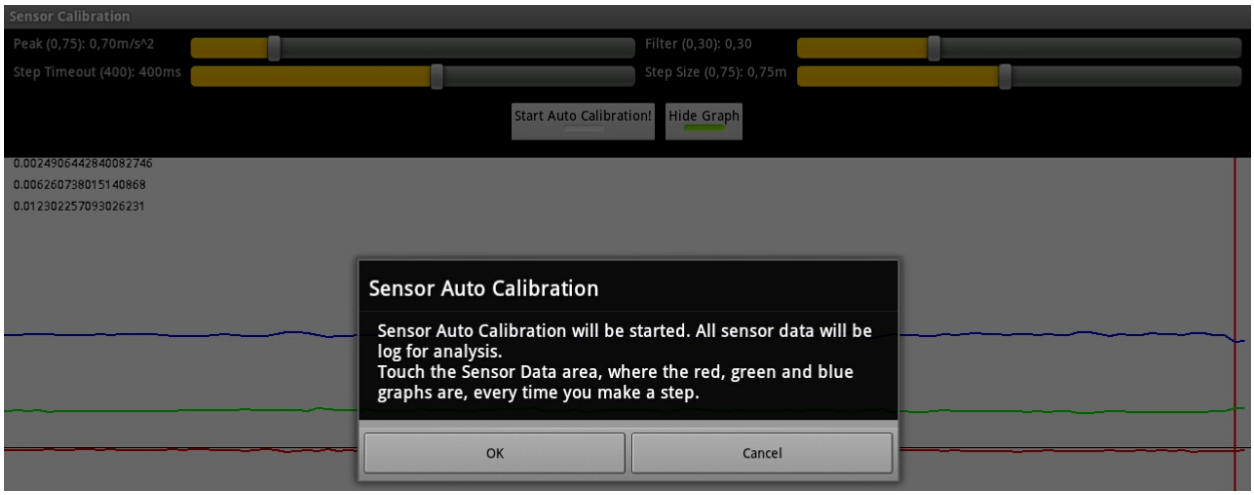

Figure 45: Explanation dialog in the Sensor Calibration activity

The user then taps “Start Auto Calibration” and touch the graph every time they make a step. A sample calibration process is shown in Figure 46.

Figure 46: A sample calibration process

Once approximately 100 steps are done, the user taps “Stop Auto Calibration”. The user is then presented with a dialog where they can enter a tolerance value for the step detection. This means that if the user taps the graph a bit too late or too early when making a step, the tolerance value is the time window within which the software searches for detected steps. This is necessary, because otherwise steps may not be detected correctly (see Figure 47).

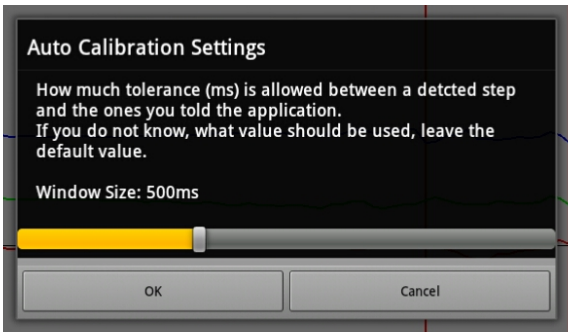

Figure 47: Configuration dialog for the tolerance value in Step Detection

The software then tries to calculate the ideal settings for the given device and the user holding it. As there are many different Android devices on the market, the values may differ from device to device (see Figure 48). For optimal calibration settings of different devices see section 6.2 in the appendix.

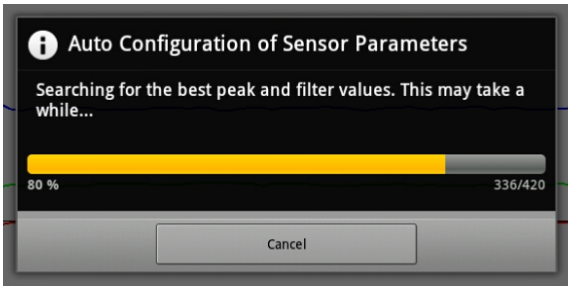

Figure 48: The ideal sensor settings are calculated

Once the calculation is done, the user is presented with the results of the calculation (see Figure 48). These sensor settings are then saved in the application. Note that the sensor calibration settings are not stored in the database as they are device-specific.

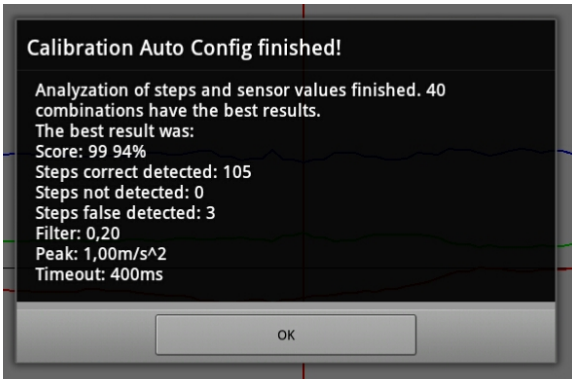

Figure 49: Sensor auto calibration results

Once the calibration is done, the user can create a new project in the home screen (see Figure 50).

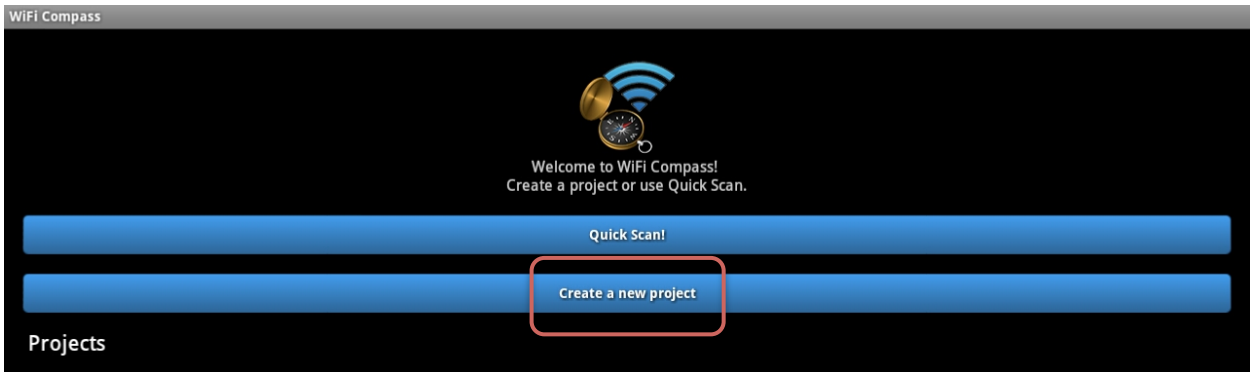

Figure 50: The home screen of WiFi Compass

When the user clicks “Create a new project”, a new activity is started. This is where the user can enter a project name, a description, and create a new survey site (one project may contain several survey sites, such as one site per floor). The activity is shown in Figure 51.

Figure 51: The “Create a new project” activity

When a new survey site is added, the user is asked to enter a title for the site. In this case, the title is “Main floor” (see Figure 52).

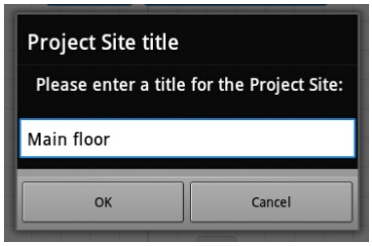

Figure 52: The Project Site name dialog

When the Project Site is opened for the first time, the initial setup dialog is started (see Figure 53). This dialog suggests loading a background image (the map of the floor).

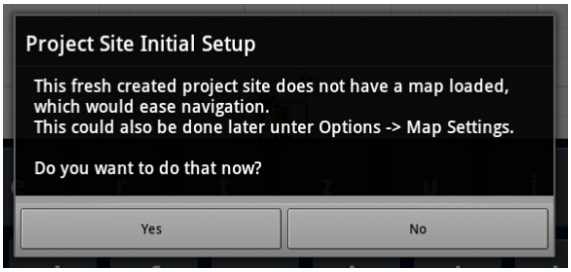

Figure 53: The Project Initial Setup dialog

Figure 54 and Figure 55 show how the background image is set. The file browser (Figure 55) automatically shows only compatible file types.

Figure 54: The Map Image dialog

Figure 55: The selection of an image file

Figure 56 shows the example Project Site with a map image set.

Figure 56: The Project Site activity with a map in the background

In order to use Step Detection, the user must specify the scale of the map. This is necessary so that the application can accordingly move the user icon when a step of, for example, 40 cm into a certain direction is detected. The setting can be found in the menu “Map Settings” (see Figure 57), but is also shown as the next step in the initial configuration (see Figure 58).

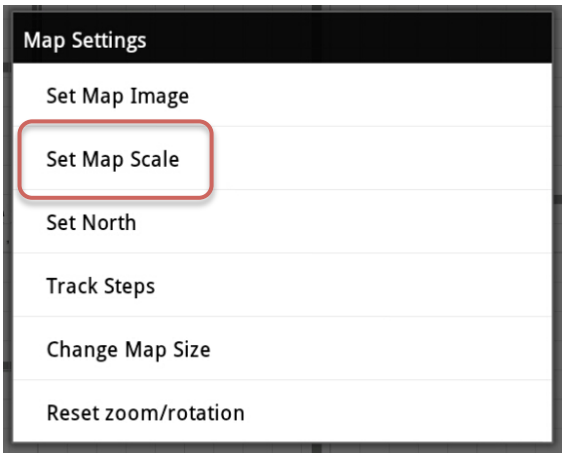

Figure 57: The “Map Settings” menu with the “Set Map Scale” item

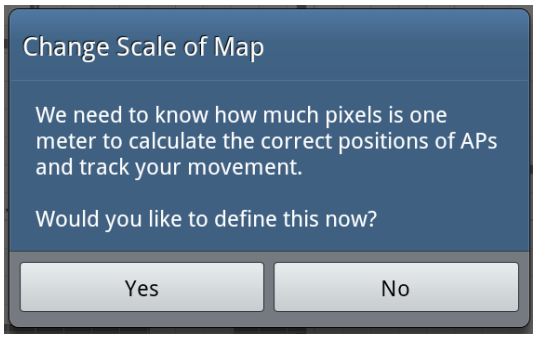

Figure 58: Scale of map auto configuration dialog

When the menu item is tapped or the dialog is confirmed with “Yes”, two red arrows and a red line connecting them appear on the screen like in Figure 59.

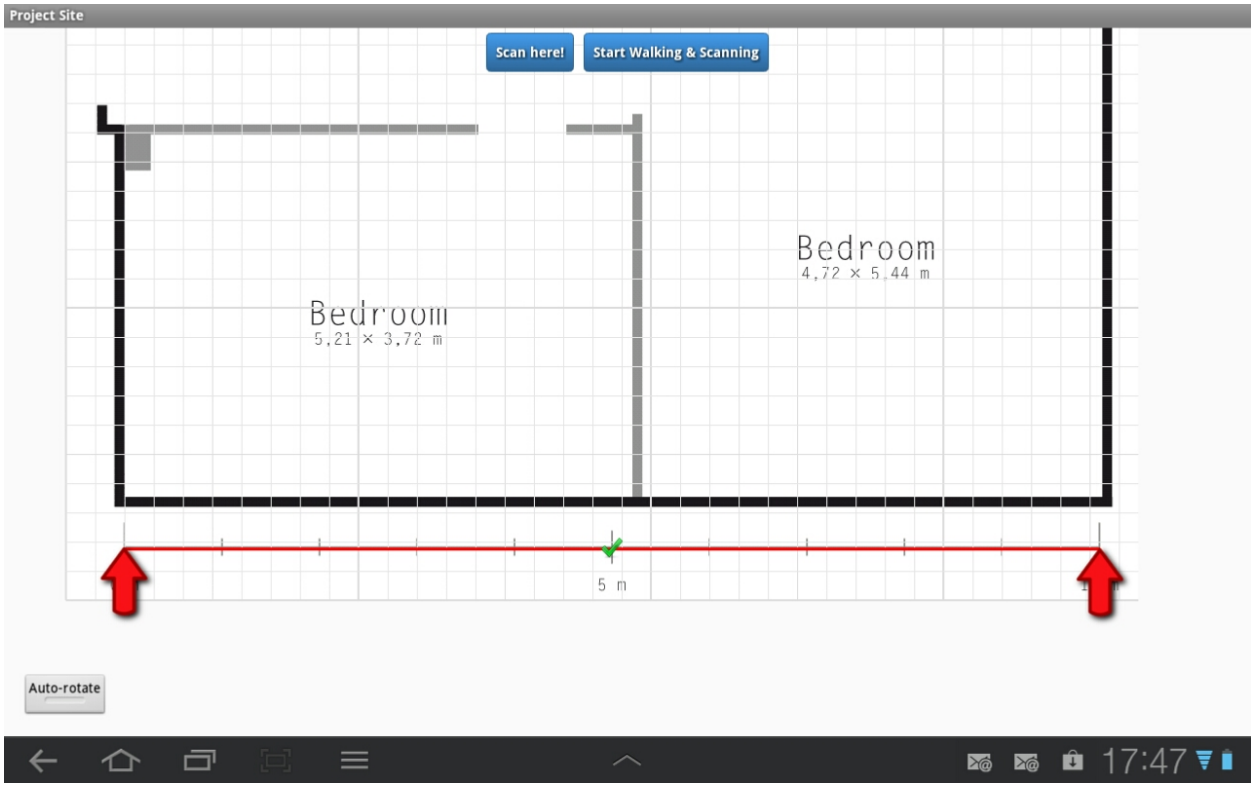

Figure 59: Setting the scale of the map

The user is now asked to move the arrows to two points that they know the distance of. For example, if there is a legend on the map indicating 10 m, the user moves the first arrow to the beginning of the line and the second arrow to the end of the line. When they tap the green tick at the center of the line, a dialog (Figure 60) pops up and asks the user how many meters have been marked. In this case, the legend shows 10 m so “10” is entered.

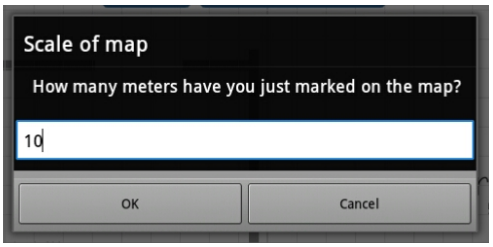

Figure 60: Entering a scale value in meters

The gray grid in the background now changes to an interval of exactly 1 m if the map scale was set correctly (see the background of Figure 63).

The Step Detection algorithm requires one more parameter to be entered: The direction of the magnetic north. This is necessary for the Step Detection algorithm to accordingly recognize the direction towards which the user is heading. The parameter can be set in the “Map Settings” menu (see Figure 61) and is prompted in the initial setup (see Figure 62).

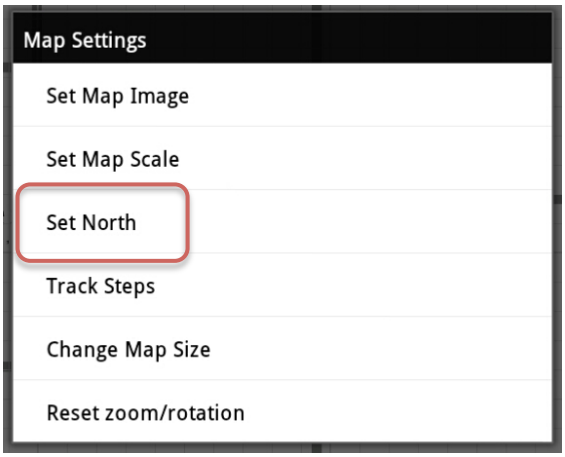

Figure 61: The “Map Settings” menu with the “Set North” item

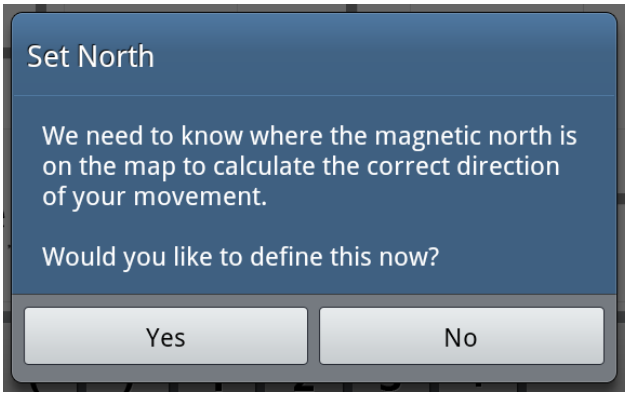

Figure 62: inital setup dialog for magnetic north adjustment

When the menu item is tapped, a draggable dialog like in Figure 63 appears on the map. The user is asked to turn the device until the map rotation on the screen equals the real orientation. The compass at the very top of the dialog shows the current alignment of the device compass. The current angle adjustment is shown below the compass. This angle indicates how much the real magnetic north differs from the map north.

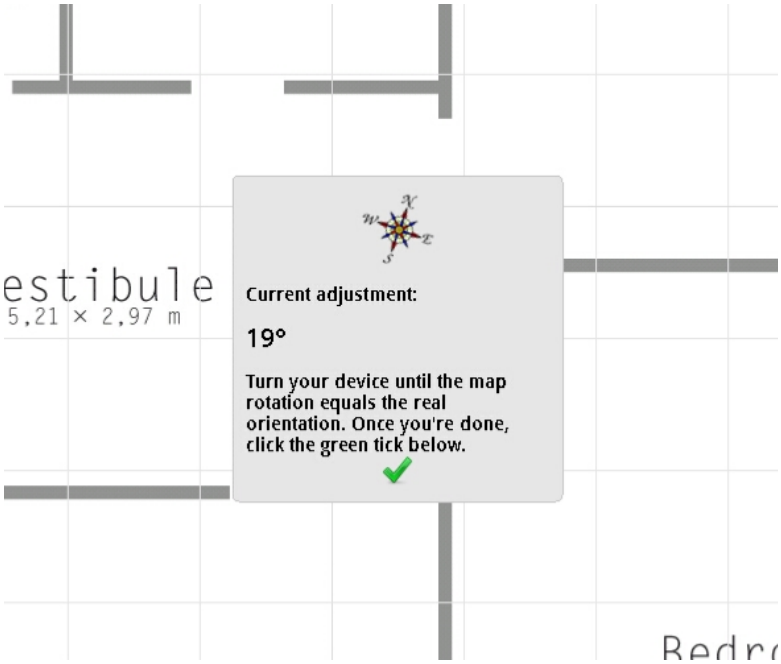

Figure 63: Setting the magnetic north (used for Step Detection and Auto-rotate)

Now that all the necessary settings are done, the collection of the measurement data can be started. The user does so by tapping the “Start Walking & Scanning” button at the top of the screen (see Figure 64).

Figure 64: Collection measurement data

While the user walks, the Step Detection algorithm tracks their position (the red line in Figure 64 shows the walking path of the user). Scanning activities are started in the background while the user walks. As soon as the application gets Wi-Fi scanning results from the operating system, a new measuring point (the blue foot icons in Figure 64) is created at the current position of the user. The found networks are stored in the measuring point.

While the user is walking, the scanning results are held in memory because storing them in the database immediately can be very slow depending on the amount of networks found and the performance of the device. When the user clicks “Stop Walking & Scanning”, the results are stored in the database (see Figure 65).

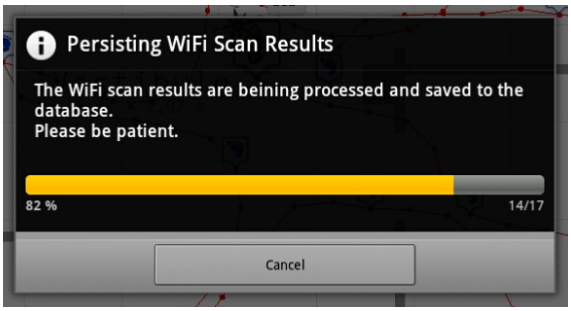

Figure 65: The scan results are stored in the database once the scan is done

Figure 66 shows the entire walking path of the user. Note that measuring points can also be created manually. The user icon (the icon with the yellow man) can be dragged over the map and measurements can be taken by clicking the “Scan here!” button on top of the screen.

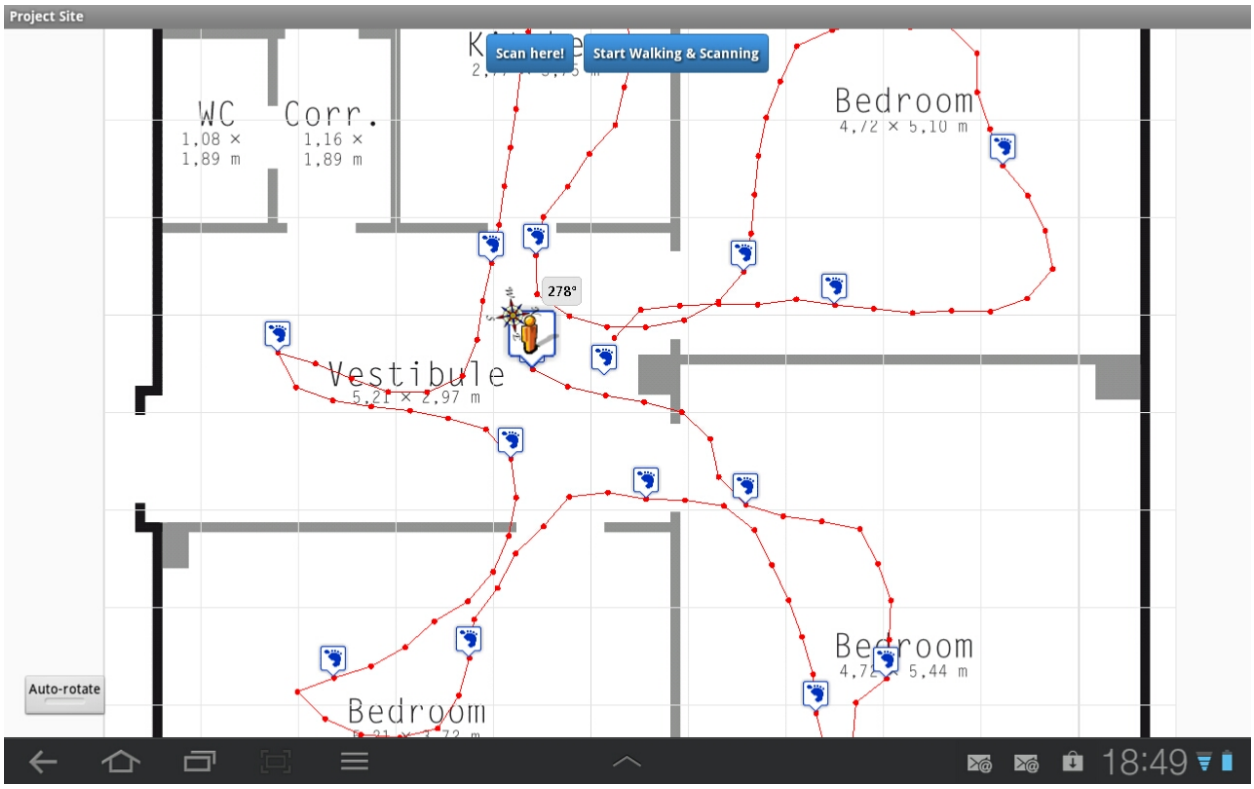

Figure 66: The walking path (red line), the steps (red dots) and the measuring points (blue foot icon)

If the user wants to scan only for certain BSSIDs, they can do so by selecting a set of BSSIDs in the “Select BSSIDs” dialog (see Figure 67 and Figure 68).

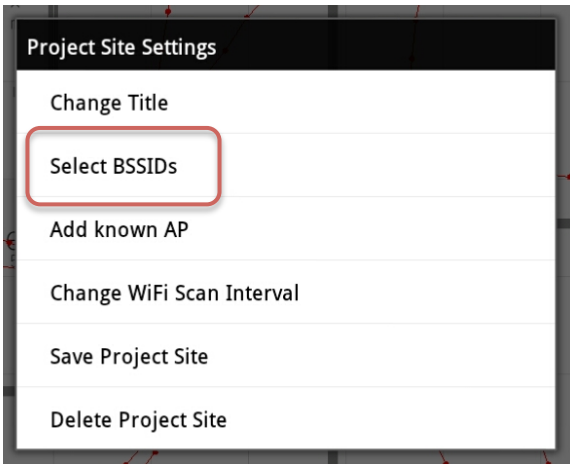

Figure 67: The “Project Site Settings” menu

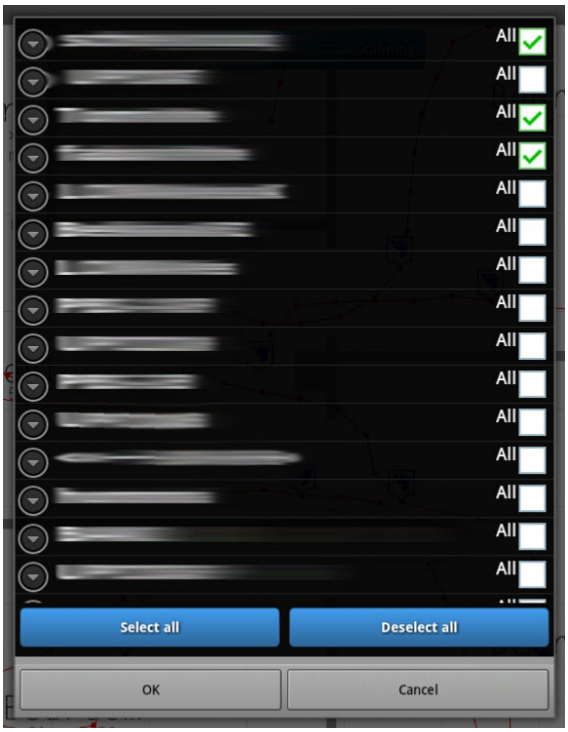

Figure 68: The BSSID selection dialog

Once all the measurements are done and the BSSIDs are selected, the user can calculate the estimated access point positions by clicking “Calculate AP positions” in the main menu. The calculation progress is then shown on the screen (see Figure 69).

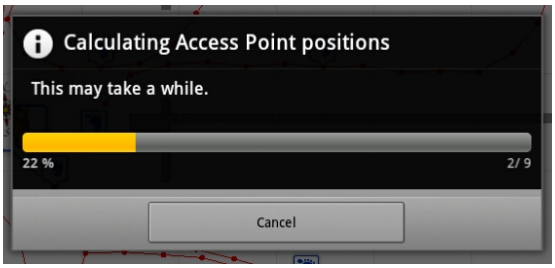

Figure 69: The progress dialog while the access point positions are calculated

The final result is shown in Figure 70.

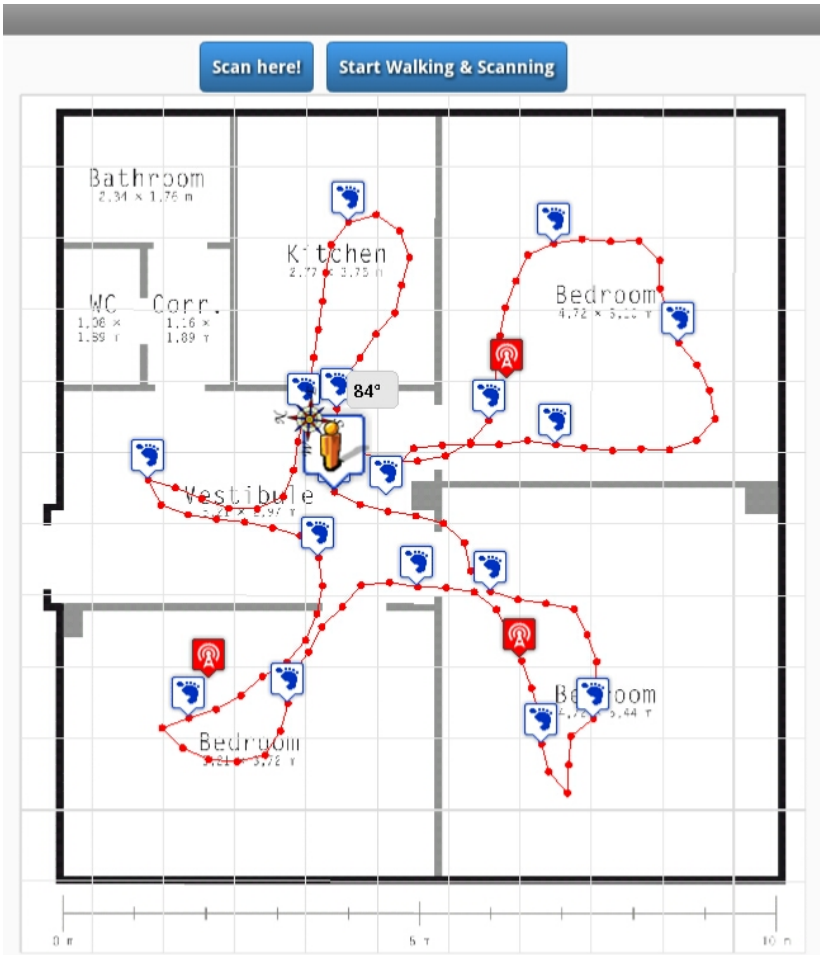

Figure 70: The finished project

The estimated access point positions are shown as small red icons (see Figure 70). Clicking on the access point icon shows the SSID, BSSID, frequency, capabilities and estimated position in coordinates (see Figure 71).

Figure 71: A popup shows details about the access point

4.4 Project Website and Source Code

The project WiFi Compass has been developed as a Google Code Project and is available under http://wificompass.googlecode.com. The project site contains the latest release of the Android application, as well as documentation and an issue tracking system as shown in Figure 72.

Figure 72: Google Code WiFi Compass Project Site

The Android application is licensed under GNU Public License v3[9] and therefore also available as source code. It can be downloaded and modified by anyone, but any modifications must also be published under the GPLv3. To track changes to the source code, a version control system is used. During the developing phase, the VCS has been changed from SVN to Git. The latest source code is available through the source page on the project site: http://code.google.com/p/wificompass/source/checkout.

[9] http://www.gnu.org/licenses/gpl.html

The Google Code issue tracking system helps to track bugs and requests. In order to ease the management of changes in the code, they are often linked to issues in the issue tracking system.

The application is also available in the Google Play store (see Figure 73). It can be freely downloaded on any compatible Android device.

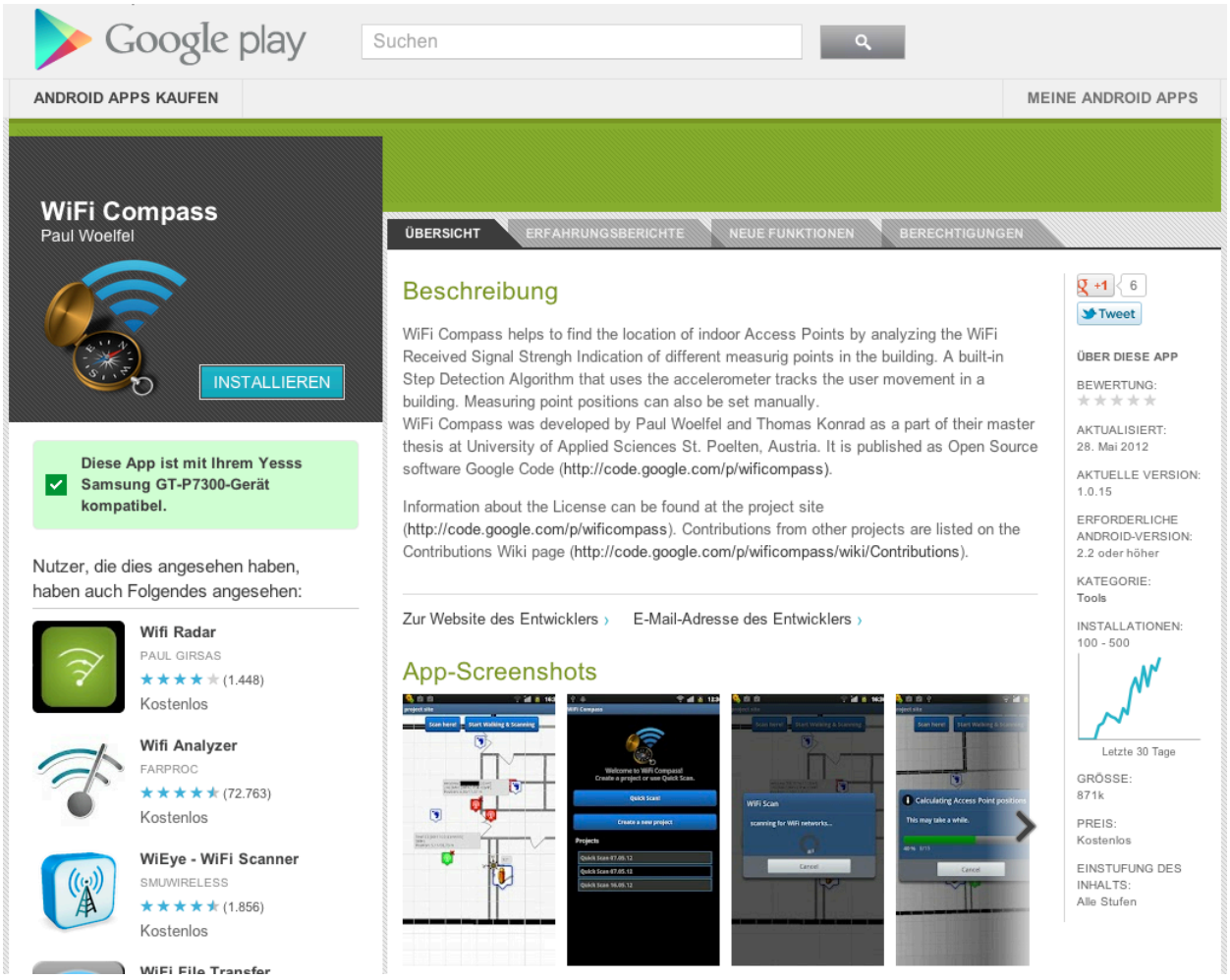

Figure 73: WiFi Compass in the Google Play store

5 Conclusion and Future Work

WiFi Compass has been developed as an Open Source project and is now available for installation on Android devices from Google Play. The project source code is licensed under GPLv3 and available on the Google Code Project website[10]. It is fully functional and provides the possibility to track device movement, scan for WiFi networks, and estimate the position of access points. It can be used with a grid as background in unknown locations or with a map in known locations.

5.1 User Tracking

Several localization and tracking algorithms have been evaluated to see whether or not these can be used as a navigation system for Android devices in an indoor environment. The best-performing algorithm was an Inertial Navigation System, which is based on the accelerometer and on the compass. The algorithm implemented in WiFi Compass proved to be up to 97% accurate on straight walks and is able to track the movement over a five-minute walk. This localization system helps the user to track their position in the building. If the algorithm fails because of magnetic field distortion, the user can adjust the position manually by dragging the icon to the correct position. The initial position is also defined in the same way by a simple drag and drop operation.

Future Work

The tracking algorithm could be improved further, if the gyroscope of the device would be taken into account. With the second sensor, the direction estimation could be made more reliable, because two independent sources are used.

As mentioned in chapter 2.3.2, Chintalapudi et al. propose a system, which solely relies on the calculation of the relative positions of access points and mobile devices to calculate the position of a device. The system is built as a client-server model, but may also be converted to a standalone system, which could run on an Android device. This requires further research, which was not conducted by the authors and may be considered as future work.

5.2 Access Point Trilateration

Concerning the calculation of the access point positions, three algorithms have been evaluated. These are the Weighted Centroid algorithm, the Advanced Trilateration algorithm and the Local Signal Strength Gradient algorithm. The **Weighted Centroid** algorithm works by simply

[10] Source Code available at: http://code.google.com/p/wificompass/source/checkout

calculating the geometric centroid of all measuring points weighting each point's x and y coordinates by the received signal strength. The algorithm is simple and fast, even on mobile Android devices. The **Advanced Trilateration** algorithm is more complex and way slower than Weighted Centroid. It works by calculating the estimated distance of the access point from each measuring point, and taking the circumference of the imagined "circle" around the measuring point as the peak of a Gaussian distribution. This is done for each measuring point, and the results of the Gaussian distribution calculation of every measurement point are then summed up for each point on the map. The point on the map with the highest sum is then the location where the access point is most likely located. The **Local Signal Strength Gradient** algorithm takes each measuring point and calculates the direction towards which the access point is most likely located. This is the direction where the RSSI values of the surrounding measurement points (those which lie within a predefined window size of e.g. 3 meters) increase most. After that, the sum-squared angular error of all "arrows" is calculated for each point on the map. The access point is estimated to be located at the point where the sum-squared angular error is the lowest.

Ground Truth

The algorithms have been tested in a real-world scenario. The environment was an office building of approximately 45 × 20 m in size. 37 unique BSSIDs were on the floor, and signal strength data from a total of 180 measuring points was collected. The survey showed that, in a common scenario, the Weighted Centroid algorithm performed best. It was both the most accurate and by far the fastest of all algorithms. A second survey with the same data was done where only measuring points with a certain distance from the access point were taken into consideration. This was done in order to simulate an environment where the measurement points do not surround the access points. Again, Weighted Centroid performed best, although none of the algorithms got even close to an accurate estimation. Both other algorithms had a significant amount of outliers. Due to these findings, Weighted Centroid has been implemented in WiFi Compass and is used as the default algorithm. However, it is strongly recommended to take the measurements as widespread as possible, because Weighted Centroid can only accurately calculate the positions of access points, which are surrounded by measuring points.

Future Work

The currently used algorithm in WiFi Compass, the Weighted Centroid algorithm, is simple and quite accurate. However, when the measuring points do not surround the access points, the algorithm gives highly inaccurate results. This causes access points, which are in fact far away, to appear within the measuring area. So if there are only a couple of access points within the building, and many outside the building, then the map is "flooded" with wrong results. That is why access points with very low signal strengths could be filtered in order to improve accuracy and usability.

Maybe the Local Signal Strength Gradient algorithm can be improved to solve this problem. It can theoretically give results outside the measuring area, but the test has shown that these

results are very inaccurate. There is possibly a way to optimize the algorithm regarding this challenge.

The Advanced Trilateration algorithm is also very inaccurate when the measuring points do not surround the access points in consideration. Maybe there is a way to improve the algorithm to give more accurate results in these situations.

There may also be a way to combine the algorithms. The Local Signal Strength Gradient algorithm works by calculating estimated directions towards which the access point is located, so this information could probably be used to estimate a rough direction. The Advanced Trilateration algorithm calculates estimated distances, so there is probably a way to combine these two algorithms to get more accurate results.

5.3 The Android Application

The Android application WiFi compass has room for improvements, but the main target to create an application, which is easy to use without deep knowledge of the underlying algorithms, has been fulfilled. The user is guided through the setup of a map to provide all details necessary to calculate access point positions. A multi-touch user interface has been implemented to support natural handling of objects on a map.

6 Appendix

6.1 A: Access Point Trilateration

6.1.1 MATLAB Implementation of the Advanced Trilateration Algorithm

The code snippet of Listing 16 is a MATLAB implementation of the Advanced Trilateration algorithm as described in section 3.3.

```
function [posX, posY] = tri(data)

    % We need to make those global so that the probability function
    % prob(x) has access to them
    global xdata
    global ydata
    global zdata
    global C
    global n

    % Split up the data into x, y, and RSSI (z)
    xdata = data(:,1);
    ydata = data(:,2);
    zdata = data(:,3);

    % These values are added to the estimated C and n. They have proven
    % to give more accurate results than the raw estimation.
    deltaC = 14;
    deltan = 3.2;

    % The Weighted Centroid algorithm roughly estimates the position of
    % the access point
    [wcX, wcY] = weighted_centroid(xdata, ydata, zdata);
    Z = [ones(length(xdata),1)...
         -5*log((wcX - xdata).^2 + (wcY - ydata).^2)];

    % Use the method of least squares to estimate C and n
    a = Z\zdata;
    C = a(1)+deltaC;
    n = a(2)+deltan;

    % Use the optimization tool 'fminunc' to search the local minimum
    % starting from the Weighted Centroid estimation
    x0 = [wcX, wcY];
    options = optimset('Display', 'Off', 'LargeScale','Off');
    [xres, fval] = fminunc(@prob, x0, options);
```

```
37
38      % The result: the point with the highest probability
39      posX = xres(1);
40      posY = xres(2);
41  end
42
43  % The probability function. Returns an inverted probability for the AP
44  % to be located at x (x contains x and y coordinates)
45  function p = prob(x)
46
47      % Gives us access to the measuring data
48      global xdata
49      global ydata
50      global zdata
51      global C
52      global n
53
54      % We start with probability 0 and then add up the probability values
55      % for each measuring point
56      p = 0;
57
58      for i=1:size(xdata)
59          % Get the current measuring point data
60          currx = xdata(i);
61          curry = ydata(i);
62          currz = zdata(i);
63
64          % Calculate estimated circle circumference d
65          d = 10^((C - currz)/(10 * n));
66
67          % Calculate the distance to the circle
68          deltaD = sqrt((currx - x(1))^2 + (curry - x(2))^2) - d;
69
70          % The standard deviation for Gaussian distribution depends
71          % on the estimated distance
72
73          sigma = 0.64 * d + 5;
74          % Calculate the probability
75          pTemp = 1 / sqrt(2 * pi * sigma^2) *...
76                  exp(-(deltaD^2) / (2 * sigma^2));
77
78          % Add the probability value of the current measuring point
79          p = p + pTemp;
80      end
81
82      % We have to invert the value because the optimization tool can only
83      % find local minima, not maxima
```

```
84      p = -p;
85  end
```

Listing 16: A MATLAB implementation of the Advanced Trilateration algorithm

6.1.2 MATLAB Implementation of the Local Signal Strength Gradient Algorithm

Listing 17 shows a MATLAB implementation of the algorithm described in section 3.4.

```
1   function [posX, posY] = lssg(x, y, z)
2
3       % The window size and g are set statically
4       windowSize = 2;
5       g = -0.4;
6
7       arrows = [];
8       currentRow = 1;
9
10      for i=1:size(x)
11          xCenter = x(i, 1);
12          yCenter = y(i, 1);
13
14          % Find all the points that are within the window (default 5 meters)
15          indices = find(x >= xCenter-windowSize &...
16                         x <= xCenter+windowSize &...
17                         y >= yCenter-windowSize &...
18                         y <= yCenter+windowSize);
19
20          % Extract only the indices that are within the window
21          xCurrWindow = x(indices);
22          yCurrWindow = y(indices);
23          zCurrWindow = z(indices);
24
25          areCollinear = collinear2(xCurrWindow, yCurrWindow);
26
27          currSize = size(xCurrWindow);
28          sizex = currSize(1, 1);
29
30          % Only calculate the arrow if the x-y coordinates are not collinear
31          % and if there are more than 3 points, otherwise no plane can be
32          % calculated for the points
33          if ~areCollinear & sizex >= 3
34              % Do the magic
35              % matrix of overdetermined system
36              ACurrWindow = [xCurrWindow yCurrWindow ones(size(xCurrWindow))];
37              cCurrWindow = ACurrWindow\zCurrWindow; % Least squares solution
```

```
38
39              % Store the arrows on matrices
40              arrows(currentRow, 1) = xCenter;
41              arrows(currentRow, 2) = yCenter;
42              arrows(currentRow, 3) = cCurrWindow(1, 1);
43              arrows(currentRow, 4) = cCurrWindow(2, 1);
44              arrows(currentRow, 5) = z(i);
45
46              currentRow = currentRow + 1;
47          end
48      end
49
50      if size(arrows) > 0
51          % Weight the arrows by their RSSI
52          arrows(:,5) = (10.^(arrows(:,5)/20)).^g;
53          sumArrowsRssi = sum(arrows(:,5));
54
55          % Default values for the result
56          posX = 0;
57          posY = 0;
58
59          step = 0.25; % Steps (in meters) by which the area is divided in
60                       % order to calculate the heat map
61          areaFactor = 1; % Factor by which the area is expanded (to search
62                          % for access points outside the given area)
63
64          cenx = (max(x) - min(x)) / 2 + min(x);
65          dx   = abs(max(x) - cenx);
66          ceny = (max(y) - min(y)) / 2 + min(y);
67          dy   = abs(max(y) - ceny);
68
69          % The rectangle where the angular errors are calculated
70          minx = cenx - dx * areaFactor;
71          maxx = cenx + dx * areaFactor;
72          miny = ceny - dy * areaFactor;
73          maxy = ceny + dy * areaFactor;
74
75          counter = 1;
76          xcounter = 1;
77
78          % Calculate the sum-squared angular error for each point on the map
79          for i=minx/step:maxx/step
80              ycounter = 1;
81              currx = i*step;
82              px(xcounter) = currx;
83
84              for j=miny/step:maxy/step
```

```
85
86                  curry = j*step;
87                  py(ycounter) = curry;
88
89                  sump = 0;
90
91                  for k=1:size(arrows(:,1))
92
93                      % Skip if there is no arrow at this point
94                      if ~((arrows(k, 3) == 0) & (arrows(k, 4) == 0))
95                          % Calculate the arrow angle
96                          arrowAngle = wrapTo2Pi(atan2(arrows(k, 4),...
97                                                       arrows(k, 3)));
98
99                          % Calculate the angle of the point towards the
100                             % arrow position
101                             pointAngle = wrapTo2Pi(atan2(curry - arrows(k, 2),...
102                                                          currx - arrows(k, 1)));
103
104                             % See how much the two previously calculated arrows
105                             % differ from each other
106                             angleDifference = abs(arrowAngle - pointAngle);
107
108                             % Always use the minimum difference of the two angles
109                             % (190 degrees would be 170 then)
110                             if angleDifference > pi
111                                 angleDifference = 2 * pi - angleDifference;
112                             end
113
114                             % The probability: the squared angular error
115                             % weighted by RSSI measured
116                             p = angleDifference^2 / (arrows(k, 5) / sumArrowsRssi);
117
118                             % Sum up the errors of all arrows to the current
119                             % point
120                             sump = sump + p;
121
122                         end
123                     end
124
125                     % The probability values are written onto an array
126                     pp(xcounter, ycounter) = sump;
127
128                     counter = counter + 1;
129                     ycounter = ycounter + 1;
130                 end
131                 xcounter = xcounter + 1;
```

```
132             end
133
134             minv = realmax;
135             minx = 1;
136             miny = 1;
137
138             % Search for the minimum error fo all points on the map
139             for i=1:size(px')
140                 for j=1:size(py')
141                     if pp(i, j) < minv
142                         minv = pp(i, j);
143                         minx = px(i);
144                         miny = py(j);
145                     end
146                 end
147             end
148
149             % Return the point where the error is minimum
150             posX = minx;
151             posY = miny;
152         else
153             posX = 0;
154             posY = 0;
155         end
156
157     % Returns whether or not the points are collinear
158     function [areCollinear] = collinear2(x, y)
159         collinear = @(varargin) rank(cat(1,varargin{:}) -
    circshift(cat(1,varargin{:}),1)) == 1;
160
161         points = zeros(0);
162
163         for i=1:size(x)
164             points(i, 1) = x(i, 1);
165             points(i, 2) = y(i, 1);
166         end
167
168         areCollinear = collinear(points);
```

Listing 17: MATLAB implementation of the Local Signal Strength Gradient algorithm

6.2 B: Sensor Calibration Test Results

	Device	User	Steps	Timeout	Window	Score	Pct	Found	Not found	False	Filter	Peak	Def score	Def pct	Def found	Def not found	Def false found
1	Samsung GT1000	1	35	367	500	33	92	34	1	0	0,70	2,60	n/a	n/a	n/a	n/a	n/a
2	Samsung GT1000	1	36	367	405	24	67	36	0	6	0,70	1,30	n/a	n/a	n/a	n/a	n/a
3	Samsung GT1000	1	73	367	500	49	66	66	7	5	0,70	1,80	n/a	n/a	n/a	n/a	n/a
4	Samsung GT1000	1	123	400	500	101	82	119	4	7	0,75	1,70	n/a	n/a	n/a	n/a	n/a
5	ZTE Blade	1	30	403	500	24	80	29	1	2	0,55	1,50	n/a	n/a	n/a	n/a	n/a
6	HTC Desire Z	1	42	400	500	38	90	41	1	1	0,45	2,50	n/a	n/a	n/a	n/a	n/a
7	ZTE Blade	2	35	403	500	13	37	30	5	6	0,45	1,90	n/a	n/a	n/a	n/a	n/a
8	ZTE Blade	2	97	403	500	79	81	94	3	6	0,65	2,50	n/a	n/a	n/a	n/a	n/a
9	Samsung Galaxy S2	3	20	406	500	18	90	20	0	1	0,15	1,60	n/a	n/a	n/a	n/a	n/a
10	Samung Galaxy S2	3	104	406	500	102	98	104	0	1	0,40	1,50	n/a	n/a	n/a	n/a	n/a
11	Samsung Galaxy	4	80	400	500	72	90	79	1	3	0,2	0,9	n/a	n/a	n/a	n/a	n/a
12	Samsung GT1000	5	71	400	500	67	94	71	0	2	0,4	0,7	n/a	n/a	n/a	n/a	n/a
13	Samsung GT1000	6	51	400	500	47	92	50	1	1	0,1	0,3	n/a	n/a	n/a	n/a	n/a
14	Samsung GT1000	6	75	400	500	71	95	75	0	2	0,25	0,6	n/a	n/a	n/a	n/a	n/a
15	Samsung GT1000	7	50	400	500	50	100	50	0	0	0,15	0,4	n/a	n/a	n/a	n/a	n/a
16	Samsung GT1000	1	58	400	500	54	93	58	0	2	0,5	2,1	34	59	58	0	12
17	Samsung GT1000	1	61	400	500	57	93	61	0	2	0,75	1,0	31	51	61	0	15
18	Samsung GT 1000	2	72	400	500	60	83	70	2	4	0,15	1	30	42	68	4	17
19	Samsung GT 1000	1	104	400	500	98	94	104	0	3	0,25	1,2	80	77	104	0	12
20	Samsung GT 1000	1	95	400	500	93	98	95	0	1	0,4	1	83	87	95	0	6
21	Samsung Galaxy 2	8	41	400	500	25	85	41	0	3	0,15	1,40	19	46	41	0	11
22	Samsung Galaxy Tab 8,9“	1	64	400	500	62	97	63	1	0	0,20	1,50	30	47	63	1	13

Table 8: Auto Configuration results

7 Index

7.1 List of Figures

7.2 List of Tables

7.3 List of Formulas

7.4 Source Code Listings

7.5 Bibliography

[1] Anton Hansson and Linus Tufvesson. (2011, Sep.) Using Sensor Equipped Smartphones to Localize WiFi Access Points. Master Thesis, Lund University, Department of Automatic Control. [Online]. http://www.control.lth.se/Publication/5880.html [Accessed 22 Feb. 2012].

[2] Google. (2012, Mar.) Android SensorManager. [Online]. http://developer.android.com/reference/android/hardware/SensorManager.html [Accessed 07 Mar. 2012].

[3] James Diebel. (2006) Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. [Online]. http://ai.stanford.edu/~diebel/attitude/attitude.pdf [Accessed 07 Mar. 2012].

[4] Google. (2012, Mar.) Android Sensors Overview. [Online]. http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords [Accessed 07 Mar. 2012].

[5] Google. (2012, Mar.) Android Motion Sensors. [Online]. http://developer.android.com/guide/topics/sensors/sensors_motion.html [Accessed 07 Mar. 2012].

[6] Yutaka Masumoto, "Global positioning system," U.S. Patent 5210540, Jun 12, 1992. Int. Pat. H04B 7/185; G01S 5/02.

[7] Jörg H. Hahn and Edward D. Powers, "A Report on GPS and Galileo Time Offset Coordination Efforts," in *Proceedings of Frequency Control Symposium, 2007 Joint with the 21st European Frequency and Time Forum. IEEE International*, 2007, pp. 440-445.

[8] Paramvir Bahl and N. Padmanabhan Venkata, "RADAR: An In-Building RF-based User Location and Tracking System," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2000, pp. 775 -784 vol.2.

[9] F. Lassabe, D. Charlet, P. Canalda, P. Chatonnay, and F. Spies, "Friis and iterative trilateration based WiFi devices tracking," in *Proceedings of Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, 2006, p. 4 pp.

[10] Krishna Chintalapudi, Anand Padmanabha Iyer, and Venkata N. Padmanabhan, "Indoor localization without the pain," in *MobiCom '10 Proceedings of the sixteenth annual*

international conference on Mobile computing and networking, Chicago, Illinois, USA, 2010, pp. 173-184.

[11] Timea Bagosi and Zoltan Baruch, "Indoor localization by WiFi," in *Proceedings of Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on*, 2011, pp. 449-452.

[12] Konrad Lorincz and Matt Welsh, "MoteTrack: a robust, decentralized approach to RF-based location tracking," *Personal Ubiquitous Comput.*, vol. 11, no. 6, pp. 489-503, Aug. 2007.

[13] Hung-Huan Liu and Yu-Non Yang, "WiFi-based indoor positioning for multi-floor Environment," in *Proceedings of TENCON 2011 - 2011 IEEE Region 10 Conference*, 2001, pp. 597-601.

[14] Jó Ágila Bitsch, Paul Smith, Nicolai Viol, and Klaus Wehrle. (2011, Sep.) Proceedings of the 2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN), Guimaraes, Portugal. [Online]. http://www.comsys.rwth-aachen.de/fileadmin/papers/2011/2011-IPIN-bitsch-footpath-long.pdf [Accessed 04 Mar. 2012].

[15] Niveditha Sundaram and Parameswaran Ramanathan, "Connectivity Based Location Estimation Scheme for Wireless Ad Hoc Networks," in *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, 2002, pp. 143 - 147 vol.1.

[16] Neal Patwari and Alfred O. III Hero, "Using proximity and quantized RSS for sensor localization in wireless networks," in *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, vol. WSNA '03, San Diego, CA, USA, 2003, pp. 20-29.

[17] Harold Lerman, Neil A. Sanchirico, and John P. Sputz, "Inertial Navigation," US Patent 3,260,485, July 12, 1966.

[18] David H. Titterton and John L. Weston, *Strapdown Inertial Navigation Technology*, 2nd ed. Herts, United Kingdom: The Instititution of Engineering and Technology, 2004.

[19] Guanling Chen and David Kotz, "A Survey of Context-Aware Mobile Computing Research," Department of Computer Science, Dartmouth College, TR2000-381, 2000.

[20] Vasileios Zeimpekis, Giaglis, George M. , and Lekakos, George, "A Taxonomy of Indoor and Outdoor Positioning Techniques for Mobile Location Services," *SIGecom Exch.*, vol. 3, no. 4, pp. 19-27, Dec. 2002.

[21] van Diggelen, Frank and Abraham, Charles, "Indoor GPS Technology," in *Proceedings of CTIA Wireless-Agenda*, Dallas, 2001.

[22] Wang, Jinling, Tsujii, Toshiaki, Rizos, Chris, Dai, Liwen, and Moore, Michael, "GPS AND PSEUDO-SATELLITES INTEGRATION FOR PRECISE POSITIONING," *Geomatics Research Australasia*, no. 74, pp. 103-117, 2000.

[23] Stephan Schuhmann, Klaus Herrmann, Kurt Rothermel, Jan Blumenthal, and Dirk Timmermann. (2008, Jun.) Improved Weighted Centroid Localization in Smart Ubiquitous Environments. [Online]. http://www.imd.uni-rostock.de/veroeff/2008-06_Improved_Weighted_Centroid_Localization.pdf [Accessed 5 Mar. 2012].

[24] Anand Prabhu Subramanian, Pralhad Deshpande, Jie Gao, and Samir R. Das, "Drive-by Localization of Roadside WiFi Networks," in *Proceedings of the 27th Annual IEEE Conference on Computer Communications (INFOCOM'08)*, 2008, Computer Science Department, Stony Brook University, Stony Brook, NY.

[25] Dongsu Han, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, and Srinivasan Seshan, "Access Point Localization Using Local Signal Strength Gradient," Carnegie Mellon University; Intel Research Pittsburgh, Pittsburgh, USA, Studie 2009.

[26] Google Inc. (2012, May) ScanResult | Android Developers. [Online]. http://developer.android.com/reference/android/net/wifi/ScanResult.html [Accessed 11 May 2012].

[27] Jürgen Dankert and Helga Dankert. (2010) Überbestimmte Gleichungssysteme. [Online]. http://www.tm-mathe.de/Themen/html/uberbestimmte_gleichungssystem.html [Accessed 14 May 2012].

[28] Luke Hutchinson. (2012, Apr.) android-multitouch-controller - Simple multitouch pinch-zoom library for Android - Google Project Hosting. [Online]. http://code.google.com/p/android-multitouch-controller/ [Accessed 23 May 2012].

[29] Google Inc. (2012, May) Android Supported Media Formats | Android Developers. [Online]. http://developer.android.com/guide/appendix/media-formats.html [Accessed 20 May 2012].

[30] Google Inc. (2012, May) Data Storage | Android Developers. [Online]. http://developer.android.com/guide/topics/data/data-storage.html [Accessed 20 May 2012].

[31] Gray Watson. OrmLite - Lightweight Object Relational Mapping (ORM) Java Package. [Online]. http://ormlite.com/ [Accessed 20 May 2012].

[32] Gray Watson. OrmLite - Lightweight Java ORM Supports Android and SQLite. [Online]. http://ormlite.com/sqlite_java_android_orm.shtml [Accessed 20 May 2012].

[33] Gray Watson. ORMLite Package: 4. Using With Android. [Online]. http://ormlite.com/javadoc/ormlite-core/doc-files/ormlite_4.html [Accessed 20 May 2012].

[34] SpringSource VMware Inc. Grails Object Relational Mapping (GORM). [Online]. http://grails.org/doc/latest/guide/GORM.html [Accessed 20 May 2012].

[35] Kuo-Fong Kao, I-En Liao, and Jia-Siang Lyu, "An indoor location-based service using access points as signal strength data collectors," Dept. of Inf. Networking Technol., Hsiuping Inst. of Technol., Dali, Taiwan, Dali, Taiwan, 978-1-4244-5865-3, 2010.

[36] Andreas Haeberlen et al., "Practical Robust Localization over Large-Scale 802.11 Wireless Networks," in *MobiCom '04 Proceedings of the 10th annual international conference on Mobile computing and networking* , Philadelphia, PA, USA, 2004, pp. 70-84.

[37] Torben Schüler. (2007, Jun.) Indoor Positionierung mit GPS und alternative Verfahren. [Online]. http://www.nexus.uni-stuttgart.de/de/aktuelles/ereignisse/RingvorlesungSS07/Presentation_Schueler.pdf [Accessed 20 Feb. 2012].