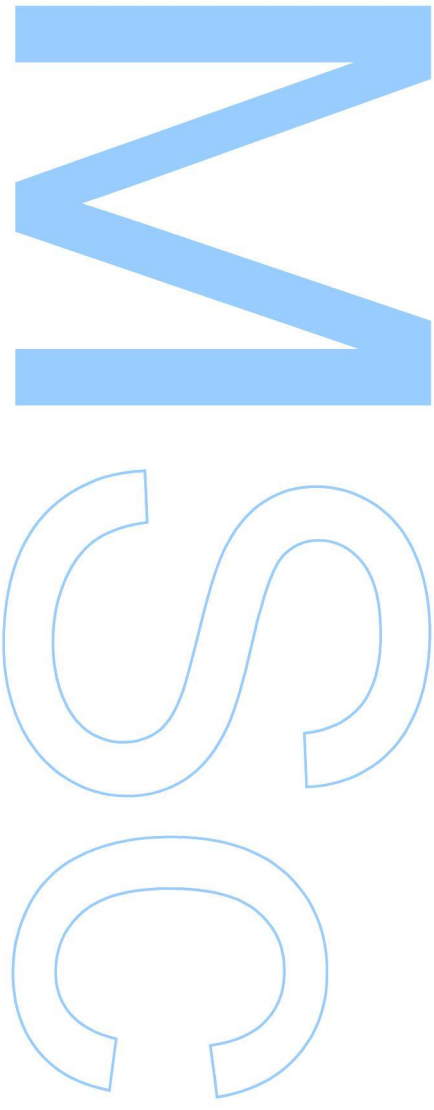
Sensor Fusion for Indoor Localization

Tiago Lopes de Melo
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências de Computadores
2022

Orientador
Sérgio Armindo Lopes Crisóstomo, Professor Auxiliar
Faculdade de Ciências da Universidade do Porto

Coorientador
Rui Pedro de Magalhães Claro Prior, Professor Auxiliar
Faculdade de Ciências da Universidade do Porto

Coorientador
Eduardo Resende Brandão Marques, Professor Auxiliar
Faculdade de Ciências da Universidade do Porto



Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Declaração de Honra

Eu, Tiago Lopes de Melo, inscrito no Mestrado em Engenharia de Redes e Sistemas Informáticos da Faculdade de Ciências da Universidade do Porto declaro, nos termos do disposto na alínea a) do artigo 14.º do Código Ético de Conduta Académica da U.Porto, que o conteúdo da presente dissertação reflete as perspetivas, o trabalho de investigação e as minhas interpretações no momento da sua entrega.

Ao entregar esta dissertação, declaro, ainda, que a mesma é resultado do meu próprio trabalho de investigação e contém contributos que não foram utilizados previamente noutros trabalhos apresentados a esta ou outra instituição.

Mais declaro que todas as referências a outros autores respeitam escrupulosamente as regras da atribuição, encontrando-se devidamente citadas no corpo do texto e identificadas na secção de referências bibliográficas. Não são divulgados na presente dissertação quaisquer conteúdos cuja reprodução esteja vedada por direitos de autor. Tenho consciência de que a prática de plágio e auto-plágio constitui um ilícito académico.

Tiago Lopes de Melo
Porto, 30/09/2022

Acknowledgements

I want to start by thanking my advisors, Professor Sérgio Crisóstomo, Professor Rui Prior and Professor Eduardo Marques for giving me the opportunity to pursue this topic and for their continued assistance.

This work was supported by the Augmanity project / POCI-01-0145-FEDER-046103 (PPS3). Additionally, I would like to thank Instituto de Telecomunicações (UIDB/50008/2020) and CRACS/INESC-TEC for hosting this work. I would also like to thank Faculdade de Ciências da Universidade do Porto for providing me with the education necessary to not only complete this work but also to pursue my career.

I want to thank my friends for their continuous interest in my success and for inspiring me to be better. Most importantly, I am deeply grateful to my parents for their unconditional support throughout my life. Their steady encouragement helped me persist through all hardships. Finally, this dissertation is dedicated to the memory of my grandfather, who would be elated to see this work completed.

Abstract

Localization systems are growing in applicability through the increased presence of services like navigation systems, personnel and asset tracking and applications with location-aware behaviour. In indoor spaces, widespread outdoor localization options like the *Global Positioning System* (GPS) are often unavailable. As such, there is a challenge in how to seamlessly provide accurate positioning in indoor environments.

Prevalent localization solutions for indoor environments leverage the ubiquitous nature of wireless sensor networks, which is supported by the emergence of embedded sensors in network-enabled equipment like mobile devices. These solutions typically enable localization by using network signals to estimate distances and angles between devices or by describing their movement through a combination of inertial sensor measurements. These methods can potentially provide accurate results, but are often challenged by common problems in indoor scenarios, for instance signal interference, multi-path propagation or the accumulation of errors over time as in the case of dead reckoning algorithms. To mitigate these problems, *Sensor Fusion* algorithms use data from heterogeneous sensors in order to improve location accuracy, compared to algorithms that feed on just one type of sensor measurement.

In this dissertation, we explore localization methods for smartphone users. Namely, we make use of *Android* devices to collect *Wi-Fi RTT* ranging and *inertial motion sensor* data. Wi-Fi RTT positioning is performed through the *multilateration* of reported distances between Android devices and Wi-Fi RTT access points. Inertial sensors instead enable relative positioning of devices by characterizing their movement over time through *Pedestrian Dead Reckoning*. Our main contributions involve algorithmic improvements to these two positioning methods, first by applying data smoothing techniques to the Wi-Fi RTT localization process and then by combining the two processes through a sensor fusion algorithm that uses a *Kalman Filter* with weighted state updates.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The benefits brought by localization systems are becoming increasingly apparent in our routine. *Location Based Services* are growing wider in use as they provide solutions to needs that are progressively more commonplace such as navigation, personnel and asset tracking and location-aware behaviour in applications for leisure and business alike [17]. However, a challenge that is still current is how seamless positioning between outdoor and indoor environments can be provided with an acceptable level of accuracy [18].

1.1 Motivation

Global Navigation Satellite Systems (GNSS) are presently the most prominent localization solutions, of which the most popular example is the *Global Positioning System* (GPS). Through constellations of satellites, they transmit signals from space in order to carry information that enables receivers to determine their own location. In indoor environments, GNSS are often unable to guarantee full coverage [33] due to signal strength. Even when available, the accuracy of the estimated positions tends to be inadequate for the scale of an indoor space. As such, a demand exists for dedicated indoor localization systems.

Meanwhile, the *Internet of Things* (IoT) has developed into an indispensable field in the industry, having propagated the ability for everyday objects to connect to wireless networks. Through *embedded systems*, these devices can now communicate all sorts of data captured through a variety of sensors. Due to IoT's frequent application in indoor scenarios, research into enabling localization through wireless sensor networks has seen substantial advancements [24].

Existing technologies have been expanded and repurposed in order to provide localization-enabling data. Initially, positioning algorithms used *Received Signal Strength* (RSS) data collected from network specifications like *Wi-Fi* and *Bluetooth*. Geometry-based algorithms such as multilateration used distances between devices estimated by using RSS [16] while fingerprinting methods directly used sampled RSS data to train machine learning models [32]

[34]. Later revisions to the *IEEE 802.11* standards introduced the *Wi-Fi RTT* feature, which brought an alternative approach to distance estimation by using the measured time that network packets take to travel between devices [10]. This *Time of Flight* (ToF) measurement technique was enhanced with the appearance of radio technologies like *Ultra-wideband* (UWB) [48] [49], which tends to offer greater precision while functioning at shorter ranges and being generally more demanding when it comes to infrastructure and installation. In the meantime, Bluetooth 5.1 also introduced the *Direction Finding* [41] feature that allows for the measurement of angles between devices, which reopened the possibility of high precision localization through other geometry-based solutions like triangulation [37].

Alternatively, a different indoor localization paradigm uses inertial data [30]. The presence of embedded *inertial motion sensors* have also been steadily expanding, especially in IoT and mobile technologies. Inertial sensor readings can be used to obtain data about the motion of an object. For example, acceleration data allows for the calculation of velocity and orientation can be retrieved by measuring the magnetic field or angular rate. By combining this information, systems can characterize how the object moved and estimate its position relative to its previous position [31].

Given this diversity of indoor localization approaches, our focus is to study positioning solutions for mobile devices. Smartphones are our localization tag of choice due to their popularity among users and their relatively advanced integration with ranging and motion sensor technologies. Namely, *Android* devices have been expanding their support for Wi-Fi RTT [13] and possess an expansive motion sensor suite [1], each one with a useful API for developers.

1.2 Objectives

The main goal of this dissertation is to research the state of the art in indoor localization in order to identify important properties in localization-enabling technologies for Android smartphones. Then, we aim to implement methods that leverage the strengths of our chosen technologies while mitigating their weaknesses. For that purpose, we see potential in the fusion of different types of localization-enabling data.

The aforementioned concepts can be divided and structured into the following objectives:

1. Research the state of the art on indoor localization technologies and methods.
2. Implement a system for collecting localization-enabling data from different technologies available in Android devices.
3. Implement algorithms that produce base positioning estimates through the collected data.
4. Develop techniques to maximize accuracy in the implemented methods, with focus on mitigating the weaknesses of each technology and the fusion of data from multiple sources.

1.3 Outline

This dissertation is divided into six different chapters. The beginning of each chapter presents a brief introduction of its contents that provides context regarding the chapter's overall structure and motivation. In this section, we summarily outline this information.

1. **Introduction** - The motivation, objectives and contributions of this work are introduced.
2. **Background** - The description of concepts necessary throughout this dissertation together with additional information about comparable methods, techniques and technologies gathered through literature review.

   This chapter divides the relevant indoor localization knowledge into three components. First, how indoor localization estimates are produced through information obtained through wireless networks. Second, how indoor positioning can be done through inertial motion sensor data. Finally, discussion regarding challenges in indoor localization and techniques meant to improve positioning performance.
3. **Algorithm Design for Indoor Localization** - The description of our research algorithms for indoor localization with Wi-Fi RTT and Android motion sensors. Additionally, we introduce our proposed sensor fusion method for Wi-Fi RTT and inertial sensor data.
4. **Implementation** - Notes on the implementation of the software necessary to develop and test our solutions. We describe each of the associated technology's properties, summarize the process of using their API and discuss development challenges.
5. **Evaluation** - Evaluation of the implemented localization methods. We begin by discussing our methodology for testing and quality assessment of our solutions. We then discuss the results and highlight any strengths and weaknesses that were identified.
6. **Conclusion** - Our remarks about the developed methods and thoughts regarding future work.

Chapter 2

Background

In this chapter, we discuss the background and state of the art of indoor localization. We divide this chapter into sections about ranging-based methods, sensor-based methods and localization improvements.

The first two sections feature methods and technologies used to produce indoor localization estimates. In the third and last section, we discuss problems that can make indoor localization estimates worse and describe methods that can be used to improve them.

2.1 Localization through ranging

Indoor localization can be performed by using real-time data to estimate the distance or angle of a target entity to points of reference. Target entities are commonly referred to as *tags*. Reference points are usually referred to *beacons*, *access points* or *anchors*, depending on the context of the associated technology.

Beacons are devices (such as routers) capable of communicating with other devices in a network. This feature is used to continuously send data that a device can use to estimate the range between itself and the beacon. Beacons must also make their position in a space known, either through previously established convention or constantly advertising this information within the network.

If at a certain point in time enough information is known about anchor positions and the range of these anchors to a tag, then the location for that tag can be estimated.

The following sections detail what methods are used to estimate positions through ranging, exactly how ranges between devices are calculated and the prevalent technologies that use this approach.

2.1.1 Methods

Methods of localization through ranging use distance-based and angle-based information to estimates positions. Specifically, in this section we focus on how methods can use this information to calculate positions through trigonometry and train models that provide estimates through fingerprinting and machine learning.

2.1.1.1 Multilateration

Multilateration uses trigonometry to calculate positions of a tag from its distance to several anchors. Figure 2.1 shows a visual intuition of the multilateration process, where each reported distance constitutes the radius of a circle centered on the corresponding reporting anchor. By intersecting enough circles, we can narrow down the position of the tag. There are various ways to estimate the anchors' set of reported distances, as we further discuss in 2.1.2.

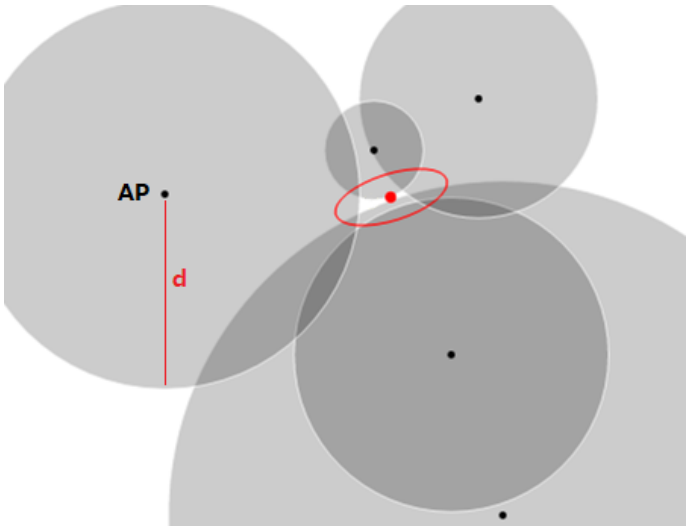


Figure 2.1: Example of how multilateration can provide multilateration algorithms. d represents a given anchor's reported distance to the tag.

Least Squares is one of the most relevant approaches to multilateration. It is an optimization technique that, given a set of equations, tries to minimize the sum of the square of its residuals. The initial formulation for a three-dimensional least-squares localization problem is the equation that defines the distance between a tag and a specific anchor:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = d_i^2 \tag{2.1}$$

Where d_i denotes the ith distance between the tag and an access point with coordinates (x_i, y_i, z_i). The tag's position is described by (x, y, z), which are the variables we are trying to calculate.

Solving a set of non-linear least-squares equations is complex. Linearizing [36] these equations becomes possible if we choose an anchor of reference r. Linearization can then be achieved by subtracting all other equations to the equation of r:

$$\Big[(x-x_r)^2+(y-y_r)^2+(z-z_r)^2\Big]-\Big[(x-x_i)^2+(y-y_i)^2+(z-z_i)^2\Big]=d_r^2-d_i^2 \tag{2.2}$$

By squaring each expression in parenthesis and solving these parenthesis we get:

$$\begin{aligned} x^2-2xx_r+x_r^2+y^2-2yy_r+y_r^2+z^2-2zz_r+z_r^2 \\ -x^2-2xx_i+x_i^2+y^2-2yy_i+y_i^2+z^2-2zz_i+z_i^2 \\ =d_r^2-d_i^2 \end{aligned} \tag{2.3}$$

At this point, it is possible to simplify the equation in order to remove the unknown coordinate values (x, y, z):

$$-2xx_r-2yy_r-2zz_i+2xx_i+2yy_i+2zz_r+x_r^2+y_r^2+z_r^2-x_i^2-y_i^2-z_i^2=d_r^2-d_i^2 \tag{2.4}$$

Which can be rewritten as:

$$2x(x_i-x_r)-2y(y_i-x_y)-2z(z_i-z_r)=d_r^2-d_i^2-x_r^2-y_r^2-z_r^2+x_i^2+y_i^2+z_i^2 \tag{2.5}$$

For two-dimensional positioning, information about at least 3 anchors is necessary. This requirement goes up to 4 anchors in a three-dimensional system. Then, for example, with four anchors we can replicate Equation 2.5 for anchors of index $k=2,3,4$, with the index of our reference anchor being $r=1$.

$$2\underbrace{\begin{bmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \\ x_4-x_1 & y_4-y_1 & z_4-z_1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} d_1^2-d_2^2-x_1^2-y_1^2-z_1^2+x_2^2+y_2^2+z_2^2 \\ d_1^2-d_3^2-x_1^2-y_1^2-z_1^2+x_3^2+y_3^2+z_3^2 \\ d_1^2-d_4^2-x_1^2-y_1^2-z_1^2+x_4^2+y_4^2+z_4^2 \end{bmatrix}}_{\mathbf{b}}$$

Finally, we solve this using linear algebra:

$$\mathbf{x}=\mathbf{A^{-1}b} \tag{2.6}$$

$$\mathbf{x}=(\mathbf{A^T A})^{-1}\mathbf{A^T b} \tag{2.7}$$

2.1.1.2 Fingerprinting

Fingerprinting methods sample localization-enabling information in order to create fingerprint maps. Fingerprint maps are built by dividing a space into a grid and collecting sample data for

each grid point. An unique fingerprint is then calculated using each grid point's collected samples. These data sets are often then used to train a model in order to use machine learning techniques.

The *K-Nearest Neighbours* (KNN) [32] [16] learning method is one of the simplest approaches to estimating positions through fingerprinting. This method calculates a distance metric from the online localization-enabling information sample to each reference grid point. The method then estimates that the tag is located at the grid point with the minimum distance metric which, for example, can be defined as the following:

$$D_n = |\rho_n - \delta| = \sqrt{\sum_{n=1}^{N} (\rho_n - \delta)^2} \tag{2.8}$$

Where vector $\sigma_n = \{\upsilon_1, \upsilon_2, \upsilon_3, ..., \upsilon_k\}$ represents the fingerprint, composed of multiple localization-enabling υ information from anchors $i = 1, 2, 3, ...k$ at a specific grid point $n = 1, 2, 3, ..., N$. $\rho = \{\sigma_1, \sigma_2, \sigma_3, ...\sigma_N\}$ is the collection of fingerprints from all grid points. $\delta = \{\delta_1, \delta_2, \delta_3, ...\delta_k\}$ is the real-time sampled information from the tag to each anchor.

Using the resulting matrix D_n of distance metrics the tag location is then estimated as the minimum value of D_n.

Other relevant machine learning methods include *Random Forest Regression* [40] [29], which during a training phase uses the data set to construct several decision trees. During prediction, the method makes all the trees predict the value for the data point. Then, the estimated grid position could either be the class with most votes across all trees or a mean value of the classes. Alternatively, *Neural Networks* [34] can also be modeled to instead determine the probabilities of the tag being at each of the grid's points of reference.

Fingerprinting is useful because it can be used with values (like RSSI) that can't be directly used to infer distance. Machine learning methods also don't require any specific number of anchors to function.

Fingerprint maps, however, are specific to a certain space's configuration. They are hardly exchangeable between different places. Furthermore, these methods are especially susceptible to become unreliable if there are changes to a place's layout. Changes can be as small and common as the presence of more or less people at a point in time. To maintain reliability and accuracy in these occasions, it is advisable to rebuild the map which requires a whole re-sampling of the data.

2.1.1.3 Triangulation

Triangulation is the process of estimating positions by measuring the angles between a tag and known anchors. There is further discussion on how to obtain angle measurement data in Section 2.1.2.

As illustrated in Figure 2.2, the intuition behind this method is forming a triangle from the unknown tag to known anchors. Finding the unknown position then becomes possible through

basic trigonometry.

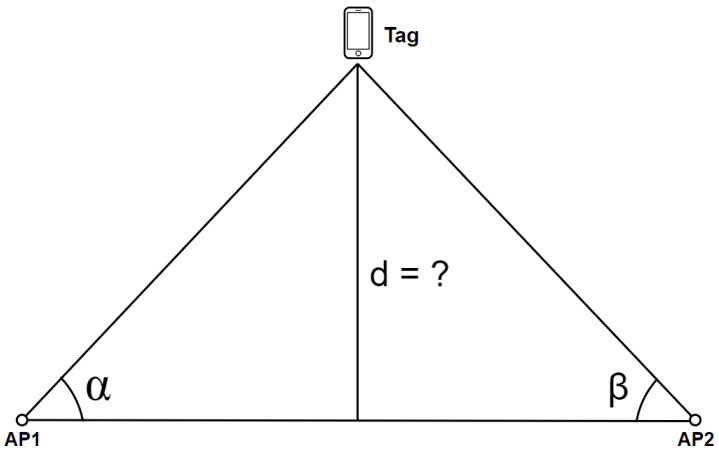


Figure 2.2: Triangulation set-up in a 2-D plane.

If $d_{AP1,AP2}$ is the known distance between AP1 and AP2, α is the angle measured from AP1 to the tag, β is the angle measured from AP2 to the tag and $d_?$ is the unknown distance illustrated in 2.2, then we have:

$$d_{AP1,AP2} = \frac{d_?}{\tan\alpha} + \frac{d_?}{\tan\beta} \tag{2.9}$$

Using trigonometric equivalences, we can transform this equation to find $d_?$:

$$d_? = d_{AP1,AP2}\ \frac{\sin\alpha \sin\beta}{\sin\alpha + \beta} \tag{2.10}$$

Localization through triangulation is proven to be a technique whose algorithms are often associated with limitations [38] such as having blind spots, requiring a particular ordering of beacons and not working for the entirety of a plane depending on anchor layout.

2.1.1.4 Proximity

Proximity methods always attempt to estimate the position of the anchor that's closest to the tag. The criteria to determine this beacon depends on the data measurement technique that is being used. For instance, the best RSSI or the shortest distance estimated by Time of Flight or Round Trip Time measurements could be used for this process. The coordinates that are estimated are the exact coordinates of the chosen anchor, which necessarily means that this method is only appropriate if we only need looser approximates instead of accurate positions.

Alternatively, proximity estimation could also be used obtain high accuracy positioning in a selected set of spots. Using technologies like *Radio-frequency identification* (RFID) or any kind

of *Near-Field Communications* (NFC), we can know that at a certain point in time a tag is at an exact position due to the fact that these readers can only read the tag when it is in very close proximity. The trade-off associated with this approach is that it is highly impractical to perform real-time localization, since it specifically requires readers to be spread at specific points.

2.1.2 Data Measurement Techniques

Technologies that perform ranging use a diversified number of techniques to calculate distance ranges from each beacon to a target device. In this subsection, we list these techniques, analyze how they are calculated and discuss the strengths and weaknesses associated to them.

2.1.2.1 RSS

Received Signal Strength (RSS) measures the amount of power in a received radio signal. Measuring the signal strength can allow us to infer how far a receiver is from a sender because closer they are, the more the strength of a received signal tends to increase.

Most predominant RSS-enabled systems use a *Received Signal Strength Indicator* (RSSI) that is measured in *dBm*. The more negative the RSSI value is, the weaker the signal. Signal strength increases as values get closer to 0.

RSSI data is simple to use because it's compatible with most relevant systems without the need for further integration. Distance estimation is possible with RSSI [16], although not particularly reliable and accurate because of the effects of reflection and scattering in Non-Line of Sight (NLOS) scenarios [19]. As discussed in 2.1.1.2, the strengths of machine learning methods through fingerprinting are more suitable to the usage of RSSI.

2.1.2.2 ToF

Time of Flight (ToF) is the measurement of the time a wave spends travelling a distance through a medium. The distance between a transmitter and a receiver can be calculated using the ToF of messages between the two:

$$d = c \times t_{flight} \tag{2.11}$$

Where d is the distance between devices, c $\approx 3.8 \times 10^8$ m/s is the speed of light and t_{flight} is the time of flight.

This formula causes the calculated distance to incur a lot of error if there's significant error in the ToF [48]. Using the right technique to approximate the ToF through measurements is therefore vital to accurate estimation of distances. In the following subsections we discuss several techniques that use the concept of Time of Flight for distance estimation.

2.1.2.3 ToA

Time of Arrival (ToA) is the time instant when a signal is received by a device. Finding the ToF is possible through ToA at the receiver by subtracting it to the time the signal was sent by the transmitter. Then, distance can be estimated:

$$d = c \times (t_{arrival} - t_{sent}) \tag{2.12}$$

Where d is the distance, c $\approx 3.8 \times 10^8$ m/s is the speed of light, $t_{arrival}$ is the ToA at the receiver and t_{sent} is the time of sending at the transmitter.

This technique allows for distance to be estimated using one-way communication. However, there are requirements for it to properly function.

First, the transmitter must send information about the time the signal was sent. Additionally, the internal clocks of the transmitters and receivers must be highly synchronized, since as discussed in 2.1.2.2 errors in ToF can cause a lot of error in distance estimation. Finally, ToA is based on the assumption that the signal travels from transmitter to receiver along the shortest path. As is the case for every ToF-based technique, this isn't the case in Non-Line of Sight conditions.

2.1.2.4 TDoA

Time Difference of Arrival (TDoA) is a technique that uses the times of arrival at multiple receivers. Once the signal has been received at two reference points, the difference in ToA is used to calculate the difference in distances between the transmitter and both receivers:

$$\Delta d = c \times \Delta t \tag{2.13}$$

Where Δt is the difference $|t_1 - t_2|$ in ToAs t_1 and t_2 at receivers 1 and 2 respectively and c is the speed of light.

TDoA allows us to directly estimate positions instead of distances [48]. If (x_1, y_1) and (x_2, y_2) are the coordinates of known positions of receivers 1 and 2, we have the following equation:

$$\Delta d = \sqrt{(x_2 - x)^2 - (y_2 - y)^2} - \sqrt{(x_1 - x)^2 - (y_1 - y)^2} \tag{2.14}$$

Where (x, y) are the coordinates of the unknown transmitter position. This equation can be converted into a hyperbola. Once enough hyperbolas are found, (x, y) can be calculated by finding their intersection.

TDoA is beneficial because it relaxes the internal clock synchronization requirements for devices.

It is only needed that if there are multiple transmitters or receivers their internal clocks must all be synchronized.

2.1.2.5 RTT

Round Trip Time (RTT) is the amount of time between the sending of a signal added to the time for its acknowledgement to be received by the transmitter.

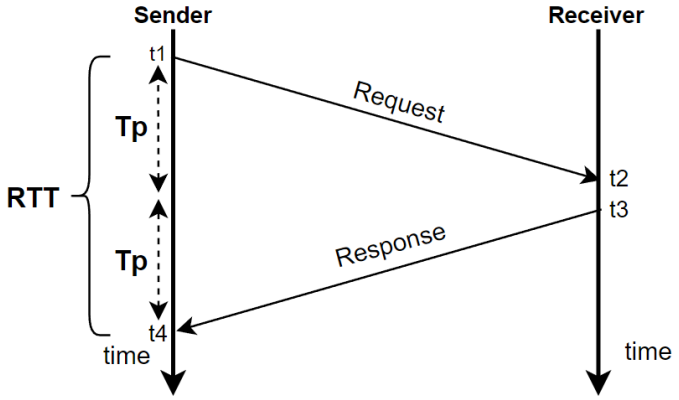


Figure 2.3: Example of message exchange between two devices for RTT calculation. Tp is the propagation delay of messages.

As illustrated in Figure 2.3, if t_1 is the timestamp at which the ranging request was sent and t_4 is the timestamp of receiving the acknowledgement for that request, the sender can estimate its distance to receiver through the following formula:

$$2 \times d_{1,2} = (t_4 - t_1) \times c \equiv d_{1,2} = \frac{(t_4 - t_1) \times c}{2} \tag{2.15}$$

RTT uses a two-way communication approach that requires no timestamps from other devices for Time of Flight calculations. Therefore, it overcomes the clock synchronization constraints present in other ToF-based techniques.

2.1.2.6 AoA

A signal's *Angle of Arrival* (AoA) is the angle from which it is received. This angle can be measured using TDoA measurements at each individual measurement of an antenna array. Antennas that are closer or further to the transmitter will receive the signal at different phases, which allows us to infer the direction which the signal was sent from.

In *Wireless Sensor Networks*, however, using arrays of antenna is impractical due to the nature of popular tag devices such as smartphones. There are ways of estimating AoA through data

that's available in these circumstances [28], but most are demanding in specialized hardware requirements.

2.1.3 Technologies

Localization through ranging is necessarily tied to radio wave-based technologies. There are many that can provide support for indoor localization [47]. In this section, we discuss these technologies by focusing on Wi-Fi, Bluetooth and Ultra-wideband.

2.1.3.1 Wi-Fi

Wi-Fi Round Trip Time (Wi-Fi RTT) is a feature added to the IEEE 802.11 protocol by the IEEE 802.11 Working Group, Task Group mc (TGmc). It is commonly abbreviated to 802.11mc and also refered to as *Wi-Fi Fine Timing Measurement* (FTM). This feature allows computing devices to measure the distance to nearby Wi-Fi access points.

Wi-Fi FTM's technique for data measurement works very similarly to the concept of RTT introduced in 2.1.2.5. Figure 2.4 illustrates the specific Wi-Fi FTM procedure.

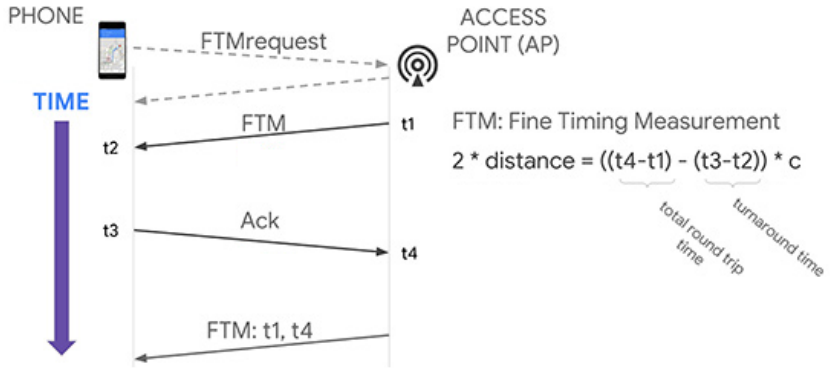


Figure 2.4: The concept of a Wi-Fi FTM exchange. [27]

Additionally, the Wi-Fi RTT ranging process can be configured to provide results based on the average of a set of distances that are estimated by executing this Wi-Fi FTM procedure with multiple network packets. The parameter that specifies the amount of packets to be used is called the RTT burst size [12]. This can potentially be advantageous to detecting and mitigating unreliable distance estimations.

Following IEEE 802.11mc standards, Wi-Fi itself is a network protocol with an indoor reception range of around 20 meters. It operates on the frequencies of 2.4 GHz and 5GHz. Modern Wi-Fi is commonly associated with throughputs between 500 *Mbps* and 1 *Gbps*. [14]

Wi-Fi already has a widespread infrastructure and integration with most popular devices. This makes it a popular choice for indoor localization research. Wi-Fi RTT, however, is a relatively

recent addition. It is predominantly limited to Google Wi-Fi APs and Google Nest Wi-Fi Routers [10].

Google Wi-Fi's main purpose is to serve as a line of mesh-capable wireless routers and add-on points. It's aim is to improve Wi-Fi coverage in a indoor environment while additionally providing smart home functionalities. It is decently easy to integrate its APs into currently existing infrastructures.

As 802.11mc becomes supported by more Wi-Fi devices, Wi-Fi RTT grows in potential for commercial use. For devices with no support, Wi-Fi indoor localization can be performed with RSSI, which is already supported.

2.1.3.2 Bluetooth

Bluetooth is a specification of wireless networks that is mainly used for data exchange between devices at short ranges, up to 10 meters. It operates on 2.4 GHz frequencies and can offer throughputs up to 2 *Mbps* as of Bluetooth 5.1.

Indoor localization through this technology relies on *Bluetooth Low Energy* (BLE) beacons. Traditionally, this technology has RSSI data measured at receivers [43] which, as mentioned in 2.1.2.1, can be used to estimate distances or in fingerprinting methods.

Bluetooth 5.1 introduced major improvements to localization with the Direction Finding feature [41]. This capability relies on Angle of Arrival data (as described in 2.1.2.6), which multiple-antenna array receivers can measure from messages broadcast from the tag device. Alternatively, a complementary *Angle of Departure* (AoD) technique can be used. In this approach, antenna arrays send signals through its multiple antennas, which the localization tag receives through its single antenna. The tag then calculates the transmitter's relative direction by analyzing the data samples of the in-phase and quadrature components of the signal.

2.1.3.3 Ultra-wideband

Ultra-wideband (UWB) is a radio technology that uses a very low energy level for short range, high bandwidth communications. It operates in frequencies between 3.1 GHz to 10.6 Ghz, covering a very wide portion of the radio spectrum. UWB is often linked with channel bandwidths of over 500 MHz.

The specifications of UWB allow the technology to transmit at high rates while overtaking interference from other radio technologies that operate within the same frequencies. Due to its large frequency spectrum, it is able to penetrate many softer materials in Non-Line of sight scenarios [49] [35]. Furthermore, UWB can be used to determine the time of flight at various frequencies. Some frequencies have a LOS trajectory to the receiver, while others with indirect paths will have longer delays [15]. These are all useful characteristics when attempting to

overcome multipath propagation.

Ultra-wideband is a promising choice for indoor environments due to their varying degrees of obstruction. It is able provide an accuracy of up to 10 centimeters and cover ranges of up to 200 meters, operating most effectively between 1 and 50 meters [48].

UWB's data measurement technique of choice is Time of Arrival, detailed in 2.1.2.3. Therefore, one of the technology's downsides is that accurate estimates require the internal clocks between senders and receivers in a UWB network to be highly synchronized. Furthermore, this technology has seen a considerably slow adoption rate. Not only is it not supported by most current devices, but also there currently are hardly any streamlined options for installation and integration of UWB into current infrastructures.

2.2 Localization through inertial motion sensors

Indoor localization can be performed through information about the motion of a tag. By combining sensor data, we can obtain information that allows us to estimate the displacement of a device during a time interval. Displacement can be estimated using a device's travelled distance and its heading, the direction in which it has travelled.

If we have information about the target's starting position and a series of displacements over time, we can continuously update its position. This is done by estimating a new position in function of its previous position and the displacement of the tag at that time interval.

In this section, we discuss the methods of localization through motion sensors, sensor data that is relevant to obtain motion information and prevalent sensor technologies that can be used to perform localization.

2.2.1 Methods

Methods of localization through motion sensors use information about acceleration, rotation and altitude to infer characteristics tag's movement. In this section, we focus on variants of a method that uses movement characteristics such as velocity and heading to iteratively update position estimates based on the tag's previous position.

2.2.1.1 Dead Reckoning

The base concept of *Dead Reckoning* is to incorporate estimates of speed, heading direction and time elapsed with the previous position of a tag in order to calculate its current position. Traditional dead reckoning is predominantly used in vehicle navigation and localization [20].

It can be simply conceptualized as follows: if v_t is the velocity vector at time instant t, p_t is the

position vector of the tag and R_t and a_t are the sampled rotation matrix and acceleration vector at instant t, then

$$v_{t+1} = v_t + (R_t a_t) \tag{2.16}$$

$$p_{t+1} = p_t + (v_{t+1}\Delta t) \tag{2.17}$$

Essentially, we use the sampled rotation information to orientate the acceleration vector in accordance to the heading of the tag in the global coordinate system. We then calculate the current velocity by adding this transformed acceleration information to the previous velocity. Displacement of the tag is then calculated by multiplying current velocity with the time elapsed since the last instant. Finally, a new position is found by adding the displacement to the previous position.

2.2.1.2 Pedestrian Dead Reckoning

The objective of *Pedestrian Dead Reckoning* is to provide a motion sensors navigation method specialized for human use. The algorithms that follow this category of methods are primarily meant to be installed in tag devices like smartphones or smart wear. It's three main components are step detection, step length estimation and heading estimation.

Step detection techniques aim to develop the functions of a pedometer using built-in device sensors. Two of the most relevant approaches use the acceleration and rotation data of the tag [30] to detect step occurrences. The intent behind them is to detect moments when the user's foot lifts up, swings and touches back down on the ground.

In an acceleration step detection approach, the concept is to

- Transform sampled acceleration data into useful acceleration information. For example, this could be extracting the acceleration's vertical component a_{z_i} or computing the magnitude of the acceleration $a_i = \sqrt{a_{x_i}^2 + a_{y_i}^2 + a_{z_i}^2}$.
- Apply filtering techniques to remove outliers in the data. This could be done to the raw sampled data or the processed information.
- Use a peak detection or thresholding method on the acceleration information to identify steps.

In the alternative approach that uses rotation data, gyroscopes and magnetometers are used in conjunction to calculate either the total magnitude or a component of the tag's angular rate. The method is then similar in concept to the acceleration method, applying filtering techniques to remove outliers and thresholding the angular rate to detect step events.

Stride estimation techniques estimate the step length (SL) of the user at a given step event. The

following list contains some of the most relevant approach in the wide variety of available solutions:

- Set SL to a constant value, normally between 0.5 to 1 meters.
- Configure it to a constant value in function of the current user's characteristics like height, age and gender, which tend to influence the stride [21].
- Compute it for every step event in function of the changes in vertical acceleration [21] [31]. This assumes that SL is proportional to a human body's vertical bounce as their hip moves between steps.
- For every step event, calculate SL in function of the tag's linear velocity [30]. In certain foot stances during a step, this velocity is known to be zero. This approach finds the moments where those stances occur and corrects the linear velocity values accordingly. With linear velocity and time elapsed we can calculate position displacement vector, of which we take the horizontal distance components to find SL.

Heading is then estimated through a combination of gyroscope and magnetometer readings that try to emulate the function of a compass. There are multiple common output units for this information including angles [45], rotation matrices [44] and quaternions [42]. All of these units can be obtained from each other through transformations.

Assuming the heading is given by an angle h_t with relation to Earth's magnetic north (in a global East-North-Up coordinate system, GCS), then a PDR position update [31] at timestamp t can be expressed by

$$x_t^{GCS} = x_{t-1}^{GCS} + SL \times \sin(h_t) \tag{2.18}$$
$$y_t^{GCS} = y_{t-1}^{GCS} + SL \times \cos(h_t) \tag{2.19}$$

Where (x_t, y_t) are the coordinates of the tag's position at time t.

2.2.2 Data Measurement Techniques

Motion sensor technologies are valuable for localization because they can provide data used to calculate the distance travelled and bearing of a tag. In this subsection, we list motion sensors that are relevant for enabling localization and the techniques associated with using the sensor's data to estimate characteristics of a target's movement.

2.2.2.1 Accelerometer

Currently, most motion sensor technologies employ three-axial accelerometers. Multi-axis accelerometers can measure the magnitude and direction of acceleration caused by movement,

vibration or gravity. This force is usually represented in vector format, using m/s^2 as an unit.

In localization, acceleration can be used to obtain speed and travelled distance. Velocity can be calculated by integrating acceleration. Using velocity and time elapsed, calculating distance travelled becomes possible. If v corresponds to the velocity, a to the acceleration and d to the travelled distance, then the general concept is the following:

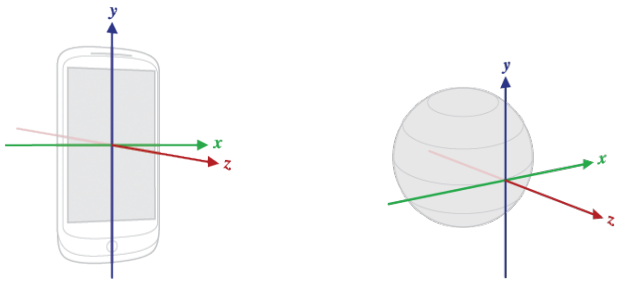
$$v(t) = \int a \; dt \tag{2.20}$$

$$d(t) = \int v \; dt \tag{2.21}$$

Accelerometer outputs, however, are often transformed in combination with other sensor data in order to obtain actually useful information about the movement of a device [26].

There are several sources of error in applying travelled distance estimation techniques to an accelerometer's data alone. First, as is the case with Android smartphones, the device's axial system can be relative to the orientation of the phone itself (Figure 2.5). Second, regardless of phone orientation, raw accelerometer data is influenced by the pull of gravity (9.81 m/s^2). This disconnect is especially problematic because localization systems tend to use coordinate systems that represent device motion in relation to the Earth, typically in a East-North-Up axis system.

Further discussion on how specific technologies combine sensors to provide useful motion information can be found in subsection 2.2.3.



Local coordinate system Earth coordinate system

Figure 2.5: Representations of smartphone and earth coordinate systems [1]

2.2.2.2 Magnetometer

Magnetometers measure the orientation of a device in relation to Earth's magnetic north. Without any close by magnetic interference, it will sense the Earth's magnetic field, which points north. Magnetometer data is usually presented as a 3D vector pointing to the strongest magnetic field.

Magnetic field sensor data is usually combined with gyroscope to provide accurate information about a device's change in orientation. In 2.2.2.3, further details about this process are provided.

Orientation data can be used together with acceleration data to calculate velocity of a device in relation to Earth's referential. As detailed in subsection 2.2.1, information about heading and velocity of a device is required to perform methods of Dead Reckoning.

2.2.2.3 Gyroscope

Gyroscopes measure angular velocity in relation to the device's coordinate system (Figure 2.5), with most devices being equipped with a three dimensional gyroscope that measures angular velocity at all three axes. Measurements are usually presented in *rad/s*.

A gyroscope is usually used in combination with a magnetometer to measure changes in the rotation of a device [26]. The gyroscope's function is to complement the magnetometer's readings for better precision about a device's rotations. It also enables increased frequency in rotation measurements since, usually, a magnetometer is much slower than a gyroscope.

Overall, gyroscope data is good at informing about small changes in rotation rather than the device's orientation.

2.2.2.4 Barometer

Barometers are used to measure air pressure. Depending on the type of calibration given to it, a barometer can serve as a pressure altimeter that is used to measure the altitude above a fixed level, since pressure lowers as the altitude increases.

In indoor localization, altitude estimation is particularly helpful to know if the user crosses between the floors of a building. This is useful complementary information to a lot of localization methods that are better suited to estimating within a specific floor. Positioning estimates can be narrowed down in moments where a floor change has been detected. If we have previous information about the building layout, then we can identify a limited set of floor entry points.

Barometer readings are affected by factors like changes in temperature which impact air pressure. However, especially considering that pressure altimeters have a different calibration, this noise is relatively small compared to readings that correspond to actual changes in altitude.

2.2.3 Technologies

Localization through inertial motion sensors requires that a variety of sensor readings are taken for a single tag. In this section, we discuss hardware technologies that incorporate multiple sensors in order to coherently provide all necessary measurements.

2.2.3.1 Inertial Measurement Units

Inertial Measurement Units (IMU) are embedded electronic devices composed of accelerometers, gyroscopes and magnetomers. They measure triaxial linear acceleration, triaxial angular velocity and magnetic field strength.

As discussed in 2.2.2.2 and 2.2.2.3, magnetometer readings are combined with gyroscope readings to obtain the device's orientation with respect to Earth's magnetic north (for example, in a East-North-Up global coordinate system, GCS). Local linear acceleration readings from the IMU can then be transformed to GCS values by using device orientation:

$$a_t^{GCS} = R_t a_t^{LCS} \tag{2.22}$$

Where for time t, a_t^{GCS} and a_t^{LCS} are the global and local linear acceleration vectors respectively and R_t is the device orientation in rotation matrix [44] format.

Typical uses of IMUs include vehicle and robot navigation and measuring acceleration and orientation in consumer devices like smartphones, tablets and wearables. Almost all modern Android devices contain IMUs.

2.2.3.2 Android Sensors Suite

Android devices are equipped with a diversified suite of sensors. A detailed overview of all available sensors in the Android API can be found in [2], with sensor availability depending on specific devices. Plenty of can be used for indoor localization, including accelerometer, gyroscope, magnetometer, barometer, proximity, temperature, light and humidity sensors.

One of the most useful Android sensor features is that its software offers functionalities to transform the raw sensor data at the API level. For instance, the following can be used:

- A linear acceleration sensor type which outputs triaxial linear acceleration with the force of gravity already filtered out.
- A rotation vector sensor type which outputs device orientation as a quaternion, with reference to a East-North-Up global coordinate systems.
- Functions that transform rotation vectors into other rotation information units.
- Functions that use rotation information to rotate geometric structures.

Achieving these results usually requires a combination of multiple sensors which takes up significant development time to implement.

Android smartphones are very interesting tags for indoor localization. They are already commonly used devices, they are able to collect data for motion sensor and ranging methods and

they provide convenient software interfaces for both. This makes them helpful for users and useful for developers.

2.3 Localization improvement

The development of localization solutions includes a variety of nuances that go beyond the process of simply pairing technologies with methods to produce positioning estimates.

In this section, we begin by discussing common challenges in localization research that have an impact on the quality of the developed solutions. Then, we talk about useful techniques and general good practices for improving localization accuracy while introducing the proper definitions of the most relevant methods.

2.3.1 Line of Sight in ranging measurements

The existence of Line of Sight (LOS) between a tag and reference points is a factor that heavily influences the accuracy of the distances reported by each anchor device. Network-based technologies that perform range estimation rely on communications through radio waves. However, when line of sight between transmitters and receivers is obstructed the propagation of waves will be different.

Non-Line of Sight (NLOS) propagation occurs through less predictable phenomena such as reflections, changes of direction when waves bounce off of surfaces or refraction, when waves bend as they pass between materials. This can lead to issues like multipath propagation which occurs when a signal reaches a receiver through multiple paths. If the receiver doesn't have any safeguards or mechanisms to isolate the signal from its replicas, this can result in significant interference.

The main challenge NLOS propagation brings to range estimation is its disruptive impact on localization-enabling data. Distance estimation techniques were primarily designed under the assumption of the data being collected in predictable line of sight conditions. Furthermore, systematically identifying instances of this effect is complex, as is defining how to compensate for its impact. As mentioned in 2.1.2.1, RSSI distance estimation is unreliable due to the effect of reflections and scattering. Angle of Arrival estimation (described in 2.1.2.6) is necessarily affected if, due to phenomena like reflections, waves reach antenna arrays in different directions than the direction of their transmitter. Finally, as shown in 2.1.2.2, inaccuracies in the Time of Flight cause significant error in distances estimated through it. NLOS propagation causes the ToF to increase, which will proportionately manifest itself in estimates through a consistently excessive distance estimation.

2.3.2 Drift in Dead Reckoning positioning

Localization through inertial motion sensors commonly tends to rely on dead reckoning methods to provide positioning estimates. The core issue with these methods originates from their reliance on relative positioning, which is a consequence of motion sensors not having any means to collect information that enables the estimation of absolute positions.

To elaborate, as we describe throughout 2.2.1, dead reckoning algorithms always estimate the position of a tag in function of the tag's previous position. This is done because acceleration and rotation information only allow for the estimation of how the tag moved over a period of time. As such, if the algorithm can characterize this movement then it can also calculate a new position by applying that movement as a displacement to the current position.

The problem with this approach lies in the fact that acceleration and rotation measurements usually carry error. As a consequence, the displacement that is calculated using that data will result in the updated position being inaccurate. At first, this imprecision is residual. However, as the algorithms iterate new position updates will be performed in function of positions that are already imprecise. As such, consecutive errors in sensor information will cause positions to *drift* further from the truth, resulting in increasing levels of inaccuracy over time.

2.3.3 Kalman Filter

The Kalman Filter [46] [16] is an iterative mathematical process that uses a set of equations to consecutively estimate true values from a series of noisy measurements of those values. Due to its key role in localization improvement through data smoothing and data fusion (as we discuss in 2.3.4 and 2.3.5), we will now introduce the algorithm using Table 2.1 as a reference that labels each of the key terms used in the set of equations that compose the filter. Every iteration, this process has two phases: the prediction stage and the update stage.

In the prediction stage, information about the value's previous state and how the value is expected to transition between states is used to predict the value's next state and the uncertainty associated with that prediction.

$$\hat{x}_{t_p} = A\hat{x}_{t-1} \tag{2.23}$$

$$P_{t_p} = AP_{t-1}A^T + Q \tag{2.24}$$

In the update stage, the prediction's uncertainty is used to calculate the Kalman gain. This represents how reliable the process considers the predicted value and measured value to be. A smaller Kalman gain means the predicted value is more trusted. As such, less weight should be given to the measured value when the state variable is updated.

$$K = \frac{P_{t_p} H}{H P_{t_p} H^T + R} \tag{2.25}$$

$$\hat{x}_t = \hat{x}_{t_p} + K(z_t - H\hat{x}_{t_p}) \tag{2.26}$$

$$P_t = (I - KH)P_{t_p} \tag{2.27}$$

Equation 2.26 shows how the next state is calculated in function of the predicted and measured values for the current iteration, using the Kalman gain as a weight factor for each value. In equation 2.27, the uncertainty in the newly estimated state is updated in function of prediction's uncertainty and the inverse of the Kalman gain.

Term	Meaning
$\hat{x}_t$	State estimate
$\hat{x}_{t_p}$	Prediction state estimate
P_t	Uncertainty of the state estimate
P_{t_p}	Uncertainty of the predicted state estimate
A	Predicted State transition
K	Kalman gain
z_t	Measurement
Q	Process noise
R	Measurement uncertainty
I	Identity matrix
H, H^T	Matrices used to transform parameters into the format of measurements

Table 2.1: Key terms in the Kalman Filter equations

2.3.4 Data Smoothing

Smoothing is the process transforming data in a manner that allows it to preserve important patterns while reducing any noise that obstructs the information we want to observe. In the context of localization, smoothing is useful because a lot of sensor readings tend to contain sizeable uncertainty in them. Without any adjustments, many methods that use this data will provide estimates that tend to also have sharp fluctuations. As such, data smoothing could be applied to a variety of sensor readings such as distance data from ranging technologies or acceleration and rotation data from motion sensors. Alternatively, some smoothing methods could also be applied to the actual positions that are estimated by localization algorithms.

A common and relatively simple way to remove noise is the *low-pass filter*, which passes signals whose frequencies are lower than a specified threshold while attenuating signals that are higher. These can be especially useful, for example, to smooth motion sensor data [31] [30] which is a type of data that's prone to a good amount of uncertainty.

Alternatively, moving averages smooth out small variations while preserving lasting tendencies in the data. This is done by using the series of measurements to create a series of averages from

different subsets of the entire data set. The subsets that are considered for this process are the ones resulting from a shifting window that moves through every data point, selecting a subset of the last k data points before it.

For a data point n, a simple moving average with window size k is described by equation 2.28, where p_i is the measurement for data point i.

$$MA_k = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i \tag{2.28}$$

Finally, the Kalman Filter is an especially relevant smoothing technique. As previously described in 2.3.3, they take a series of noisy measurements and estimate values that are more accurate and coherent with the variable's past states. In fact, they are designed to take into account previous knowledge relating to multiple types uncertainty associated with the process, measurements and state transitions. In practice, the fact that they can be applied to position estimates also makes them particularly interesting in the field of localization tracking.

2.3.5 Data Fusion

Data Fusion represents the process of combining data from multiple sources in order to transform them into a single piece of information that is more useful and precise. When applied to the context of localization, Sensor fusion becomes the relevant subset of data fusion. This is the process of integrating multiple sources of sensor data or, alternatively combining sensor data with other relevant information such as video camera information processed through computer vision or Wi-Fi localization signals.

One of the motivations behind sensor fusion is that distinct sensor types tend to have a lower likelihood of the noise that is inherent to each type being correlated with another type's noise. As such, when combining these sensors, the intent is that the parts of the signal that approximate true values become more pronounced due to their correlation while the noise in the signal becomes attenuated.

A relevant case which illustrates the point of sensor fusion is the calculation of a tag's orientation through the fusion of a device's magnetometer and gyroscope readings. As previously mentioned throughout 2.2.2.2 and 2.2.2.3, both sensors provide data about rotation. A magnetometer by itself can already measure the orientation of a device, but the quality of this measurement is reduced if the sensor is near any materials with magnetic pull. In practice, keeping the magnetometer away from such materials is hard. Therefore, fusion with angular rate data that comes from the gyroscope is a common technique because those readings suffer from inherent noise relative to their frequency which is not related to the magnetometer's noise source.

In the context of localization improvement, ranging localization algorithms will provide position estimates that showcase different weaknesses from estimates provided by dead reckoning

algorithms. This happens due to the fact that their sources of error tend to have little to no strong correlation. As such, the application of data fusion to their results tends to be beneficial.

One of the more common approaches to this problem is the Kalman Filter [39] [16], which we detail in section 2.3.3. One of the more useful things about the Kalman Filter is that it enables for the mathematical model of the system to already be built into the fusion procedure. This mathematical model allows us to introduce our knowledge about what we're trying to estimate. For instance, the use case of estimating the position of an object moving through a two-dimensional plane is especially relevant to localization. For that example, the Kalman Filter enables us to introduce our knowledge about how the object is expected to move according to its velocity over a time frame.

Regardless of being employed at the raw data-level or at a higher level of processed information, there are several other algorithms that can accomplish sensor fusion. Other interesting solutions include the use of *particle filters* [39] which, unlike the Kalman Filter, don't incorporate previous knowledge about the distribution of measurements and instead approximate that distribution through the use of discrete samples. Alternatively, machine learning approaches that employ *recurrent neural networks* [25] [23] can be used to recursively predict errors in recent measurements through information about the tag's previous position and velocity.

Chapter 3

Algorithm Design for Indoor Localization

In this chapter, we discuss our proposed methods for indoor localization using Android smartphone devices. Namely, we focus on our work on multilateration methods using Wi-Fi RTT data and Pedestrian Dead Reckoning using Android motion sensor data. Finally, we introduce our proposed sensor fusion algorithm. It aims to combine both of our technologies of choice in order to reduce the impact of their inherent weaknesses and improve localization accuracy as a result.

3.1 Multilateration using Wi-Fi RTT

Wi-Fi is an interesting option for localization research due to its already existing widespread infrastructure. The Wi-Fi RTT feature expands this technology by allowing compatible devices to measure the distance to Access Points through a time of flight-based technique.

Although this feature has limited availability in devices due to its slow adoption in Wi-Fi chipsets, it counterbalances that fact with its potential for greater precision and ease of integration with current systems and infrastructures. In fact, Android devices already have an comprehensible Wi-Fi RTT API [13].

In this section, we explain what localization-enabling information Wi-Fi RTT provides and our ideas for using that data. We describe our chosen base method to achieve Wi-Fi RTT localization and discuss techniques that can be used to improve accuracy in this process.

3.1.1 Localization using Least Squares

Least Squares is the base algorithm we chose to provide localization estimates through the multilateration of Wi-Fi RTT measurements. We first introduced and defined the concept of

multilateration by applying a least squares optimizer to reported distances between a tag and points of reference in Section 2.1.1.1.

To summarize, formulating a Least Squares localization problem requires that we know the (x_i, y_i, z_i) coordinates of a set of access points and d_i, the distance between AP_i and the tag, with $i = 1, 2, ...N$ and N being the amount of available APs. This information allows us to characterize a set of expressions whose composition is described in the following equation:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = d_i^2$$

Using linearization and linear algebra transformations, we can solve an equation system with this set of equations to find the optimal (x, y, z) tag position for the given reported distances to each AP. Wi-Fi RTT enables these calculations by providing estimated distances between Wi-Fi Access Points and the smartphone device they are reporting to.

We previously detailed how this technology handles distance estimation in 2.1.3.1. Essentially, Wi-Fi RTT employs a technique to measure the time of flight of packets between devices without relying on external timestamps. By using the device's own timestamps, it eliminates any need to synchronize internal clocks between participating devices. Then, Wi-Fi RTT assumes that distance is directly proportional to this time of flight.

Meanwhile, mechanisms to advertise AP positions are largely limited or unavailable for Wi-Fi RTT enabled devices. For most use cases, other ways must be found to share this information. For example, this could be done by downloading it from a database or by having a previously established convention in the application's code.

3.1.2 Improving Least Squares localization

During our research we considered multiple techniques that could be used to improve the process of applying Least Squares to Wi-Fi RTT data. In specific, we found that data smoothing tended to be quite beneficial. Additionally, we examined what uses could be given to the additional data reported in Wi-Fi RTT ranging results.

We now describe four data smoothing techniques, two of which can be applied to Wi-Fi RTT ranges, one that is applied to positions estimated by Least Squares and a last one that combines smoothing in both ranges and positions. Then, we finish by discussing optimizations to Wi-Fi RTT data that take advantage of the standard deviation metric that is reported in each access point's ranging results.

3.1.2.1 Moving Average applied to Access Point ranges

This technique applies N independent moving averages to each individual series of reported distances $P = \{p_{j,1}^{dist}, p_{j,2}^{dist}, .., p_{j,M}^{dist}\}$ from a set of access points identified by $j = 1, 2, 3...N$ and M

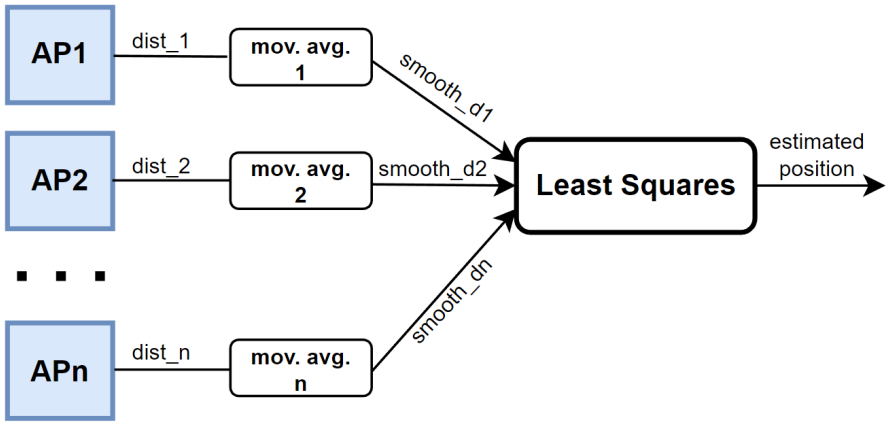


Figure 3.1: An overview of the moving average data smoothing technique.

being the total number of ranging results over time, as illustrated by Figure 3.1. A simple moving average smooths data by transforming each data point into the unweighted mean of the last k data points.

In this context, we consider a data point p to be a pair of timestamp p^{ts} (in deciseconds) and p^{dist}, which is an AP's reported distance (in meters) at that timestamp. Then, we apply a moving average over a time window of 1 second. Since the timestamps of the data point series are not necessarily periodic, this means that the size of k can vary every data point.

In specific, if we are calculating the moving average for data point n, then we have

$$MA_n = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i^{dist}, \quad k = \left\{ \max(t) \;\middle|\; p_n^{ts} - p_{n-t+1}^{ts} \leq 10 \times w \right\} \tag{3.1}$$

which requires k to be the largest amount of data points whose timestamps are not more than $w = 1$ seconds apart from the timestamp of data point n.

We consider smoothing over a time window of 1 second to be adequate for Wi-Fi RTT data, given that we observed that APs are able to output ranging results several times per second. Choosing a larger window can be done by adjusting w. However, since we need to consider tags that are moving, the risk of the window including values that are not representative of the tag's actual distance to the AP tends to increase.

3.1.2.2 Kalman Filter applied to Access Point ranges

This technique applies N independent one-dimensional Kalman Filters to each individual series of reported distances $P = \{p_{j,1}^{dist}, p_{j,2}^{dist}, .., p_{j,M}^{dist}\}$ from a set of access points identified by $j = 1, 2, 3...N$ and M being the total number of ranging results over time, as illustrated by Figure 3.2.

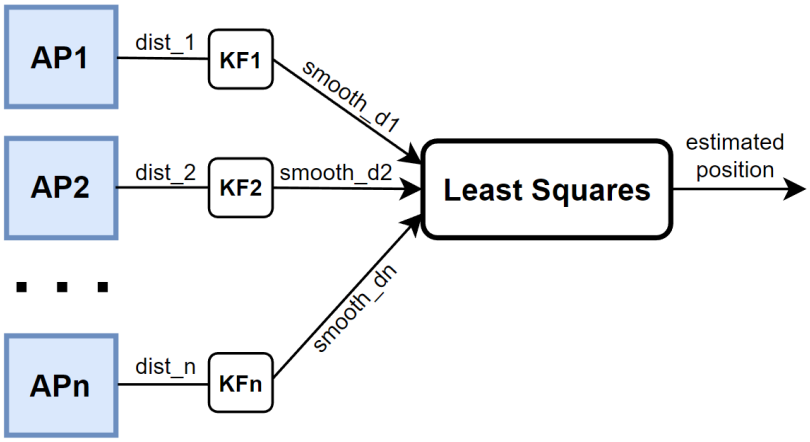


Figure 3.2: An overview of the one-dimensional Kalman Filter data smoothing technique.

We first introduced the concept of the Kalman Filter in 2.3.3. Contextualizing that description to this problem, consider that an access point's ranges are a set of data points p, a pair of timestamp p^{ts} (in deciseconds) and an AP's reported distance p^{dist} (in meters). Then, in this context the format of measurements is described by equation 3.2.

$$z_t = p_n^{dist} \tag{3.2}$$

where $t = p_n^{ts}$, the timestamp for the nth data point. This is a type of data that requires a different interpretation of the Kalman Filter than the one previously introduced. While the general concept is the same, we will now introduce a description that is more suitable for this one-dimensional data. First, we have the prediction stage:

$$\hat{x}_{t_p} = \hat{x}_{t-1} + s_t \Delta t, \quad s_t = \hat{x}_{t-1} - \hat{x}_{t-2} \tag{3.3}$$

$$p_{t_p} = p_{t-1} + \Delta t^2 \cdot U \tag{3.4}$$

In this stage, our predicted state estimate is calculated proportionally to the difference s_t between the last two estimated states. This prediction strategy can be thought of as an extrapolation of the next state estimate. Then, the uncertainty p_{t_p} associated to this predicted state estimate is found to transition proportionally to Δt, the time elapsed between timestamps t and $t-1$. Finally, for the purpose of equation 3.4, the scaling factor U is the uncertainty associated with the difference s_t, which we set to $U = 0.065$.

After the prediction stage, we have the update stage:

$$k_t = \frac{p_{t_p}}{p_{t_p} + R} \tag{3.5}$$

$$\hat{x}_t = \hat{x}_{t_p} + k_t \left(z_t - \hat{x}_{t_p}\right) \tag{3.6}$$

$$p_t = (1 - k_t)\, p_{t_p} \tag{3.7}$$

The concept behind this stage is identical to the one that was previously described. First, the Kalman gain is calculated through the uncertainty associated with the predicted state p_{t_p} and the uncertainty in the measurement, which we set to $R = 2$.

Then, in the state estimate update we use the Kalman gain k_t as a weight factor between the predicted value $\hat{x}_{t_p}$ and the measured value z_t. A higher Kalman gain results in the measured value having more weight, as the system considers it more reliable than the prediction. Meanwhile, the uncertainty in this updated state estimate is inversely proportional to the Kalman gain.

Other than providing adequate results, the basis behind the choice of the value for R is that 2 meters is a suitable margin of error for distances estimated by Wi-Fi RTT access points [10]. Meanwhile, $U = 0.065$ was tuned considering that changes in reported distances between data points were expected to be small due to the high frequency of ranging results, with typical values of 2 to 3 results per second.

3.1.2.3 Kalman Filter applied to Least Squares estimates

This technique applies a single Kalman filter over the series of estimated positions that result from iteratively applying Least Squares to sets of reported distances from all APs to the tag, as illustrated by Figure 3.3.

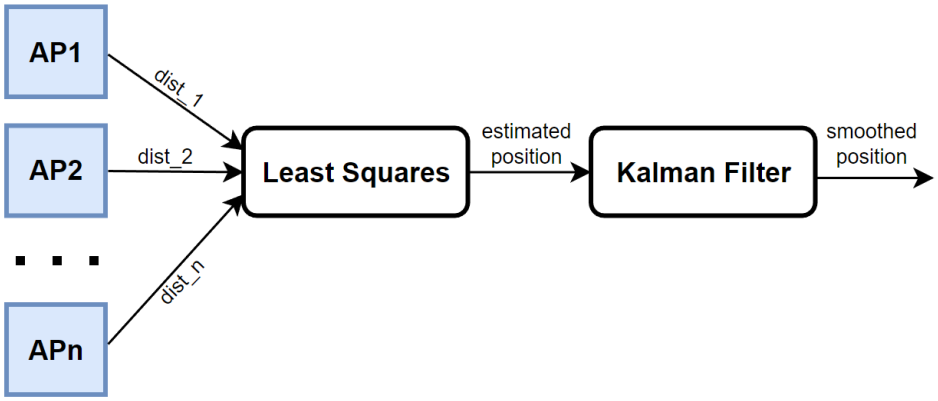


Figure 3.3: An overview of the two-dimensional Kalman Filter data smoothing technique.

The concept of the used Kalman Filter approach is identical to the one previously described in 2.3.3. In this context, we apply this description to the problem of a moving object in a two-dimensional plane. We will now highlight the key differences between the two. First, in this application the state estimate takes the form of

$$\hat{x}_t = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

where x and y are the coordinates of the tag's two-dimensional position and $\dot{x}$ and $\dot{y}$ are the components of the tag's velocity in the x and y direction. In turn, the state transition is defined as

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which describes the transition of state $\hat{x}_t$ from time $t-1$ to time t, with an elapsed time interval of Δt.

Finally, in order to predict a new state, we apply the state transition A to $\hat{x}_{t-1}$ by means of multiplication:

$$A\hat{x}_{t-1} = \begin{bmatrix} x + \Delta t\dot{x} \\ y + \Delta t\dot{y} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

When contextualized to the problem of a moving object, this equation intuitively describes the addition of the travelled distance over a time interval in the x and y directions to the previous position of the object. This results in an extrapolation of the object's new position through the information about its estimated velocity in the previous iteration of the filter.

3.1.2.4 Combining Kalman Filter techniques

Kalman filters applied to Wi-Fi RTT access point ranges can be combined with the ones applied to Least Squares estimates. The way we designed this combination is rather simple, as illustrated by Figure 3.4.

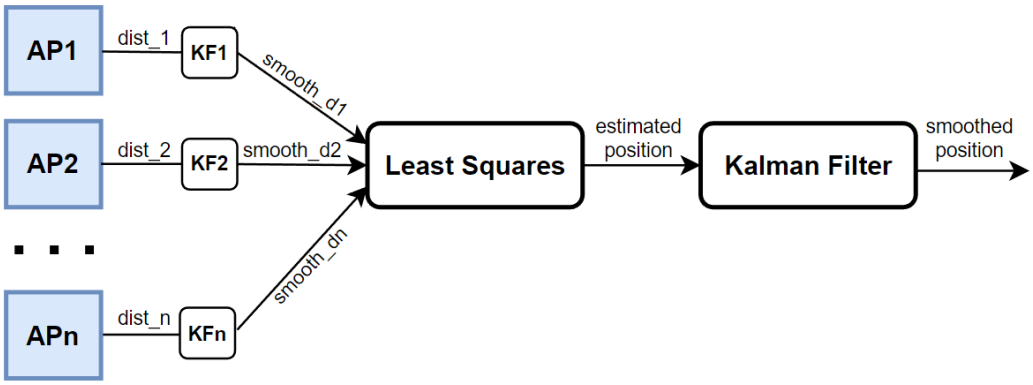


Figure 3.4: An overview of the combination of both Kalman Filter data smoothing techniques.

To begin, N independent Kalman Filters are applied to the individual series of reported distances, one for each Wi-Fi RTT AP. Then, the resulting series of smoothed distances are used to perform Least Squares. Finally, the positions estimated by Least Squares are passed through the Kalman Filter that acts on LS estimates.

3.1.2.5 Optimizations using Standard Deviation data

Every ranging result, Wi-Fi RTT access points report a variety of data together with their estimated distance to the tag. This includes a standard deviation field. The distances estimated by Wi-Fi RTT are the mean value of a set of distances, as previously documented in 2.1.3.1. The size of this set is determined by the RTT burst size field which we set to 8, its default value in the Android Wi-Fi RTT API.

The hypothesis that is then considered is that this standard deviation metric could be used as an indicator for which access points are more reliable at a point in time. Thus, the first optimization that was tested is the one illustrated Figure 3.5. It's description is the following:

- For every iteration of Least Squares, use a subset of reported distances from the total set reported by all available APs at that time;
- The size of the subset should be equal or greater than 4, since Least Squares requires at least four reference points to calculate a three-dimensional position. For testing purposes, we always set this value to 4;
- The elements of the subset should be the reported distances that correspond to the APs with the minimum values of reported standard deviations.

Another possibility we considered was subtracting the standard deviation from its corresponding estimated distance, as shown in Figure 3.6. This consideration stems from the assumption that

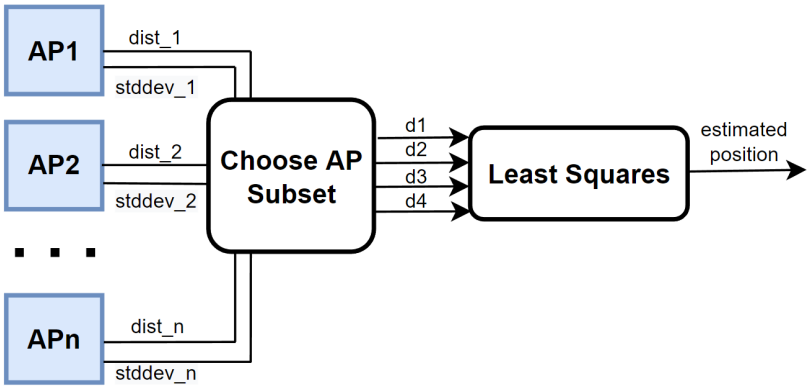


Figure 3.5: An overview of our proposed optimization that uses standard deviation reported by Wi-Fi RTT APs as a criteria to select an AP subset.

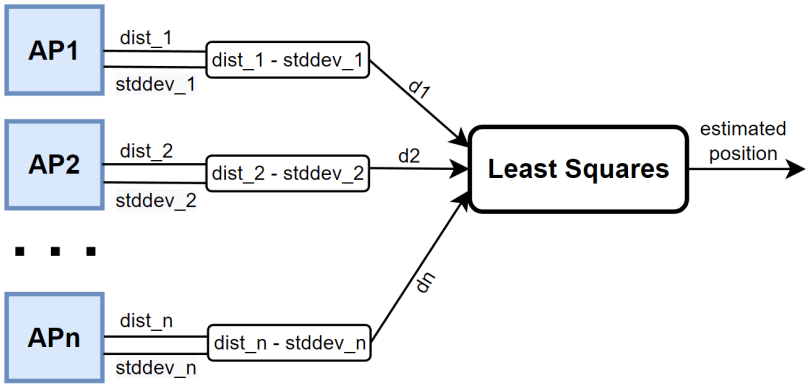


Figure 3.6: An overview of our proposed optimization that subtracts reported standard deviations to distances estimated by Wi-Fi RTT.

unreliable APs tend to estimate distances that are longer than true distances. In NLOS scenarios, phenomenons such as multipath propagation cause the time of flight of Wi-Fi FTM messages to be longer and more unstable. In turn, the estimated distances are calculated proportionally to this time of flight.

As such, our second hypothesis is that higher standard deviations will be reported in NLOS situations because the estimated distances will show more variance. By subtracting deviations from distances, we aim to partially correct the expected excess in the estimates. In opposition, when APs are reliable the reported standard deviation will be lower, which is favourable to preventing overcorrections.

3.2 SmartPDR

Our pedestrian dead reckoning method is based on the work of Kang and Han [31], who introduced *SmartPDR*. This is a solution for smartphone-based pedestrian dead reckoning that tracks user's movements using their device's embedded inertial sensors. Similarly to the concepts introduced in 2.2.1.2, this approach can be mostly divided into 3 techniques: step detection, heading estimation and step length estimation.

SmartPDR's key idea is to calculate a pedestrian's displacement from their previous position in function of step events. Step events are detected by observed signature patterns on the device acceleration's vertical component a_z, relative to the ground.

In smartphones, the z-axis component of acceleration is often distributed among all three axis since these devices output data referring to a local coordinate system (LCS) related to the device. As such, to extract the true vertical component of the signal, we must transform the acceleration data into a global coordinate system (GCS) that follows Earth's orientation

$$a_t^{GCS} = R_t \; a_t^{LCS} \tag{3.8}$$

We first explained this transformation in 2.2.3.1. To summarize, we use rotation information about the device orientation R_t in relation to Earth's magnetic North to transform the representation of acceleration in the local coordinate system a_t^{LCS} into the corresponding representation a_t^{GCS} in the global coordinate system.

From the series of a_t^{GCS}, we can finally retrieve their a_z components. From this series of vertical accelerations, we can now look for patterns in order to find step events. Step events are identified by their corresponding accelerations a_t^{step} and timestamps t^{step}. SmartPDR employs three conditions on vertical accelerations to detect valid step events:

$$t^{peak} = \left\{ t \;\middle|\; a_t^{step} > a_{t+i}^{step}, \;\; a_t^{step} > a_\tau^{peak}, \;\; |i| < \frac{N}{2}, \;\; i \neq 0 \right\} \tag{3.9}$$

$$t^{pp} = \left\{ t \;\middle|\; max\ (a_t^{step} - a_{t-i}^{step}) > a_\tau^{pp}, \;\; max(a_t^{step} - a_{t+i}^{step}) > a_\tau^{pp}, \;\; 1 \leq i \leq \frac{N}{2} \right\} \tag{3.10}$$

$$t^{slope} = \left\{ t \;\middle|\; \frac{2}{N} \sum_{i=t+1}^{t+\frac{N}{2}} (a_i^{step} - a_{i-1}^{step}) \; < 0, \;\; \sum_{i=t-\frac{N}{2}}^{t-1} (a_{i+1}^{step} - a_i^{step}) \; > 0 \right\} \tag{3.11}$$

where N is the size of a moving window of vertical acceleration samples, which we set to $N = 3$. At each window iteration, all three conditions are checked. When all iterations are completed, the sets of time points that meet each condition are returned:

- t_{peak} is the set of time points where accelerations are above the threshold $a_\tau^{peak} = 0.5 m/s^2$ and are higher than all other accelerations within their window. These are commonly

called *peaks*;

- t_{pp} is the set of time points resulting from a peak-to-peak check. In this check, each peak is compared to the *valleys* right before and after it. A *valley* is considered as the time point between two peaks where the vertical acceleration is the lowest. The differences between the peak and its adjacent valleys must both be greater than the threshold $a_{\tau}^{pp} = 1m/s^2$ in order to pass this condition;
- t_{slope} is the set of time points whose corresponding accelerations show ascending growth on their front side and descending growth on their back side.

Figure 3.7 illustrates the provided definition of peaks and valleys using a plot. The set of step events t_{step} can then be obtained by intersecting the sets resulting from each condition, as described by $t^{step} = t^{peak} \cap t^{pp} \cap t^{slope}$. Intuitively, this means that an acceleration signature is considered a step event if it meets all three conditions stated above.

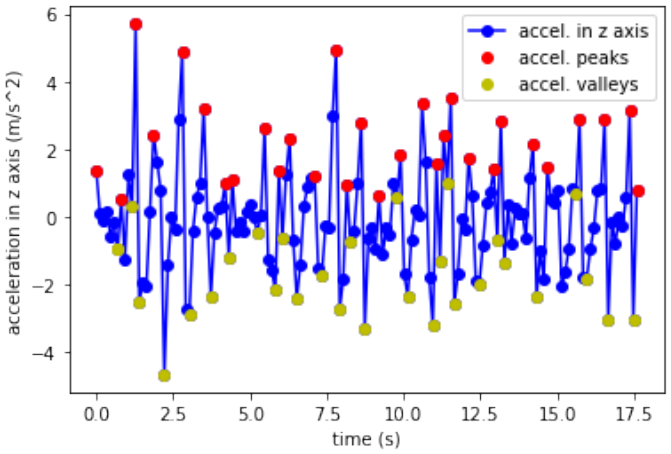


Figure 3.7: Example of peak and valley detection in the z-axis component of a tag's acceleration over time.

The final result of the step detection process is a set of step events identified by the time point and acceleration of their corresponding peaks. Heading for the kth step can then be obtained by fetching heading information for the valley that precedes that step:

$$h_k = h_t|_{t=t_k^{head}}, \quad t_k^{head} = \underset{t_{k-1}^{step}<t<t_k^{step}}{arg\ min}\ a_t^{step} \tag{3.12}$$

Additionally, step length sl_k is estimated using the concept of *vertical impact*, which is the difference in accelerations between a step's corresponding peak and its previous valley:

$$a_{pp,t}^{step} = a_{peak,t}^{step} - a_{valley,t}^{step}, \quad a_{peak,t}^{step} = h_t|_{t=t_k^{step}}, \quad a_{valley,t}^{step} = h_t|_{t=t_k^{head}} \tag{3.13}$$

Kang and Han's chosen step length estimation technique assumes that step length is proportional to the vertical impact $a_{pp,t}^{step}$. They propose two equations to calculate step length whose performance is identical. However, they observe that a logarithm-based step length estimation is slightly more accurate than a fourth root-based technique for larger step lengths, and vice-versa. As such, they decided to alternate between both equations using a threshold as a selection factor:

$$sl_k = \begin{cases} \beta_1 \sqrt[4]{a_{pp,t}^{step}} + \gamma_1, \ a_{pp,t}^{step} < a_{\tau}^{step} \\ \beta_2 \log(a_{pp,t}^{step}) + \gamma_2, \ a_{pp,t}^{step} \geq a_{\tau}^{step} \end{cases} \tag{3.14}$$

where a_{τ}^{step} is the threshold, which we set to 3.23 m/s^2 as was recommended. β is a scale factor and γ is an offset. For the fourth root function, our algorithms use $\beta_1 = 1.1$ and $\gamma_1 = -1.259$. Likewise, for the logarithm function we use $\beta_2 = 0.9$ and $\gamma_2 = 0.159$.

Finally, position updates use heading and step length information. For testing purposes, the starting position of the tag will be given to the algorithm in order to initialize the system. Then, at the kth step an updated position p_k^{user} is calculated by adding the step length sl_k to p_{k-1}^{user} in the direction of heading h_k. As shown in 3.15, this requires using trigonometry to extract the x-axis and y-axis components of h_k in order to add the correct displacement to each of the position's coordinates.

$$p_k^{user} = \begin{bmatrix} x_k^{GCS} \\ y_k^{GCS} \end{bmatrix} = \begin{bmatrix} x_{k-1}^{GCS} \\ y_k^{GCS} \end{bmatrix} + l_k \begin{bmatrix} \sin(h_k) \\ \cos(h_k) \end{bmatrix} \tag{3.15}$$

3.3 Sensor Fusion

Each of our chosen sensor technologies have unique properties in the context of localization. Our proposed fusion algorithm aims to improve upon the weaknesses of each technology by combining the data they provide.

Regarding Wi-Fi RTT, we want to make use of its ability to estimate absolute positions while mitigating its inaccuracy when faced with common challenges such as NLOS scenarios or simple network issues. Meanwhile, SmartPDR shows promise with its ability to act independently from the network through the use of motion sensors and its potential for good accuracy within short intervals. However, the positioning drift which occurs from the method's lack of mechanisms to obtain absolute positions is a weakness that causes the location error to accumulate over time.

We will now describe how we incorporate the aforementioned ideas in a sensor fusion algorithm, whose main ideas are illustrated in Figure 3.8.

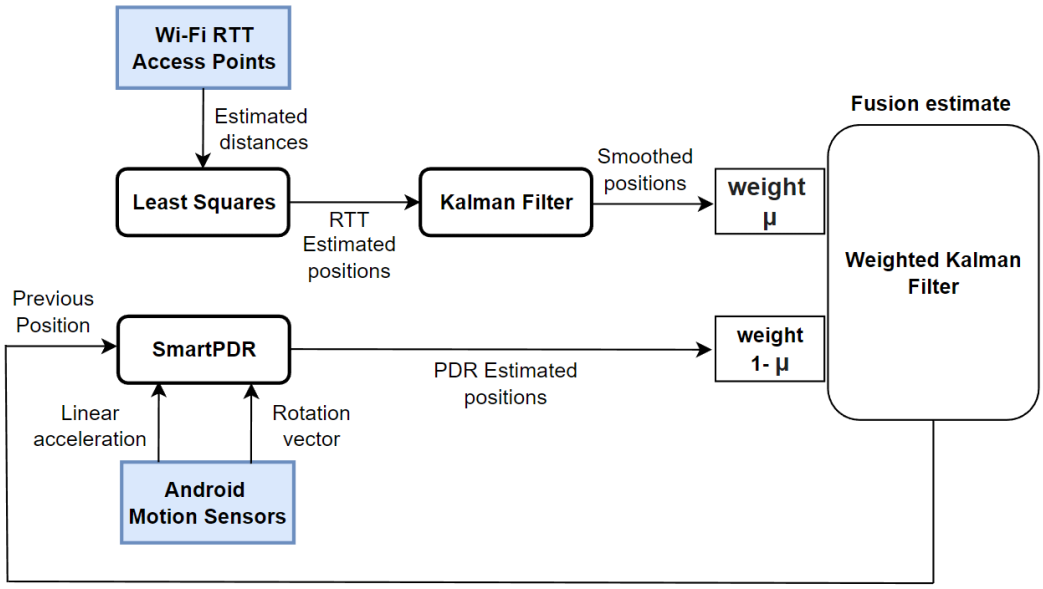


Figure 3.8: An overview of our proposed Sensor Fusion algorithm

The first step of our algorithm is to obtain base positioning estimates through the Least Squares and SmartPDR methods. For Wi-Fi RTT estimates, the results of Least Squares are then additionally passed through a Kalman Filter in order to smooth the data. The final result of this phase is two sets of estimates: a smoothed series of positions originating from Least Squares applied to Wi-Fi RTT and another series of positions coming from motion sensors through SmartPDR.

Data fusion can then be applied to these two estimate sets through the use of a Kalman Filter. The concept for the filter is similar to the one introduced in 3.1.2.3, where the filter is applied to the problem of an object moving in a two-dimensional plane. The key difference in the fusion variant lies in the state estimate update, which is shown in the following equation:

$$\hat{x}_t = \hat{x}_{t_p} + \mu\Big(K(z_t^{RTT} - H\hat{x}_{t_p})\Big) + (1-\mu)\Big(K(z_t^{PDR} - H\hat{x}_{t_p})\Big)$$

where at time t, z_t^{RTT} is the estimated position of Least Squares applied to Wi-Fi RTT data and z_t^{PDR} is the estimated position by SmartPDR. μ is a weight factor between the two methods, which for testing purposes we always set to $\mu = 0.5$. In theory, this means that all positions provided by the fusion algorithm are calculated in function of both technologies' estimates and according to the weight that was attributed to them. However, in practice this isn't possible without additional adjustments.

Wi-Fi RTT and SmartPDR are able to estimate positions at distinct frequencies. Meanwhile, every iteration the state update we defined requires entries from both technologies. As such, if at a certain timestamp a method has not estimated a position, the algorithm calculates its

expected position through linear extrapolation:

$$z_t^\tau = \begin{cases} z_t^\tau & , \ if \ z_t^\tau \in Z \\ z_{t-1}^\tau + v(t)\Delta t \ , & otherwise \end{cases}$$

where $Z = \{z_{t_1}^\tau, z_{t_2}^\tau, ..., z_{t_T}^\tau\}$ is the series of estimated positions by a method $\tau = \{RTT, PDR\}$ with $|Z| = T$. As described by equation above, the position of a moving object is extrapolated by finding its current velocity and applying that velocity to its previous position. In this case, the velocity and position that are used are the ones corresponding to the estimated positions of the method that is missing.

In its last step, the algorithm inputs the estimated fusion position into the next iteration of SmartPDR. The intent behind this adjustment is correcting positioning drift by mitigating the effects of relative positioning. This is accomplished by having new positions be calculated in function of fusion estimates, which introduces a component of Wi-Fi RTT's absolute positioning estimation into SmartPDR. Likewise, the mitigation of PDR drift should allow the method to better counterbalance phases where RTT positioning is imprecise due to its potential for high accuracy while it is unaffected by drift.

Chapter 4

Implementation

In this chapter, we discuss the implementation of all the software components necessary for the testing and evaluation of the proposed localization methods.

We discuss the implementation of an application for collecting Wi-Fi RTT and inertial sensor data. We also provide notes on the implementation of the actual localization algorithms through scripts meant to process the collected data.

4.1 Data Collection application

In this section, we discuss the system design and implementation details of our Android app for collecting motion sensor and Wi-Fi RTT data. We describe what kind of data it collects and the format of the logs produced for each type. We also mention the development environment, main packages used and required smartphone configurations to make the application work.

4.1.1 Dependencies

This app was developed in *Android Studio* using the Java programming language. The following list includes a summary of the necessary Java packages:

- Android has an extensive Wi-Fi RTT API, its documentation can be found at [13]. This package contains all necessary functions to perform Wi-Fi RTT ranging requests with compatible devices. A list of Android devices that support Wi-Fi RTT can be found at [10].
- As we detail in 4.1.2, ranging with RTT also requires that the Wi-Fi network is scanned for information about the devices currently within range. Using the WifiManager API [11] allows programs to get an information list of Wi-Fi access points that are visible from the device.

- The Android motion sensors API can be found at [1]. Functionalities of this API are used to collect information about rotation and acceleration of the smartphone.

Finally, Table 4.1 lists all the necessary Android permissions to enable all of the app's functionalities. These permissions must be given through the app's settings menu.

Permissions	**Functionality**
ACCESS_WIFI_STATE	Enable access to Wi-Fi network information.
ACCESS_FINE_LOCATION	Enable Wi-Fi scanning.
HIGH_SAMPLING_RATE_SENSORS ACTIVITY_RECOGNITION	Enable high performance usage of the Android motion sensors suite.
WRITE_EXTERNAL_STORAGE	To store the data logs in the phone's storage, to then later be exported.

Table 4.1: Permissions for the Android data collection app

4.1.2 System Design

The system is divided into two main data collection modules: the Wi-Fi RTT service module and the motion sensor module, which is composed of sub-modules for each motion sensor.

The Wi-Fi RTT module continuously performs ranging requests to access points in the network, passing the information of ranging results to a `FileOutputStream` class through callbacks for the purposes of writing logs in memory. Meanwhile, each Motion sensor sub-module actively listens for sensor events for their respective sensor type, passing them to log writers through callbacks as well.

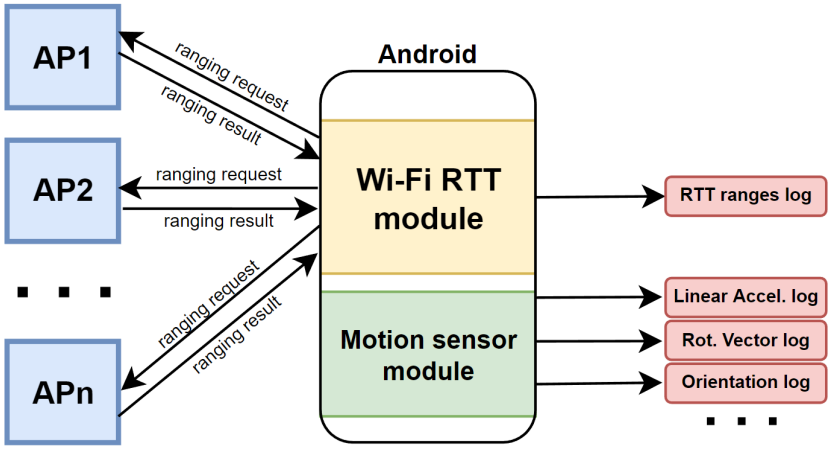


Figure 4.1: An overview of our data collection application.

Figure 4.1 illustrates the aforementioned modules through an overview of the system. Meanwhile, Figure 4.2 contains screen captures of the application's control panel.

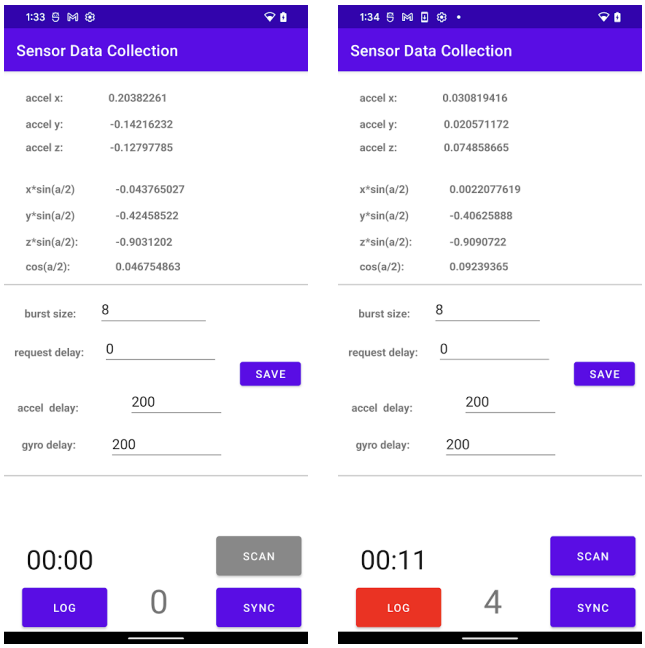


Figure 4.2: Control panel of the Data Collection app

In this section we expand on how each module performs its functions. We also discuss the challenges we found during their implementation process and our proposed solutions to these problems.

4.1.2.1 Ranging with Wi-Fi RTT

The main purpose of Wi-Fi RTT is to retrieve the estimated distance between each RTT-able AP and the device in which the app is running. This can be done by calling the `startRanging` function of the system's Wi-Fi RTT ranging service. This service requires the instantiating of a `WiFiRttManager` class through which its functions are called.

`startRanging` initiates a request to range to a set of devices specified in a `RangingRequest`. `RangingRequest` objects can be built by providing a list of `ScanResult` objects as an input, each result corresponding to a Wi-Fi RTT enabled device in the network. `RangingRequest.Builder` also allows for the configuration of the number of FTM packets used by the ranging process to estimate distance. This parameter, as mentioned in 2.1.3.1, is called RTT Burst Size and can be set to values between 2 and 31, with its default being 8.

`ScanResult` objects can be obtained by scanning the Wi-Fi network, which can be prompted using the `startScan` function of the `WiFiManager` class. The scan result list is filtered by iteratively using the `ScanResult.is80211mcResponder` function in order to remove all APs that aren't RTT-able.

The first implementation challenge associated with Wi-Fi RTT was deciding on how to retrieve and continuously update the `ScanResult` list. This is due to the introduction of Wi-Fi scan

throttling in Android 9, which has limited foreground applications to 4 scans every 2 minutes, with background applications being limited to 1 scan every 30 minutes. Limiting updates on the `ScanResult` list to 4 times every 2 minutes makes it so the delay between updates causes a disconnect between the list and the access points that are actually in range of our device.

Currently this problem can be overcome by disabling Wi-Fi scan throttling through developer options. However, in our experience, the application's scan result list of access points in range tends to remain inaccurate. This is due to the fact that, even with throttling disabled, Wi-Fi scanning's throughput is lower than the ranging frequency that the RTT service is capable of. If we sequentially alternate through scans and ranges, the scanning process will bottleneck the ranging process and therefore limit the frequency at which our localization solutions can provide estimates. If we alternatively run both functions in parallel, as APs enter and leave the range of the device there will be ranging requests that are sent to an inaccurate list of access points due to the comparative delay of the scan. These constraints necessarily impact the performance of Wi-Fi RTT localization in real-time. However, there are ways to overcome this that are mostly useful for testing purposes.

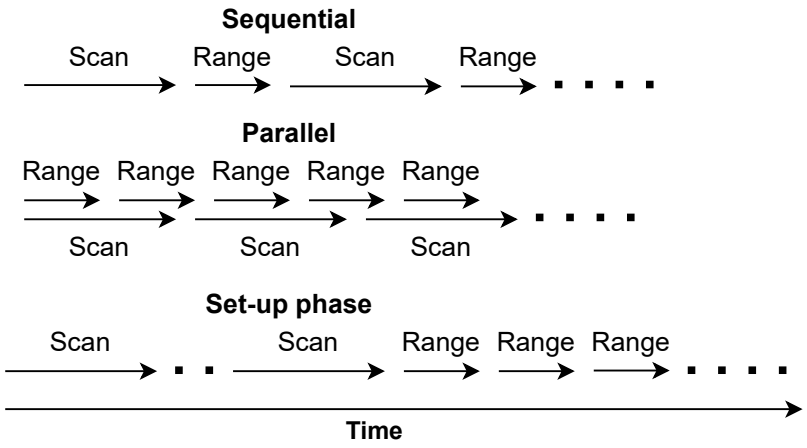


Figure 4.3: Ways to couple Wi-Fi scans and RTT ranging requests

Attempting to range with APs that are unable to respond has no drawbacks as long as the number of peers is within the amount specified by the `RangingRequest.getMaxPeers` function. By preemptively setting up a list with information about the access points we want to range with, we can circumvent the consequences of scanning delay by just continuously ranging with all APs. In Android 13, this was made possible by the `RangingRequest.Builder.addResponders` which accepts `ResponderConfig` objects. These can be built by providing MAC addresses and channel frequency information of the desired APs. However, this option was not available to us before implementation and testing was concluded due to that Android version not having been released at the time.

Our approach at preparing a preset list of access points requires a set-up phase for each experiment. As illustrated in 4.3, before beginning any test runs it is necessary to move through

the test space and prompt for Wi-Fi scans. Every Wi-Fi scan adds any newly found RTT-able access points to the `ScanResult` list. The set-up phase concludes when we verify that all the needed access points have been added. We are then able to perform any number of test runs without any additional set-up until the application is reset.

4.1.2.2 Collecting motion sensor data

Accessing Android's sensor service can be done through a `SensorManager` object using the `SensorManager.getDefaultSensor` function. By specifying a sensor type (a list of which can be found in [1]), the function returns a `Sensor` object of that type.

To effectively start getting sensor data, the system must register a listener for that `Sensor` using the `SensorManager.registerListener` function. The input contains three arguments: a `Sensor` object, an integer representing the desired sensor delay, and a `SensorEventListener`.

A `SensorEventListener` can be built by specifying behaviour for its two main functions: `onSensorChanged` which is called when there is a new sensor event and `onAccuracyChanged`, called when the accuracy of the sensor has changed. We use the `onSensorChanged` function to write the values of each `SensorEvent` to a file. On the other hand `onAccuracyChanged` currently has no relevant defined behaviour, although we consider that it could be useful for real-time localization purposes.

4.1.3 Data Types

The purpose of the application is collecting localization-enabling information from different technologies. In this section, we list all the data types that are collected, defining their parameters and presenting the structure of the data logs that the app outputs.

4.1.3.1 RTT ranging results

Wi-Fi RTT data can be collected from the components of a `RangingResult`, an object that contains all the information resulting from a RTT `RangingRequest` between the device and a single access point. To be more specific, ranging requests are built to contain the entire list of access points to range with. In association, the callback of the ranging request returns a list of ranging results, each one for an AP that was specified.

Table 4.2 displays the format for a log entry corresponding to ranging results between a device and a single AP at a point in time.

Ranging results carry the estimated distance between the device and the result's corresponding AP. As previously discussed, this resulting distance is in fact the mean of distances estimated

Field	Description
`timestamp`	Timestamp of the sensor event, value of `RangingResult.getRangingTimestampMillis`.
`mac`	MAC address of the access point of this ranging result, value of `RangingResult.getMacAddress`.
`distance`	Distance estimated by Wi-Fi RTT in milimeters, value of `RangingResult.getDistanceMm`. Uses the mean of multiple FTM packets - the amount of which is specified by the Burst Size parameter.
`std_dev`	Standard deviation of the distance (in milimeters) that is calculated over the measurements that are successfully inside a RTT burst, value of `RangingResult.getDistanceStdDevMm`.
`rssi`	Average RSSI, in *dBm*, detected between the device and the access point during a RTT measurement, value of `RangingResult.getRssi`.
`burst_size`	Number of attempted measurements to calculate the distance and standard deviation for this RTT exchange, value of `RangingResult.getNumAttemptedMeasurements`.
`successful_in_burst`	The number of successful measurements in this RTT burst, value of `RangingResult.getNumSuccessfulMeasurements`

Table 4.2: Log format for Wi-Fi RTT Ranging Result data

through multiple Wi-Fi FTM packets, the amount of which is specified by the RTT burst size parameter. Ranging results also carry information which aims to serve as indicators of the quality of the estimate: the burst size that is set, how many packets were successfully transmitted in that burst, the RSSI between the device and the AP and the standard deviation associated with the calculated mean of the distance estimate.

4.1.3.2 Linear Acceleration

Linear acceleration information is collected from the `Sensor.TYPE_LINEAR_ACCELERATION` sensor type. It is represented by a three-dimensional vector that indicates acceleration along each of the smartphone's axis (in m/s^2). Gravity is already filtered from these values.

Log entries of this data type have the format shown in Table 4.3.

Field	Description
`timestamp`	Timestamp of the sensor event, output by `SensorEvent.timestamp`.
`ax`	Acceleration at the x-axis, following smartphone's coordinate system, value of `SensorEvent.values[0]`.
`ay`	Acceleration at the y-axis, following smartphone's coordinate system, value of `SensorEvent.values[1]`.
`az`	Acceleration at the z-axis, following smartphone's coordinate system, value of `SensorEvent.values[2]`.

Table 4.3: Log format for Linear acceleration sensor type

4.1.3.3 Rotation vector

Rotation vector information is collected from the sensor type `Sensor.TYPE_ROTATION_VECTOR`. It represents smartphone orientation as a unit quaternion [42], which is a representation of a three-dimensional axis around which the device is rotated through by a certain angle.

Quaternions are useful units for expressing rotation because all that is needed to define them is an axis (x, y, z) where $x^2 + y^2 + z^2 = 1$ and the angle θ of the rotation around that axis. They also have a well-defined interpolation that is easy to implement. The elements of a quaternion are $x \times sin(\theta/2)$, $y \times sin(\theta/2)$, $z \times sin(\theta/2)$ and $cos(\theta/2)$.

The referential used for rotation vector representation is the Earth coordinate system previously illustrated in Figure 2.5. Table 4.4 displays the format of log entries of this data type.

Field	Description
`timestamp`	Timestamp of the sensor event, output by `SensorEvent.timestamp`.
`rx`	$x \times sin(\theta/2)$, first component of a unit quaternion, value of `SensorEvent.values[0]`.
`ry`	$y \times sin(\theta/2)$, second component of a unit quaternion, value of `SensorEvent.values[1]`.
`rz`	$z \times sin(\theta/2)$, third component of a unit quaternion, value of `SensorEvent.values[2]`.
`rw`	$cos(\theta/2)$, fourth component of a unit quaternion. Output by `SensorEvent.values[3]`.

Table 4.4: Log format for Rotation vector sensor type

4.1.3.4 Orientation

Orientation type data can be collected by using the function `SensorManager.getOrientation`. This computes the device's orientation based on a

rotation matrix [44] as an input, which can be obtained by applying the `SensorManager.getRotationMatrix` function to a unit quaternion. As described in 4.1.3.3, unit quaternions are obtainable through the rotation vector sensor type.

Orientation data is expressed in Euler angles [45], three angles used to describe orientation of a body with respect to a fixed coordinate system. Android specifically uses the *azimuth-pitch-roll* definition of Euler angles:

- *Azimuth*, the angle between the magnetic north and the y-axis, around the z-axis, ranges from 0 to 359 degrees.
- *Pitch*, rotation around the x-axis, ranges from -180 to 180 degrees, increasing values as the z-axis moves in the direction of the y-axis.
- *Roll*, rotation around the y-axis, increases clockwise, ranges from -90 to 90 degrees.

Table 4.5 displays the format for log entries of this data type. In this table, we consider thats `SensorManager.getOrientation` returns a `float` array of size 3.

Field	**Description**
`timestamp`	Timestamp of the sensor event, value of `SensorEvent.timestamp`.
`azimuth`	Device angle around the z-axis with respect to the magnetic north, in degrees, value of `SensorManager.getOrientation()[0]`.
`pitch`	Device angle around the x-axis, in degrees, value of `SensorManager.getOrientation()[1]`.
`roll`	Device rotation around the y-axis, in degrees, value of `SensorManager.getOrientation()[2]`.

Table 4.5: Log format for Orientation data

4.1.3.5 Synchronization points

Synchronization points are pairs of timestamps and coordinates. As detailed further in 5.1, they are used in the process of characterizing (by approximation) the true movement of the device during a test turn.

The app's implementation assumes that synchronization points are always travelled through in the same order. It requires the user to press a button in the app as they reach each synchronization point. When the button is pressed, the app retrieves the timestamp for that instant and adds it to the sequence of timestamps in the log.

Association of timestamps and coordinates is done *a posteriori* through offline scripts. That being the case, the log format for this data type is a ordered list of timestamps, each corresponding to instants when the user reached a synchronization point.

4.1.3.6 Step Counter

For the purposes of validation, data was collected from Android' Step Counter sensor type. This was used during implementation to assess if our algorithm for step detection was coherent in comparison to Android's step detection algorithm.

Sensor events of this type output a timestamp each time the algorithm detects a step. Therefore, the log format is a list of timestamps associated with the occurrence of steps.

4.2 Data processing scripts

A solid framework was required to be able to effectively develop, test and evaluate all the proposed methods along with their possible tweaks and variations. Speed and simplicity in the implementation component was required so that more time could be invested in conceptualizing and evaluating methods.

In this section, we provide notes about the process of making the data processing scripts we used to implement our algorithms and structure, visualize and interpret the information that they output.

4.2.1 Dependencies

The *Python* programming language was chosen for development due to its easy access to useful packages for the purposes of data science, data visualization and mathematical solvers. It was used inside the *Jupyter Notebook* [4] application to facilitate debugging and evaluation of methods through the interactive output of tables, plots and graphs.

As for Python packages, we used the *Python Data Analysis Library* (pandas) [7] for its accessible data analysis and processing tools together with *Matplotlib* [5] for creating data visualizations. Another needed package was *Numpy* [6] which is usually fundamental in programming in Python due to its mathematical functions, data structures and generators. Additionally, we borrowed the Least Squares solver from the *SciPy* [9] package and utilized the functions of *PyQuaternion* [8] package to represent quaternion rotation data and use those quaternion representations to apply rotations to other data structures.

4.2.2 Structuring data

The logs that are produced by our Data Collection application come in CSV files. To process this data, these files must first be read and their contents must be transformed into data structures we can manipulate. We chose to use data frames from the pandas package for this purpose, using the pandas `read_csv` function to read and convert the CSV data.

Data frames are tabular data structures with labeled rows and columns, allowing for columns to have different types of data. They are particularly useful for multiple reasons. For example, arithmetic operations can align on both columns and rows if their stored data types are compatible, giving developers simple ways to transform data. They can also be used through queries much like a database.

Lastly, they provide easy to use functions to reshape the data frame into convenient formats for data processing and analysis. For example, the `pivot_table` pandas function allowed us to reshape Wi-Fi RTT data into a format that is easier to integrate with the SciPy Least Squares solver.

As described in 4.1.3.1, Wi-Fi RTT data logs are organized by timestamps, with each line corresponding to a single AP's reported data. Meanwhile, to use Least Squares we need the set of distances for all APs in a ranging result. For this purpose, using Wi-Fi RTT data to create a table where a row has all reported distances for a specific timestamp (each column corresponding to an AP's distance) provides a much more efficient data structure to iterate.

4.2.3 Processing data

Most methods described in chapter 3 were implemented from scratch by resorting to simple functions of the NumPy, pandas and PyQuaternion packages. The only exception that should be highlighted is the usage of the Least Squares solver from the SciPy package.

The function `scipy.optimize.least_squares` requires the input of an initial guess for the solution of a least squares problem and a function that describes the problem through the target variables' squared residuals. For our localization problem, the initial guess corresponds to the latest estimated position. The function parameter is described through a list of equations, one for each access point:

```
def equation_list(guess):
        x, y, z = guess
        global aps
        global sample_range
        L = [ (x - ap['x'])**2 + (y - ap['y'])**2 + (z - ap['z'])**2
             - sample_range[ap['mac']]**2
            for _,ap in aps.iterrows() if sample_range[ap['mac']] > 0]
        return L
```

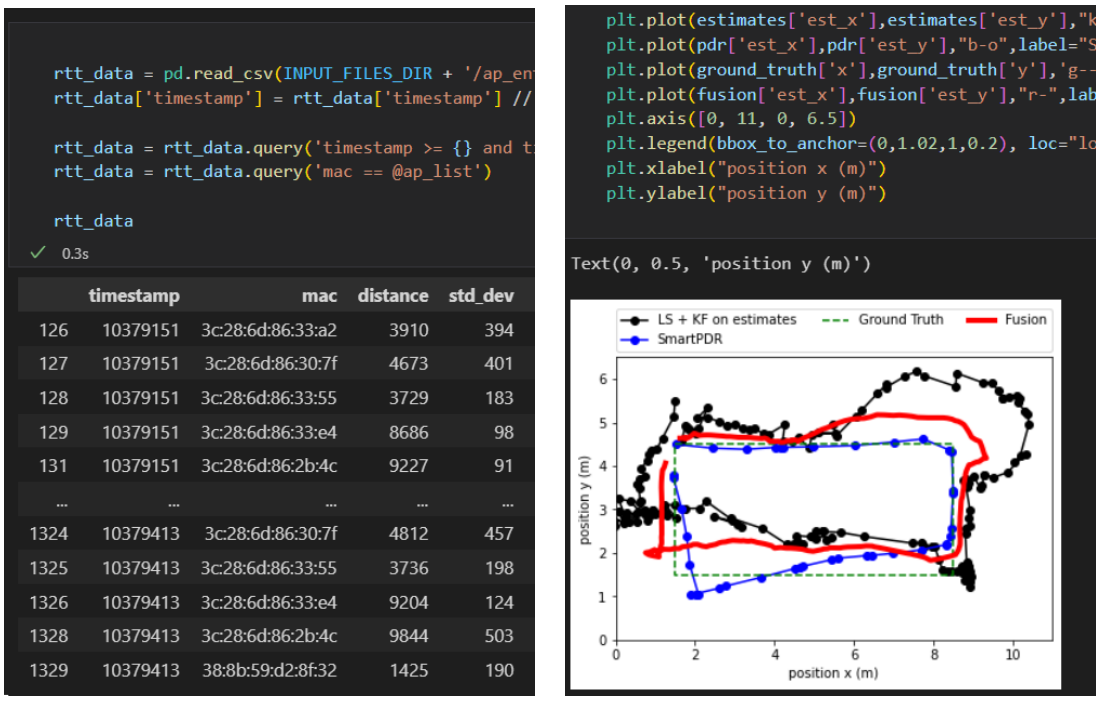
Where `aps` is a pandas data frame with three-dimensional coordinates for each access point and `sample_ranges` is a row of a data frame with reported distances grouped by timestamp, as described in the example of subsection 4.2.2. This function must be used for every set of ranging results that was collected over time. At each iteration, it returns an array that represents the three-dimensional position that the solver calculated to be optimal for the input set of reported distances.

4.2.4 Visualizing data

Evaluating, debugging and demonstrating our methods requires visualizing the relations between the data. For the most part, we do this by creating plots and tables that organize relevant information, taking advantage of pandas data frames and their developed integration with other Python tools.

When running an algorithm, we first store the outputs of our methods as two-dimensional arrays. We do this line by line, with each line entry containing all the relevant output for that iteration. We then convert these arrays into pandas data frames by specifying labels for the array's columns. Therefore, it is important that the contents within each line always follow the same order.

Then, plot generation is done through *Pyplot*, Matplotlib's Python interface, which is easily compatible with pandas data structures. A pandas data frame can be seen as a dictionary for series data structures. If we extract two series out of a data frame, Matplotlib will generally accept them as input for its plotting functions as long as their sizes match and their data types are compatible.



	timestamp	mac	distance	std_dev
126	10379151	3c:28:6d:86:33:a2	3910	394
127	10379151	3c:28:6d:86:30:7f	4673	401
128	10379151	3c:28:6d:86:33:55	3729	183
129	10379151	3c:28:6d:86:33:e4	8686	98
131	10379151	3c:28:6d:86:2b:4c	9227	91
...	...	...	...	...
1324	10379413	3c:28:6d:86:30:7f	4812	457
1325	10379413	3c:28:6d:86:33:55	3736	198
1326	10379413	3c:28:6d:86:33:e4	9204	124
1328	10379413	3c:28:6d:86:2b:4c	9844	503
1329	10379413	38:8b:59:d2:8f:32	1425	190

Figure 4.4: Example of data visualization in Jupyter Notebook: inline HTML tables (left) and inline plots (right).

Finally, Jupyter Notebook integrates with Matplotlib and Pandas to provide the inline output of images and HTML tables, as shown in Figure 4.4. Matplotlib plots are automatically displayed after the code block where they are specified. Meanwhile, Jupyter Notebook has a defined conversion of pandas data frames to HTML tables, allowing developers to call for them to be displayed after their respective code block.

Chapter 5

Evaluation

This chapter will contain all information pertaining to our testing process and evaluation of the results. First, we will describe the methodology we use to evaluate the performance of the implemented localization methods.

We then go over all our experimental set-ups, giving insight on how they were planned and characterizing all relevant specifics.

Finally, we evaluate all implemented localization methods using our stated methodology for assessing performance. We compare the algorithms to each other and highlight any strengths and weaknesses found through testing.

5.1 Methodology

We will now go over our methodology for planning experimental set-ups that ensure the data that's collected is sound for being compared and evaluated, as well as our chosen metrics for evaluating performance.

5.1.1 Evaluation method

Testing the performance of our proposed solutions requires having a system we can use to reproduce very similar testing conditions (which ideally would be identical) across different runs of an experiment.

Being able to reproduce testing conditions allows us to meaningfully compare the quality of our solutions. The error of a solution's estimates is one of the most important factors when evaluating its quality. By ensuring we can reproduce our testing configurations, a meaningful way we can evaluate localization methods is by:

- Comparing the error inherent to different solutions in the same testing configuration. This allows us to have an idea of how the performances of different methods compare to each other in a controlled scenario.
- Comparing the same solution in distinct testing configurations. Sharp variations of error across set-ups might put into evidence if the method has any clear weaknesses.

In the context of localization, the results that are produced are a series of estimated positions over a time period. That being the case, defining a reproducible referential system for localization implies:

- Deciding on a coordinate reference system for the spaces that are used.
- Attributing each relevant element of the testing environment coordinates in the space.
- Defining the evolution of the tag's movement through the testing space over time, describing it through a series of coordinates associated with periodic timestamps.

We call the progression of the target entity's movement the *ground truth*. Using the ground truth, we define the error of a solution in a point in time as the Euclidean distance between the estimate and its corresponding point in the ground truth:

$$\text{error}(t) = d(\text{estimated\_position}(t), \text{ground\_truth}(t)) \tag{5.1}$$

where the Euclidean distance is given by:

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \tag{5.2}$$

If we calculate the error of a method for a series of time instances in each run, it is possible to find the average, minimum and maximum error for that run. We can then aggregate all runs for a specific testing set-up and become able to not only calculate average, minimum and maximum error but also other important metrics like the standard deviation, median and quartiles.

5.1.2 Approximation of the Ground Truth

The *ground truth* is a periodic series of position coordinates associated with a timestamp. We use it as a way to approximate the true movement of a tag during a testing run.

It is impossible to perfectly describe the actual movement of an entity, independently of the method used to attempt it. All approaches are limited. Their quality is closely tied to the resources available to collect and store data about the movement.

Our approach to solving this issue relies in points of synchronization gathered in the data collection process for each test run.

A synchronization point list, much like the ground truth, is a series of position coordinates associated with a timestamp. Each entry in the list represents an accurate position of the entity at a point in time during a test run. There are, however, two main issues:

- The entries need to be taken through human input. It is not possible for them to be perfectly periodic.
- The entries usually aren't frequent enough to match the frequency of the localization method's estimates. This makes it so that, in their base state, they aren't useful enough for positioning error calculations.

Transforming a synchronization point list into a ground truth series can be achieved through *linear interpolation*. Applying this method requires that we assume that the movement between two synchronization points is linear. Although the test routes are planned for linear movement between sync points, there is necessarily a margin of error that comes with this simplification.

The application of linear interpolation to a two-dimensional position is done through the following formulation: for a timestamp t that is between timestamps t_i and t_{i+1} corresponding to synchronization points p_i and p_{i+1}, the approximated position of the tag at instance t is

$$x_t = x_i + f * (x_{i+1} - x_i) \tag{5.3}$$

$$y_t = y_i + f * (y_{i+1} - y_i) \tag{5.4}$$

where f is given by:

$$f = (t - t_i)/(t_{i+1} - t_i) \tag{5.5}$$

The only requirement for linear interpolation to function correctly is that the collected synchronization entries need to be strategically timed and/or frequent enough that the actual progression of the route over time is accurately described through linear movements between synchronization points.

5.2 Setup

The planning process for experimental setups requires that all relevant objects in the testing space are properly identified and have coordinates attributed to their position. This includes all devices we intend to place in the space or all the already existing physical obstacles expected to already be in it.

The following list contains all the considered objects:

- Wi-Fi RTT Access Points - Google Wifi Home routers [3];
- Points of Synchronization for the purposes of approximating the ground truth;
- Defined route for the tag to travel;
- All walls and obstacles must be properly identified.

The remainder of this section's purpose is to provide details on all the testing routes we used to gather experimental data. For each route, we will provide a plan scheme of the route and describe any additional characteristics about the testing space, device layout or movement of the tag that may be relevant for evaluation.

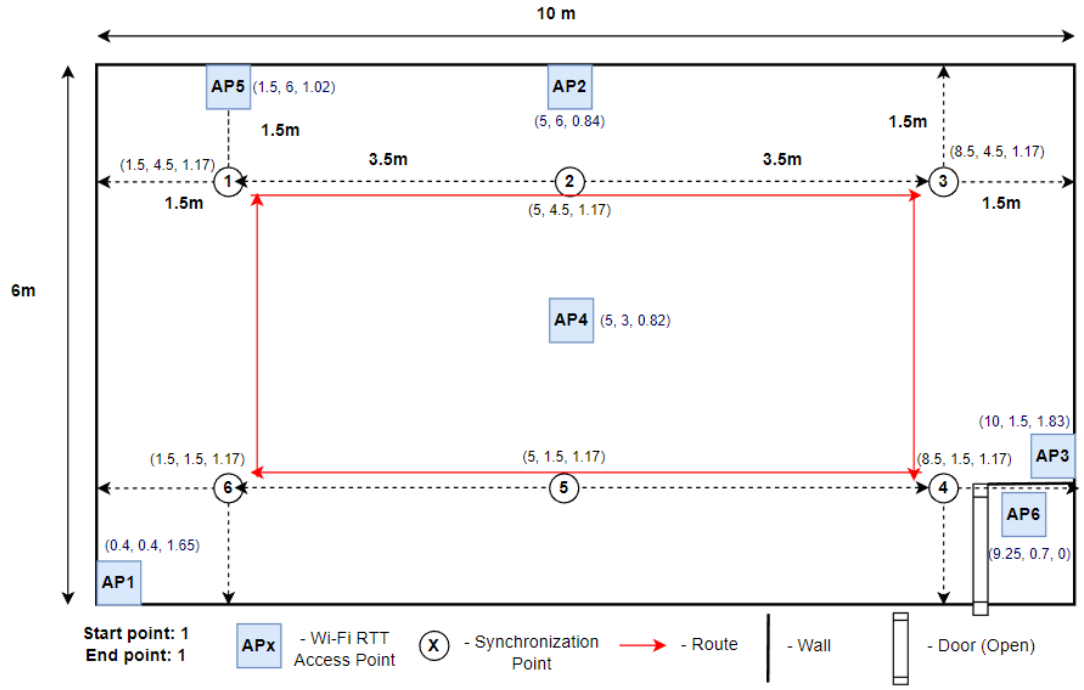
5.2.1 Routes

In our planned routes (Figure 5.1), the test space's x axis ranges between 0 and 10 meters, expanding from left to right in the schemes. The y axis ranges between 0 and 6 meters, expanding from bottom to top in the schemes. Furthermore, each of the schemes also defines the synchronization point corresponding to the starting position used to initialize the SmartPDR algorithm as well as the planned end point for each route.

The total distance covered by Route 1 is 20 meters with 3 total turns of direction. The average run duration between all test runs in this route is around 29 seconds. This means an average walking speed of around 0.69 meters per second, with no stops. Furthermore, line of sight for access points 3 and 6 doesn't exist between synchronization points 2 and 4.

The total distance covered by Route 2 is 13.89 meters with 3 total turns of direction. The average run duration between all test runs in this route is around 21 seconds. This means an average walking speed of around 0.66 meters per second, with no stops. Furthermore, line of sight for access points 3 and 6 doesn't exist near synchronization point 7 and between synchronization points 8 and 4.

Route 1



Route 2

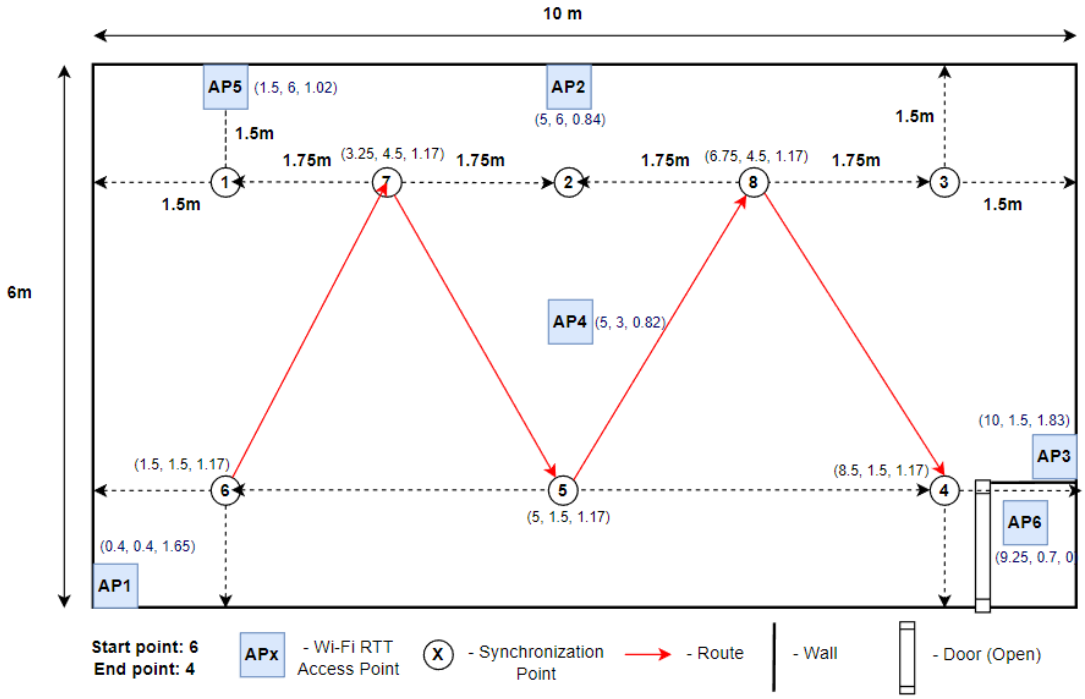


Figure 5.1: Route plans for experimental test runs

5.3 Results

For each route, we collected results from 7 test runs where the routes were travelled through once per run. We applied each method to every run separately and aggregated the error data in order to derive statistics. A compilation of these statistics for data collected with a Google Pixel 4 smartphone device and a set of Google Wifi Home routers [3] can be found in Figure 5.2, represented through each method's cumulative distribution function.

Through the remainder of this section, general considerations and additional notes on this information will be given before going into detailed discussion about specific topics in subsequent sections.

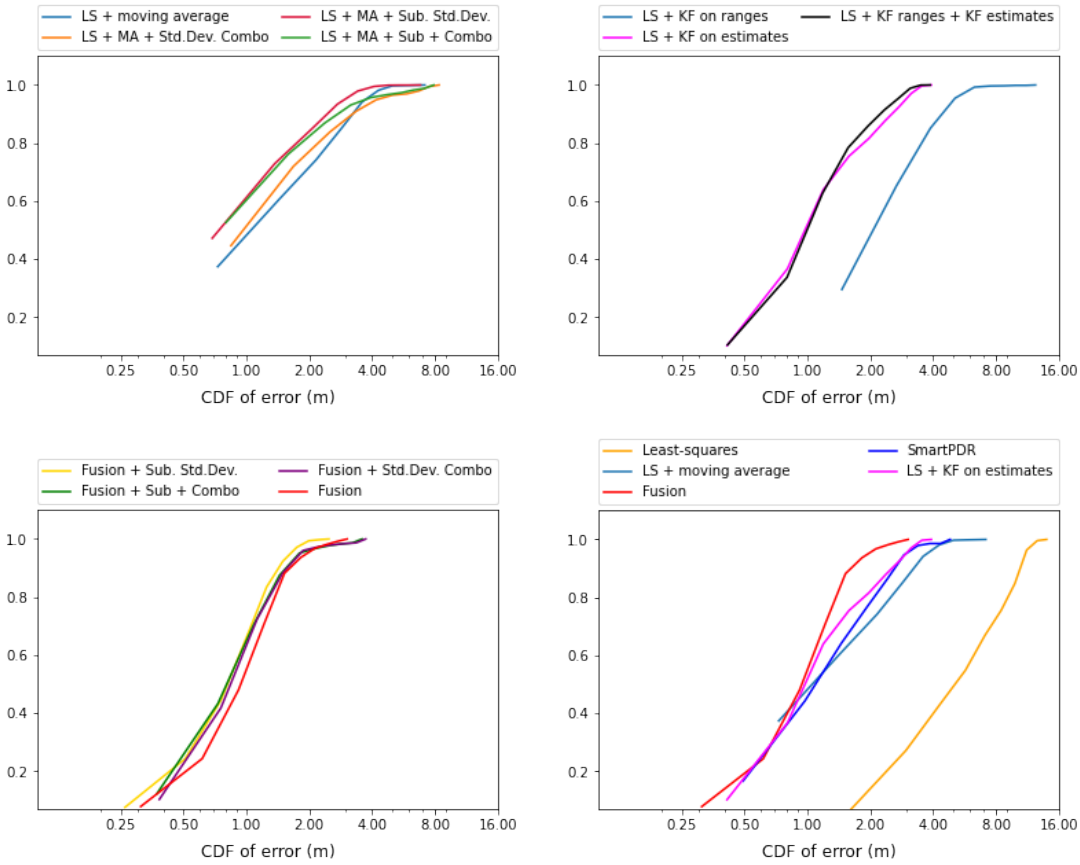


Figure 5.2: Cumulative Distribution Functions for localization methods in Google Pixel 4

Applying Least Squares without data smoothing doesn't provide great results. In both routes the average error on methods without smoothing ranges around 5 to 6 meters while methods with smoothing lower this to 1.3m for the Least Squares estimate smoothing technique and 2.5 meters for AP ranges smoothing technique.

Using a Kalman Filter to smooth distances reported by Access Points is less effective than applying the filter directly to Least Squares' estimated positions. On average, in both routes the distance smoothing approach presents errors that are at least 1 meter larger than estimate

smoothing.

An alternative approach that applies a second filter to estimates of Least Squares using smoothed distances results in an error that is more similar to estimate smoothing. However, this is more complex while generally offering no improvements in precision.

As for SmartPDR, the general impression is that the magnitude of its error is in the same value range as smoothed Least Squares methods for Route 1. However, this doesn't hold true for Route 2 even if the average error for SmartPDR between both routes (1.35m in Route 1 compared to 1.42m in Route 2) doesn't change significantly.

Our proposed fusion algorithm looks to be the more accurate solution in both routes. In route 1, its error belongs in a distinct value range from all other methods, with fusion showing an average error of 0.99 while Kalman Filter smoothing averages 1.3 meters. Meanwhile, in route 2 the gap between smoothed Least Squares approaches and Fusion is smaller, with fusion averaging at 0.92 meters error and Kalman Filter smoothing approaches staying just under 1 meter on average.

The average error for Fusion is consistent in both routes, staying under 1 meter with outliers that can go to a maximum of around 3 meters. That being the case, the smaller difference between fusion and smoothing methods are attributed to smoothed Least Squares performing better in Route 2's conditions . The same logic can be applied to the aforementioned disparity between SmartPDR and Least Squares methods in Route 2. This makes sense because Route 2 has less of its length under NLOS conditions.

Finally, the proposed optimizations that use the standard deviation reported by APs to compensate for multipathing have a positive impact. Applying these techniques generally improves precision regardless of usage of data smoothing or fusion.

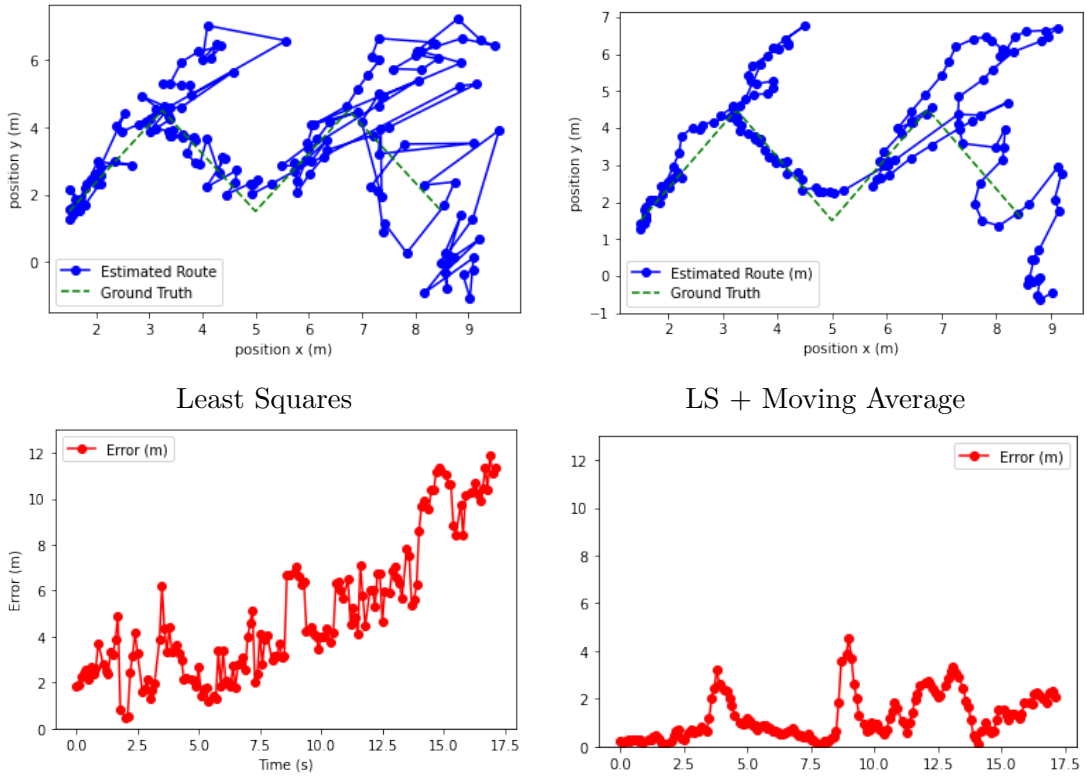
The variant where standard deviation is subtracted from reported AP distances performed better than using standard deviation as a criteria for choosing subsets of AP. In fact, subtracting standard deviation seems to even outperform an approach where both optimizations are used. The two options conflict because in situations where the reported distance is close to the true distance, subtracting standard deviation results in a poorer distance estimate. In turn, choosing subsets of APs tries to create more instances where the reported distances are good.

5.3.1 Least Squares

The impact of data smoothing on Least Squares' accuracy is an interesting matter to study further. Figure 5.3 showcases data smoothing on an example of a relevant test run.

By themselves, Least Squares estimates have a lot noise that is related to the usage of Wi-Fi RTT data. APs can report distance data several times per second, with every report having value fluctuations that impact estimates, manifesting as sharp jumps in positioning.

To demonstrate, smoothing with a technique as simple as a 1 second moving average over the



Least Squares | LS + Moving Average

Least Squares error | LS + Moving Average error

Figure 5.3: Example of the impact of data smoothing on Least Squares estimates

reported ranges already results in a significant error reduction. Unlike more robust Kalman Filter approaches, a moving average doesn't even fully take into account the tag's previous position and movement. Despite that, in this example the average error goes down from around 5.05 meters to around 1.2 meters.

Another matter we considered was how additional data reported by Wi-Fi RTT APs could be used in order to improve Least Squares' performance. Issues with reported distance caused by poor AP positioning or multipath propagation in NLOS scenarios are a major source of localization error. Therefore, we set out to find criteria to evaluate the distances reported by each AP.

Figure 5.6 illustrates an experiment where we found reported standard deviation to be an acceptable indicator of problems in the corresponding reported distances. As exemplified in the figure, there's a degree of correlation between phases where standard deviation is especially unstable and moments where estimated distances considerably deviate from true distances. Furthermore, through this experiment we can also confirm that problematic ranging results estimate distance in excess compared to the ground truth.

This experiment served as a basis for the proposed optimizations described in 3.1.2.5, where reported standard deviation is subtracted from reported distances or used as a criteria to pick the subsets of APs whose reported distances are used for solving Least Squares.

5.3.2 SmartPDR

One of the problems that we aim to prevent is positioning drift in Dead Reckoning methods, which occurs due to cumulative errors of step detection, stride length and heading estimation. Figure 5.4 is an experimental run that illustrates how PDR drift can impact the precision of localization.

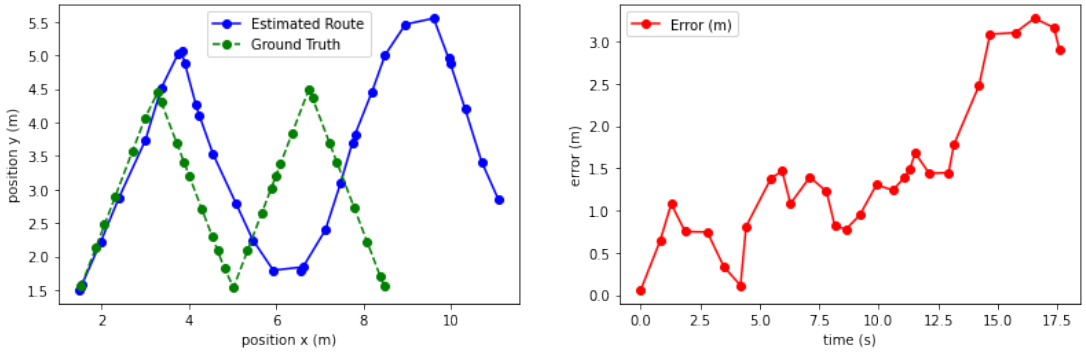


Figure 5.4: Example of drift in Pedestrian Dead Reckoning

As illustrated, faulty position estimates tend to also make error increase in posterior estimates. This is because new positions are calculated in function of previous ones, and PDR algorithms have no mechanisms to periodically correct this. As such, testing ways to re-calibrate a tag's current position is important to combat drift, which is a factor that is explored in our proposed fusion algorithm.

Additionally, to test our step detection technique we collected step counter sensor data from the Android API (described in 4.1.3.6). The results show that our algorithm's step detection lines up fairly well with Android's algorithm.

On average, SmartPDR detects 35 steps and Android Step Counter detects 33 for Route 1. In Route 2, SmartPDR detects an average of 24 while Android Step Counter detects 21. These values are within an acceptable margin of difference to validate our implemented approach.

5.3.3 Fusion

Sensor fusion aims to improve upon each sensor's weaknesses by combining their data. As such, in our proposed fusion method it is important to assess how SmartPDR and Least Squares can complement each other.

Figure 5.5 expands on the example of Figure 5.6 in order to show the fusion algorithm in action and exemplify its tendencies. This is an experimental route where we have observed reasonable degrees of inaccurate reported distances by each AP, resulting in phases where Least Squares estimates are quite imprecise. A small degree of PDR drift also begins to show itself in the second half of the run.

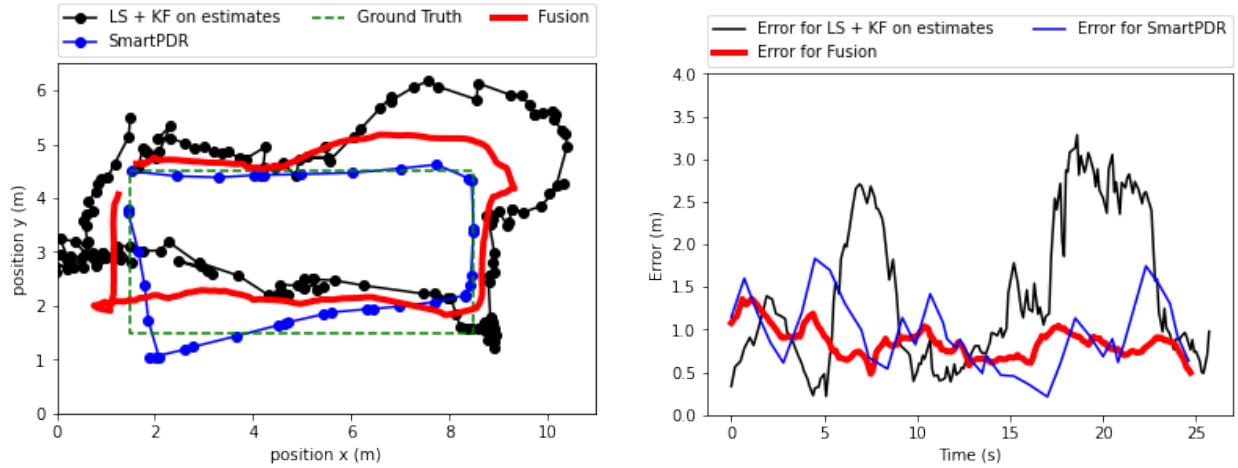


Figure 5.5: Fusion method applied to the test run of Figure 5.6.

We have observed that the smoothing effect of the Kalman Filter, combined with the additional Least Squares information tends to be effective at correcting heading drift. In turn, the influence of a corrected SmartPDR improves estimates in phases where Least Squares estimates significantly deviate from the truth. This happens because SmartPDR is not affected by sub-optimal AP positioning or multipathing in NLOS scenarios.

In general, the aforementioned observations don't hold true only for some situations where both methods provide bad estimates. Intuitively, our algorithm relies on the fact that both components actively correct each other. As such, if both are not working properly it is likely this constant mutual calibration will not be able to properly approximate its estimates to the ground truth.

5.3.4 Experiments with a different device

We also ran our algorithms on data collected with a Xiaomi Mi 9T smartphone device. With no adjustments, the quality of the results was poor. As we will elaborate in this section, the precision of Xiaomi Mi 9T's estimates was disproportionately worse for both Least Squares and SmartPDR methods. This lead us to attempt to discover why that is the case.

Google Pixel 4 supports versions of Android from 10 to 13. It still receives firmware updates as of September 2022. Meanwhile, the Mi 9T supports Android 9 to 11 and stopped receiving firmware updates in August 2021. We will now elaborate on why this can be one of the contributing factors for the disconnect in results between the two devices.

When it comes to Wi-Fi RTT data, the Xiaomi 9T exhibits a bias in the estimated ranges that isn't present in the Pixel 4 anymore. Figure 5.7 is an experiment we ran which shows estimated distances in the 9T are always at least around 2.5 meters shorter than true distances under 10 meters. Meanwhile, a bias of 1 to 1.4 meters previously existed in the Pixel 4 but has largely been corrected.

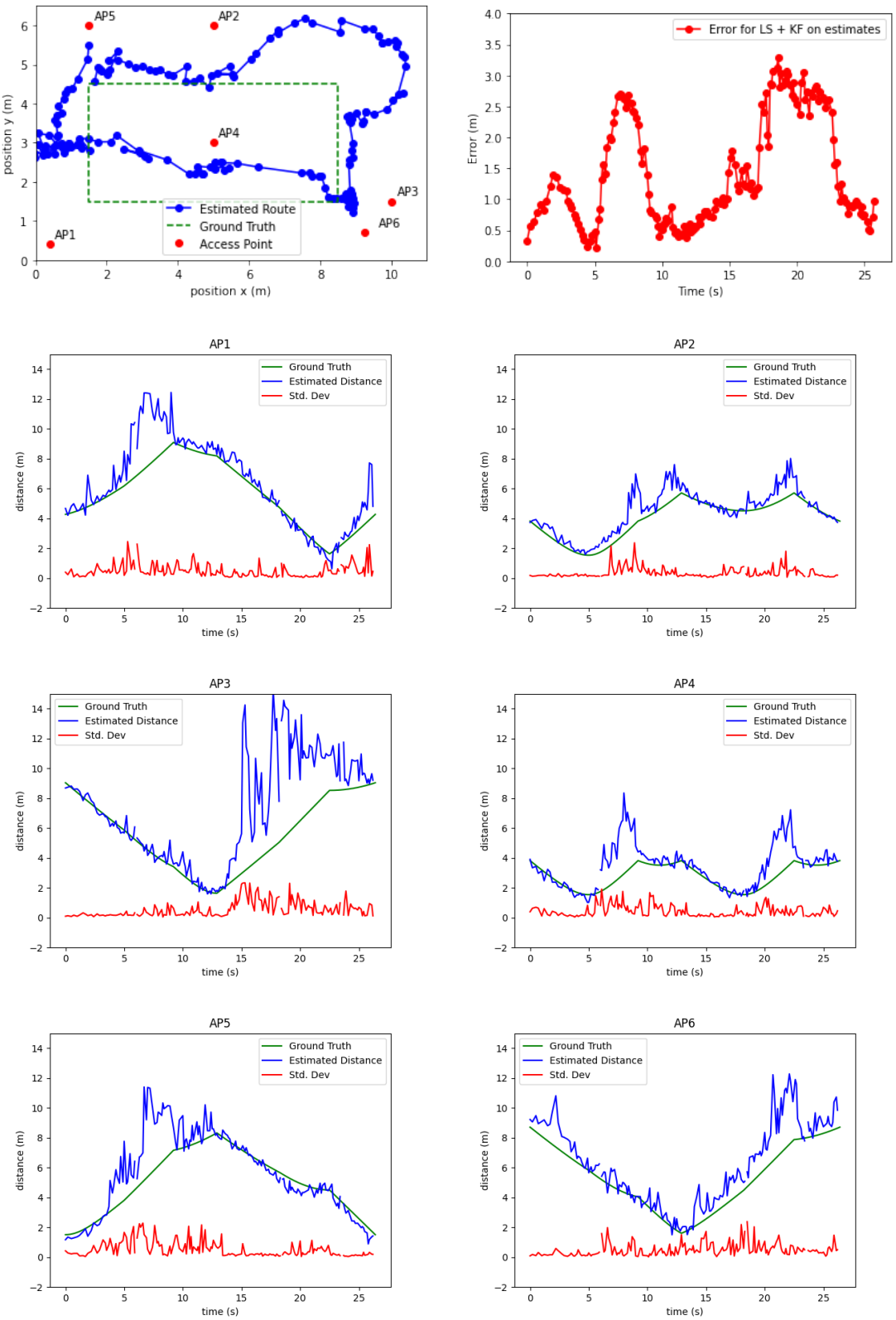
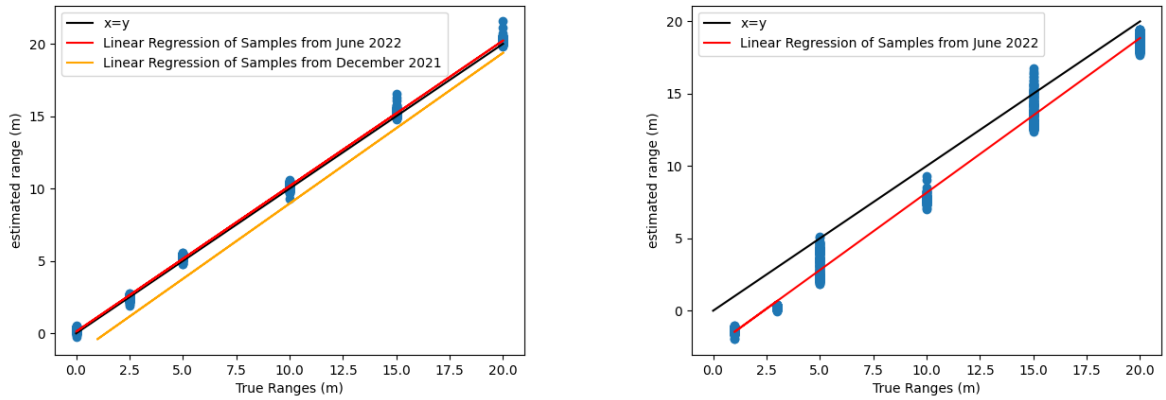


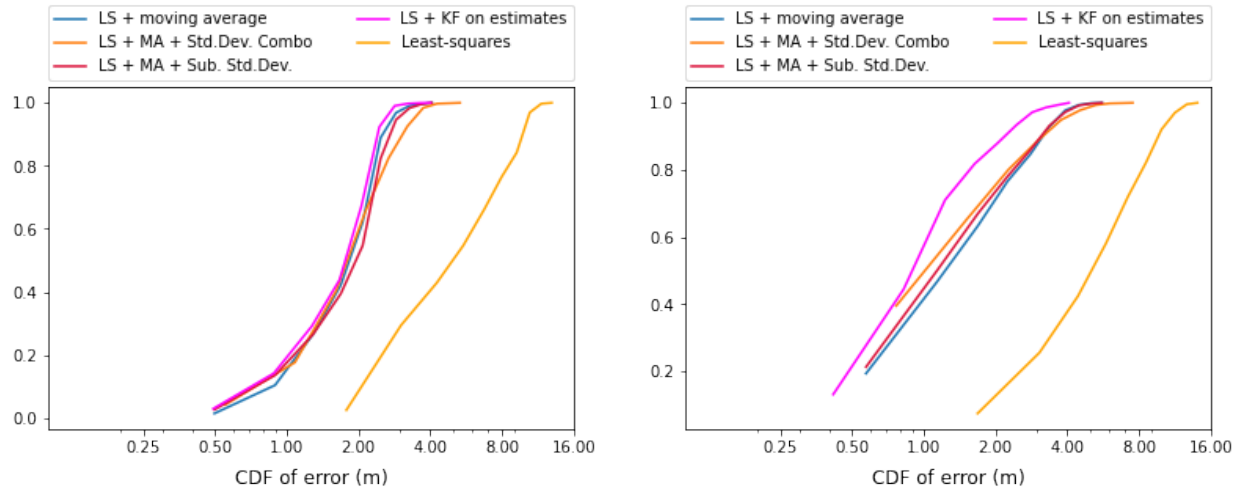
Figure 5.6: Comparison of reported distance, reported standard deviation and true distance per Access Point.



Google Pixel 4 Xiaomi Mi 9T

Figure 5.7: Comparison between True Ranges and Estimated Ranges in Google Pixel 4 (Android 12) and Xiaomi Mi 9T (Android 11)

Every AP consistently reporting distances with 2.5 meter errors will have a visible negative impact in estimate precision. In fact, accounting for this bias by adding 2.5 meters to every reported distance resulted in Least Squares estimates that tended to improve. Figure 5.8 showcases the cumulative error distribution for Least Squares methods in Xiaomi Mi 9T before and after this adjustment.



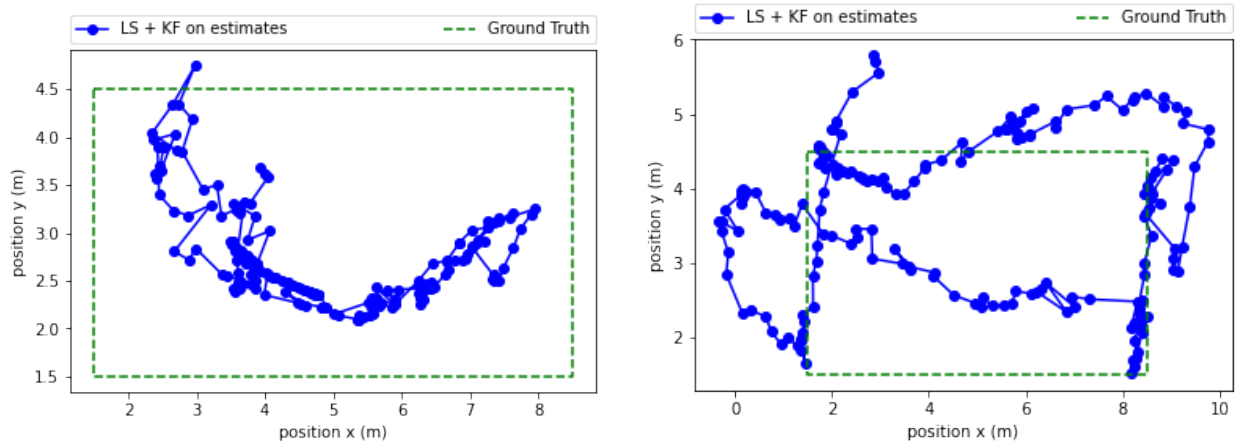
Before bias correction After bias correction

Figure 5.8: Cumulative Distribution Functions for Least Squares methods in Xiaomi Mi 9T

After bias correction, the methods show similar behaviour to when they were used in the Google Pixel 4. The only exception being the optimization of subtracting reported standard deviation, which went from resulting in noticeable improvements to barely having an impact.

Figure 5.9 shows an example of bias adjustments applied to a test run. This allows us to observe how the correction allowed us to recover the rough shape of the route, which had been lost due

to bias. Furthermore, this also illustrates one of our motives for claiming that results in the Xiaomi Mi 9T were disproportionately worse, as these accentuated distortions of the estimated route tended to happen in most test runs.



Before bias correction

After bias correction

Figure 5.9: Example of the impact of bias correction on a test run in Route 1

As for Dead Reckoning, step detection on the Xiaomi Mi 9T seems to be coherent with Pixel 4 results. For both test routes, the difference between average step count for both smartphones was within 4 steps.

Adversely, we observed that rotation information in the Xiaomi Mi 9T wasn't working properly with our heading estimation technique. This problem could indicate fundamental differences in hardware, firmware or sensor reading information between the two phones which we were unable to characterize.

Chapter 6

Conclusion

In this last chapter, we discuss our thoughts about the work done throughout this dissertation and provide insights regarding possible future work.

6.1 Discussion

In this dissertation we researched, developed and improved upon positioning methods for localization-enabling technologies present in Android smartphones. Through Wi-Fi RTT, we obtained absolute positioning estimates using the Least Squares multilateration of reported distances between the device and access points. Concurrently, through Android's sensor suite we obtained inertial data which was used to provide relative positioning estimates using SmartPDR.

Improving Wi-Fi RTT's localization approach composed a sizeable portion of our work. First, we observed that positioning using the Least Squares algorithm was fairly unstable due to the noise that stemmed from Wi-Fi RTT's high throughput. Then, we studied data smoothing techniques that acted on different components of this positioning process. All researched smoothing techniques were found to improve positioning accuracy. In particular, a two-dimensional Kalman Filter applied to Least Squares estimates yielded the best results. Finally, we tested our hypothesis of using ranging results' standard deviation data to mitigate the impact of unreliable distance reports by access points. Using this metric as a selection factor to determine which subset of APs would be used in Least Squares multilateration tended to show a modest improvement in precision. However, subtracting deviations from reported distances in order to partially correct the surplus in inaccurate estimates ultimately displayed a fairly more expressive accuracy gain.

Subsequently, our proposed sensor fusion algorithm sought to combine both of our chosen base localization methods into a single improved positioning process. For this purpose, we used a Kalman Filter with state updates performed using both estimates from a smoothed Least Squares multilateration and SmartPDR, proportionately to an attributed weight factor between

the two methods. Our main objective with this combination was to test if the methods could complement each other's weaknesses. In this regard, we observed that SmartPDR tended to be beneficial towards correcting positions in phases when Wi-Fi RTT reports were inaccurate due to this method's strength of being unaffected by network challenges. Likewise, introducing a component of absolute positioning estimation into SmartPDR by having its relative positioning method calculate in function of previous fusion results was often favorable towards correcting PDR drift. Overall, this algorithm tended to show improved precision when compared to all other tested approaches.

We must however mention that these results are in part contingent on the limitations of our testing setup. Experiments with a different phone than the one predominantly used for testing proved to be troublesome, partially due to discrepancies we found in firmware support but also potentially because of hardware differences. Through this challenge we learned that, in practice, it's not feasible to assume that commonly proposed localization methods are universally transversal across devices without further consideration on how to adapt them according to hardware and software specifications.

6.2 Future Work

We are considering a few key directions for future work, each one necessarily tied to one of the components of the work that was already done. As such, this section is subdivided into brief discussions about our ideas for each one of these modules.

6.2.1 Algorithm Design

We want to keep exploring the capabilities of the Kalman Filter for both data smoothing and fusion. For smoothing, we want to explore a different approach to smoothing Wi-Fi RTT's reported distance data since, during testing, the one-dimensional Kalman Filter approach we used to smooth these sets of distances tended to be considerably less effective to smoothing over the set of positions resulting from Least Squares multilateration. This could start with the conceptualization of another way to define the state transition in the filter's prediction stage, as right now we're using a rather simple linear extrapolation technique. For fusion, our sensor fusion algorithm only supports static weights. As such, we are considering ways of designing an adaptive weight method. Potential options to accomplish this could include using the Kalman Filter's calculated uncertainty values.

6.2.2 Testing and Evaluation

We are considering several testing setups that are likely to not only substantially increase the expressiveness of our results but also allow us to better visualize the strengths and weaknesses of

our algorithms. Longer routes with lengthier completion times would better emphasize the effect the PDR drift has over time. Meanwhile, although the impact of NLOS was already visible in our experiments, setups with less AP density would better illustrate the reality of most current infrastructures. This would also create more situations where Wi-Fi RTT positioning estimates are unavailable, which would be valuable towards testing our sension fusion algorithm. To this end, we are also considering introducing additional random periods where Wi-Fi RTT information is not accessible into the data collected from our existing routes.

6.2.3 Implementation

Future directions for software development could lead us to explore more sensor alternatives. For example, Ultra-wideband is a promising alternative for ranging data. For example, our multilateration methods are directly applicable to UWB ranges, so it would be a matter of implementing a way to collect this data. Conversely, implementing altitude estimation and expanding our methods to provide three-dimensional positioning would be valuable, but should be more complex. This is mainly due to the fact that SmartPDR can only provide two-dimensional positioning, which consequently enforces the same limitation upon our sensor fusion algorithm.

Finally, we are considering converting our methods into a real-time positioning system. Intuitively, this is a logical progression of our work which could be done by porting the algorithms to Android, interfacing with the code we already have for data collection.

Bibliography

[1] Android Motion Sensors API documentation. https://developer.android.com/guide/topics/sensors/sensors_motion. Accessed: 2022-08-15.

[2] Android Sensors Overview. https://developer.android.com/guide/topics/sensors/sensors_overview. Accessed: 2022-08-15.

[3] Google Store - Google Wifi Home router. https://store.google.com/us/product/google_wifi_2nd_gen?hl=en-US. Accessed: 2022-12-05.

[4] Jupyter Notebook. https://jupyter.org/. Accessed: 2022-09-29.

[5] Matplotlib: Visualization with Python. https://matplotlib.org/. Accessed: 2022-09-29.

[6] Numpy. https://numpy.org/. Accessed: 2022-09-29.

[7] pandas - Python Data Analysis Library. https://pandas.pydata.org/. Accessed: 2022-09-29.

[8] PyQuaternion documentation. http://kieranwynn.github.io/pyquaternion/. Accessed: 2022-09-29.

[9] ScyPi Optimization and Root finding: least_squares documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html. Accessed: 2022-09-29.

[10] Wi-Fi location: ranging with RTT. https://developer.android.com/guide/topics/connectivity/wifi-rtt. Accessed: 2022-08-15.

[11] Wi-Fi Manager API documentation. hhttps://developer.android.com/reference/android/net/wifi/WifiManager. Accessed: 2022-08-15.

[12] Wi-Fi RTT RangingRequest.Builder documentation. https://developer.android.com/reference/android/net/wifi/rtt/RangingRequest. Accessed: 2022-09-01.

[13] Wi-Fi RTT (Round-Trip-Time) API documentation. https://developer.android.com/reference/android/net/wifi/rtt/package-summary. Accessed: 2022-08-15.

[14] IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pages 1–3534, 2016.

[15] Michal Aftanas, Rovňáková Jana, Miloš Drutarovský, and Dusan Kocur. Efficient Method of TOA Estimation for Through Wall Imaging by UWB Radar. volume 2, pages 101 – 104, 10 2008.

[16] Jihun Kim Alwin Polouse and Dong Seog Han. A sensor fusion framework for indoor localization using smartphone sensors and Wi-Fi RSSI measurements. *Applied Sciences, 9(20):379*, 2019.

[17] Anahid Basiri, Elena Simona Lohan, Pedro Figueiredo e Silva, Pekka Peltola, Chris Hill, and Terry Moore. Overview of positioning technologies from fitness-to-purpose point of view. In *International Conference on Localization and GNSS 2014 (ICL-GNSS 2014)*, pages 1–7, June 2014.

[18] Anahid Basiri, Elena Simona Lohan, Terry Moore, Adam Winstanley, Pekka Peltola, Chris Hill, Pouria Amirian, and Pedro Figueiredo e Silva. Indoor location based services challenges, requirements and usability of current solutions. *Computer Science Review*, 24:1–12, 2017.

[19] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej. Using RSSI value for distance estimation in wireless sensor networks based on ZigBee. In *2008 15th International Conference on Systems, Signals and Image Processing*, pages 303–306, 2008.

[20] Martin Brossard, Axel Barrau, and Silvère Bonnabel. AI-IMU Dead-Reckoning. *IEEE Transactions on Intelligent Vehicles*, 5(4):585–595, 2020.

[21] Luis Enrique Díez, Alfonso Bahillo, Jon Otegui, and Timothy Otim. Step Length Estimation Methods Based on Inertial Sensors: A Review. *IEEE Sensors Journal*, 18(17):6908–6926, 2018.

[22] J.S. Esteves, A. Carvalho, and C. Couto. Generalized geometric triangulation algorithm for mobile robot absolute self-localization. In *2003 IEEE International Symposium on Industrial Electronics (Cat. No.03TH8692)*, volume 1, pages 346–351 vol. 1, 2003.

[23] Hai fa Dai et al. An INS/GNSS integrated navigation in GNSS denied environment using recurrent neural network. *Defence Technology*, 16(2):334–340, 2020.

[24] Pooyan Shams Farahsari, Amirhossein Farahzadi, Javad Rezazadeh, and Alireza Bagheri. A Survey on Indoor Positioning Systems for IoT-Based Applications. *IEEE Internet of Things Journal*, 9(10):7680–7699, 2022.

[25] Jamil Fayyad, Mohammad A Jaradat, Dominique Gruyer, and Homayoun Najjaran. Deep learning sensor fusion for autonomous vehicle perception and localization: A review. *Sensors*, 20(15):4220, 2020.

[26] Ruud Henken. Indoor-localization using a mobile phone. *Masters Thesis, University of Groningen, The Netherlands*, pages 10–17, May 2014.

[27] Berthold K. P. Horn. Indoor positioning using time of flight with respect to WiFi access points. http://people.csail.mit.edu/bkph/ftmrtt.shtml/. Accessed: 2022-08-25.

[28] M.I. Jais, P. Ehkan, R.B. Ahmad, I. Ismail, T. Sabapathy, and M. Jusoh. Review of angle of arrival (AOA) estimations through received signal strength indication (RSSI) for wireless sensors network (WSN). In *2015 International Conference on Computer, Communications, and Control Technology (I4CT)*, pages 354–359, 2015.

[29] Esrafil Jedari, Zheng Wu, Rashid Rashidzadeh, and Mehrdad Saif. Wi-Fi based indoor location positioning employing random forest classifier. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–5, 2015.

[30] A.R. Jimenez, F. Seco, C. Prieto, and J. Guevara. A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU. In *2009 IEEE International Symposium on Intelligent Signal Processing*, pages 37–42, 2009.

[31] Wonho Kang and Youngnam Han. SmartPDR: Smartphone-based pedestrian dead reckoning for indoor localization. *IEEE Sensors journal*, 15(5):2906–2916, 2014.

[32] Mazzullah Khan, Yang Dong Kai, and Haris Ubaid Gul. Indoor Wi-Fi positioning algorithm based on combination of Location Fingerprint and Unscented Kalman Filter. In *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 693–698, 2017.

[33] Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. Indoor positioning using GPS revisited. In *International conference on pervasive computing*, pages 38–56. Springer, 2010.

[34] Zhenyu Liu, Bin Dai, Xiang Wan, and Xueyi Li. Hybrid Wireless Fingerprint Indoor Localization Method Based on a Convolutional Neural Network. *Sensors*, 19(20), 2019.

[35] Stefania Monica and Federico Bergenti. Hybrid Indoor Localization Using WiFi and UWB Technologies. *Electronics*, 8(3), 2019.

[36] Mathias Pelka. Position Calculation with Least Squares based on Distance Measurements. *Technical Report*, 2015.

[37] Rong Peng and Mihail L Sichitiu. Angle of arrival localization for wireless sensor networks. In *2006 3rd annual IEEE communications society on sensor and ad hoc communications and networks*, volume 1, pages 374–382. Ieee, 2006.

[38] Vincent Pierlot and Marc Van Droogenbroeck. A new three object triangulation algorithm for mobile robot positioning. *IEEE Transactions on Robotics*, 30(3):566–577, 2014.

[39] Gerasimos G Rigatos. Extended Kalman and particle filtering for sensor fusion in motion control of mobile robots. *Mathematics and computers in simulation*, 81(3):590–607, 2010.

[40] Cung Lian Sang, Bastian Steinhagen, Jonas Dominik Homburg, Michael Adams, Marc Hesse, and Ulrich Rückert. Identification of NLOS and Multi-Path Conditions in UWB Localization Using Machine Learning Methods. *Applied Sciences*, 10(11), 2020.

[41] Nitesh B. Suryavanshi, K. Viswavardhan Reddy, and Vishnu R. Chandrika. Direction finding capability in bluetooth 5.1 standard. In Navin Kumar and R. Venkatesha Prasad, editors, *Ubiquitous Communications and Network Computing*, pages 53–65, Cham, 2019. Springer International Publishing.

[42] John Voight. *Quaternion Algebras*. Springer Nature, 2021.

[43] Yapeng Wang, Xu Yang, Yutian Zhao, Yue Liu, and Laurie Cuthbert. Bluetooth positioning using RSSI and triangulation methods. In *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*, pages 837–842, 2013.

[44] Eric W Weisstein. Rotation matrix. *https://mathworld. wolfram. com/*, 2003.

[45] Eric W Weisstein. Euler angles. *https://mathworld. wolfram. com/*, 2009.

[46] Greg Welch, Gary Bishop, et al. An introduction to the Kalman filter. 1995.

[47] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. A Survey of Indoor Localization Systems and Technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599, 2019.

[48] Reza Zandian. *Ultra-wideband based indoor localization of mobile nodes in ToA and TDoA configurations*. PhD thesis, Dissertation, Bielefeld, Universität Bielefeld, 2018, 2019.

[49] Lukasz Zwirello, Tom Schipper, Marlene Harter, and Thomas Zwick. UWB localization system for indoor applications: Concept, realization and analysis. *Journal of Electrical and Computer Engineering*, 2012, 2012.