Unterschrift Betreuer

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMARBEIT

Particle Filter für automatische Laufzeit-Kalibrierung des 3-Achsen-Magnetometers in Smartphones

ausgeführt am Institut für Angewandte Physik (IAP)
der Technischen Universität Wien

unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Martin Gröschl

und Mitbetreuung von
Dipl.-Ing. Dr.-Ing. Paolo Fogliaroni
Esri R&D Center Vienna

durch

Andreas Stefl, BSc

Wien, 11. Dezember 2020

Unterschrift Student

Signature of Advisor

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

DIPLOMA THESIS

Particle filter for online calibration of three axis magnetometers in smartphones

carried out at the Institute of Applied Physics (IAP)
at the TU Wien

with supervisor
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Martin Gröschl

and co-supervisor
Dipl.-Ing. Dr.-Ing. Paolo Fogliaroni
Esri R&D Center Vienna

by

Andreas Stefl, BSc

Vienna, December 11, 2020

Signature of Author

Acknowledgements

I wish to express my sincere gratitude to my supervisor Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Martin Gröschl, my employer Esri R&D Center Vienna and my manager and friend Thomas Taschauer for the cooperation and willingness to let me work on this thesis. Without them, this thesis would not have been possible.

I want to thank my co-supervisor Dipl.-Ing. Dr.-Ing. Paolo Fogliaroni for always being open to questions, proofreading my work and providing valuable feedback.

Further, I wish to acknowledge my family and friends for supporting me under all circumstances.

Last but not least, I want to thank my girlfriend Samira for always encouraging and supporting me throughout the whole thesis.

Kurzfassung

Heutige Smartphones kommen im Normalfall mit einem eingebauten Magnetometer, welches dazu benutzt werden kann, die horizontale Orientierung, wie mit einem Kompass, zu bestimmen. Das Smartphone besteht jedoch selbst aus magnetischen Materialien, die die Messungen des Sensors systematisch beeinflussen. Eine Kalibrierung ist notwendig, um die Abweichung zu bestimmen und von der Messung abzuziehen. Diese Kalibrierung wird üblicherweise durch die Kombination mehrerer Sensoren, wie Beschleunigungssensor und Gyroskop, erreicht und vom Betriebssystem oder Hersteller durchgeführt. Außerdem ist sie dafür optimiert, wenig Prozessorleistung in Anspruch zu nehmen und wird durch schnelle Rotationen des Smartphones in verschiedene Richtungen, welche durch den Benutzer durchzuführen sind, erreicht. Da es sehr viele verschiedene Geräte auf dem Markt gibt, weichen die Implementierungen voneinander ab und es ist nicht klar wann die Kalibrierung fertiggestellt ist.

In dieser Arbeit stellen wir einen Particle Filter für die kontinuierliche Kalibrierung des Magnetometers vor, der keine besonderen Handbewegungen des Benutzers voraussetzt. Da unser Filter mehrere Möglichkeiten der Kalibrierung gleichzeitig in Betracht zieht, erwarten wir eine sinnvolle Abschätzung des Fortschritts.

Wir werden zeigen, dass unsere Kalibrierung des Magnetometers in Szenarien wie Navigation, Lokalisierung und Wegfindung für Fußgänger jener des Betriebssystems weitgehend überlegen ist. Gleichzeitig wird eine große Bandbreite an Geräten unterstützt.

Abstract

Nowadays smartphones usually come with a built-in magnetometer that can be used to estimate the horizontal orientation of the phone similar to a compass. Since phones contain magnetic parts like speakers they draw their own magnetic field which will bias the measurements of the magnetometer. A calibration is necessary to estimate the bias and to subtract it from the measurement. Such a calibration could require sensor fusion with the accelerometer and gyroscope and is implemented by the Operating System (OS) or manufacturer. These implementations are optimized for low computational effort and require the user to perform rotations of the phone in multiple directions. Since there are a lot of different phones on the market the implementations will behave differently and it will not be clear when the calibration is finished.

In this thesis, we propose a particle filter to realize a different type of magnetometer calibration which is performed continuously and without forcing the user to perform special gestures. This comes at the cost of being computationally more expensive. However, in contrast to the OS calibration, our technique will be able to quantitatively estimate the progress of the calibration.

We will show that our calibration is superior compared to those of the OS in pedestrian navigation, localization and wayfinding scenarios most of the time. Additionally, a wide range of devices is supported.

Contents

1 Introduction

1.1 Motivation

Nowadays smartphones come with various sensors that can be used to measure environmental properties and to estimate the orientation and motion of the device and the user. Like all measurement devices, these sensors require calibration. One of the most common sensors in the smartphone is the magnetometer, which is usually used to estimate the horizontal orientation of the phone, similar to a compass. Accurate orientation, and therefore magnetometer calibration, is crucial in pedestrian navigation, localization and wayfinding scenarios (e.g. Indoor Positioning System (IPS), see Section 1.4 for more details) since users usually rely on the orientation shown on the map displayed on their smartphone.

In principle, the magnetometer calibration could be built into the sensor by the manufacturer statically (as parameters) or dynamically (as a program). This is not feasible in some situations. The manufacturer might not know in which environment the senors will be used and therefore static calibration is not viable. Online calibration might be an overhead for the user or is simply too computational expensive for the small sensor hardware. In these cases the user of the sensor has to deal with calibration by himself.

In case of smartphones, the calibration of the magnetometer is implemented by the OS or manufacturer. These implementations are usually optimized for low computational effort and will not update the calibration continuously. Moreover, they require the user to perform unnatural movements of the phone which is difficult to communicate. Since there are a lot of different smartphone models on the market, the implementations will behave differently and it will not be clear when the calibration is completed. As a consequence, the calibration and therefore the measurements of the magnetometer depend highly on the OS and manufacturer. Treating these measurements equally across different devices is an additional and unforeseeable source of error since the OS does not provide any quantitative estimates about the accuracy of the calibration.

The calibration of the magnetometer is usually decomposed into different effects: hard iron, soft iron, temperature, and misalignment.

The hard iron effect is usually the dominant measurement error.[1] In case of magnetometers in smartphones, this effect is caused by magnetic materials inside the device, like permanent magnets of the speaker, which retain their magnetism even after removal of external magnetic fields. Since smartphones contain various components made of various materials and no model is like the other, the burden of hard iron calibration is passed down to the OS. Additionally, the hard iron effect will change over time because of magnetization by sufficiently strong external fields which gives this effect a dynamical component.

As discovered during working on this thesis, measurements without soft iron and temperature compensation are not available on Android and iOS.[2][3] Soft iron and temperature calibration are performed by the manufacturer of the sensor or smartphone in this case.

1.2 Our solution

In this thesis we introduce a new algorithm to compensate the hard iron effect of magnetometers in smartphones for pedestrian navigation, localization and wayfinding scenarios. The main idea was to overcome the limitations of the OS and manufacturer hard iron calibration. This included:

- No dependence on special gestures that have to be performed by the user in order to progress the calibration.
- As a consequence of the first item, the calibration should run continuously and in real-time.
- Using one algorithm across different devices and platforms.
- Quantification of the calibration uncertainty. This is crucial for error propagation and bias prevention.

Since existing methods of the OS and manufacturers optimize for low computational effort, we believed that we can overcome their limitations by trading off computational performance.

An existing algorithm was used to form an Inertial Measurement Unit (IMU) by the accelerometer and gyroscope. The IMU provides information about the orientation of the device. This is beneficial for the calibration of the magnetometer because it can be used to predict upcoming measurements for comparison.

This thesis should be the first step towards an unbiased compass in pedestrian navigation, localization and wayfinding scenarios. This is a critical feature, since

users usually rely on the orientation displayed on the map on their smartphone. Additionally, a meaningful error estimation of the calibrated magnetic field is beneficial for error propagation and unbiased estimates for heading positioning in an IPS.

In order to achieve an online hard iron calibration, a particle filter was designed and implemented. Particle filters can be updated recursively, similar to Kalman filters. These filters have a state with constant memory complexity and perform updates on that state based on new observations. This is a desirable feature for real-time algorithms, since time and memory complexity is foreseeable.

In general, a particle filter is computationally more expensive compared to a Kalman filter and depends on a random number generator, while Kalman filtering is a direct method. On the other hand, a particle filter is most likely easier to implement and easier to adopt which is desirable for prototyping. Furthermore, the particle filter does not depend on linearity or specific probability distributions and can embed constraints for the state variables. Moreover, modern smartphones are perfectly capable of running a particle filter.

In addition to the hard iron calibration, this thesis contributes a system that can be used to prototype, test and evaluate real-time algorithms that depend on smartphone sensors.

1.3 Outline of this thesis

The thesis is structured as follows. Chapter 2 gives a theoretical overview of the physics and mathematics this thesis is based on. In Chapter 3, we give a formal definition of the hard iron effect and take a look at conventional methods. An introduction to particle filtering is given in Chapter 4. The implementation is extensively described in Chapter 5. In Chapter 6, we evaluate the data collection, the orientation filter, and the performance of our particle filter. Finally, we present our conclusions in Chapter 7 as well as possible future work in Chapter 8.

1.4 Indoor positioning system

An IPS replaces Global Navigation Satellite Systems (GNSS) when coverage and precision become critical because of obstacles that block satellite signals. Examples are buildings (e.g. airports, train stations, headquarters, offices, garages) and underground locations. There is a large variety of methods to achieve indoor positioning. Examples are existing Wi-Fi installations or Bluetooth beacons to

distribute signals that can be received by a smartphone, real time image processing to detect optical landmarks, and mounted IMUs. IPS does not have an official standard and is dominated by several commercial systems on the market. The implementation of an IPS highly depends on spatial dimensions, building materials, expected accuracy, and budget and is often tailored to fit its environment. While Global Positioning System (GPS) achieves about 5 meters accuracy outdoors on average, IPS highly depends on the installation and can achieve sub-meter accuracy.

A common target device for IPS is the smartphone. They usually come with various sensors and a large amount of users. The IPS can be used to help the users to navigate through buildings or track their locations for analytical purposes and placement strategies. Asset tracking is also a common use case for IPS.

IPS comes with a wide range of technology: anchor nodes with fixed positions like Wi-Fi access points, Bluetooth beacons, sound, optical signals, optical structures, magnetic fields, and dead reckoning.

2 Theoretical Background

This chapter provides a theoretical overview of the physics and mathematics this thesis is based on. First, we will define magnetic fields and provide a mathematical description. Then we will shortly discuss how magnetic fields can be measured, what the Earth's magnetic field is and how it can be approximated. Afterwards we give an introduction to quaternions and how they can be used to represent three-dimensional rotations. At the end, we introduce Bayesian statistics, which is related to particle filters.

2.1 Magnetic fields

A magnetic field is a vector field that describes the physical influence of moving electric charges and magnetic materials. Moving electric charges in a magnetic field experience a force perpendicular to their velocity and to the magnetic flux density, which is described by the Lorentz force. Additionally, the effects of the magnetic fields can be seen with permanent magnets which do not rely on macroscopic electrical currents. Permanent magnets attract or repel each other depending on their orientation and can pull on magnetic materials such as iron. The origin of macroscopic stationary magnetic fields are electrical currents and clusters of ordered magnetic moments of atoms. Magnetic moments of atoms are caused by orbital angular momentum of subatomic particles. Those are quantum mechanical effects and cannot be described by classical physics.

A magnetic field is only a part of the more general electromagnetic field. Since special relativity, we know that electric and magnetic fields can be transformed into each other by choosing different frames of reference.

In classical physics, electromagnetism has its foundation in the four microscopic Maxwell's equations and the Lorentz force. In SI units the equations are the following.[4]

$$\begin{aligned} \nabla \cdot \boldsymbol{E} &= \frac{\rho}{\varepsilon_0} & \nabla \cdot \boldsymbol{B} &= 0 \\ \nabla \times \boldsymbol{E} &= -\frac{\partial \boldsymbol{B}}{\partial t} & \nabla \times \boldsymbol{B} &= \mu_0 \boldsymbol{J} + \mu_0 \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \end{aligned} \tag{2.1}$$

$$\boldsymbol{F} = q\,(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \tag{2.2}$$

The microscopic Maxwell Equations 2.1 are composed of the electric field strength vector $\boldsymbol{E}$, the magnetic flux density vector $\boldsymbol{B}$, the total electric charge density ρ, the total electrical current density vector $\boldsymbol{J}$, the permittivity of free space ε_0, and the permeability of free space μ_0.

The Lorentz force $\boldsymbol{F}$ in Equation 2.2 is the force acting on the charge q, that is moving with a velocity $\boldsymbol{v}$ through the electric field $\boldsymbol{E}$ and magnetic field $\boldsymbol{B}$.

Maxwell's equations are a set of coupled linear partial differential equations. Therefore the sum of two solutions yields another solution and the fields can be decomposed into the components of their origins.

There are two major variants of Maxwell's equations. The microscopic Maxwell Equations 2.1 have universal applicability in classical physics but are unpractical for analytical calculations and extensive numerical simulations. Solids contain approximately 10^{23} atoms per cm^3 which interact with each other, the bound electrons, and free electrons in case of a metal. The macroscopic Maxwell Equations 2.3 define two additional auxiliary fields (see Equations 2.4) which describe the macroscopic response of the material to electromagnetic fields by neglecting microscopic structures and phenomena like atoms and spins. Their use requires material specific parameters which have to be measured experimentally or calculated with great numerical effort.[4]

$$\begin{aligned} \nabla \cdot \boldsymbol{D} &= \rho_\mathrm{f} & \nabla \cdot \boldsymbol{B} &= 0 \\ \nabla \times \boldsymbol{E} &= -\frac{\partial \boldsymbol{B}}{\partial t} & \nabla \times \boldsymbol{H} &= \boldsymbol{J}_\mathrm{f} + \frac{\partial \boldsymbol{D}}{\partial t} \end{aligned} \tag{2.3}$$

$$\begin{aligned} \boldsymbol{D} &= \varepsilon_0 \boldsymbol{E} + \boldsymbol{P} \\ \boldsymbol{H} &= \frac{1}{\mu_0} \boldsymbol{B} - \boldsymbol{M} \end{aligned} \tag{2.4}$$

The macroscopic Maxwell Equations 2.3 are composed of the electric displacement field vector $\boldsymbol{D}$, the magnetizing field vector $\boldsymbol{H}$, the free electric charge density ρ_f, and the free electrical current density vector $\boldsymbol{J}_\mathrm{f}$.

The auxiliary fields are defined in the Equations 2.4. Where $\boldsymbol{P}$ is the polarization field and $\boldsymbol{M}$ is the magnetization field. $\boldsymbol{P}$ and $\boldsymbol{M}$ are material specific and describe the response to external $\boldsymbol{E}$ and $\boldsymbol{H}$ fields.

2.2 Magnetometer

A magnetometer is an instrument that measures the local strength, direction or relative change of the magnetic field. There are many different types of magnetometers which rely on different physical phenomena of magnetic fields. Examples for these effects are induction currents, linear force or torque experienced by a magnetic dipole in a magnetic field, and the Lorentz force.

In recent years, magnetometers have been miniaturized to the extent that they can be built into integrated circuits at very low cost. In 2009, the price of three-axis magnetometers dipped below one US dollar per device and dropped rapidly.[5] As Microelectromechanical Systems (MEMS), magnetometers are increasingly used as miniaturized compasses in smartphones and other devices. Compensation for temperature, hard iron, and soft iron effects are necessary.[6]

2.2.1 Hall-effect sensor

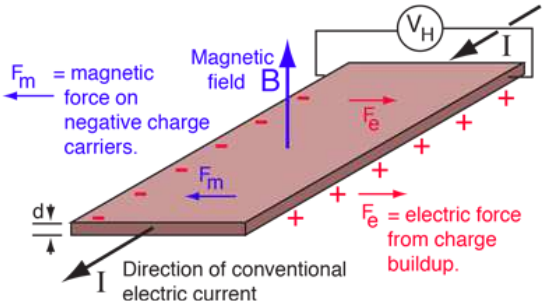


Figure 2.1: Schematic of a Hall probe measuring the magnetic field. Image gratefully taken from HyperPhysics.[7]

The Hall-effect can be used to measure the magnitude of magnetic fields in one direction and is named after Edwin Hall who discovered it in 1879. As MEMS, Hall-effect sensors are a popular alternative to magnetoresistive sensors.[8][9]

The effect is illustrated in Figure 2.1. Magnetic fields exert a transverse force on moving charge carriers. This includes electrical currents in conductors. There, the magnetic field is pushing the charges to one side of the material which is

most evident in thin flat conductors. The imbalance of the charges results in a measurable voltage between two sides of the conductor. Its output voltage V_H is directly proportional to the magnetic field strength passing through the sensor, and is equal to:[9]

$$V_H = \frac{IB}{ned} \tag{2.5}$$

Where I is the current flowing through the probe, B is the magnetic field perpendicular to the probe, n is the charge carrier density, e the elementary charge, and d is the probe thickness.

2.3 Earth's magnetic field

The Earth is sourcing a magnetic field from its interior which is reaching out into space, where it interacts with charged particles emitted by the Sun. The magnetic field is believed to be generated by electric currents in conductive convection streams of molten iron and nickel due to heat escaping the Earth's core. This process is complex and an active field of research. On the surface of the Earth, the magnitude of the magnetic flux density ranges from 25 to 65 μT. The field can be approximated by a magnetic dipole tilted at an angle of about 11 degrees with respect to Earth's rotational axis.[10][11]

At any location, the Earth's magnetic field can be represented by a three dimensional vector. This is illustrated in Figure 2.2. A typical procedure for measuring its direction is to use a compass to determine the direction of magnetic North. Its angle relative to true North is called declination or variation. Facing magnetic North, the angle the magnetic field encloses with the horizontal plane is called inclination or magnetic dip. The intensity of the field is proportional to the force it exerts on a magnet. A common representation is X (North), Y (East) and Z (Down) coordinates.[11][12]

2.3.1 Dipole approximation

In close proximity to the surface of the Earth, its magnetic field can be approximated by the field of a magnetic dipole placed at the center of the Earth and oriented with an angle of about 11 degrees to the rotational axis of the Earth. In this description the Earth can be viewed as a strong bar magnet with its south pole pointing towards the geomagnetic North Pole. The dipole field accounts for approximately 80–90% of the field in most locations.[11]

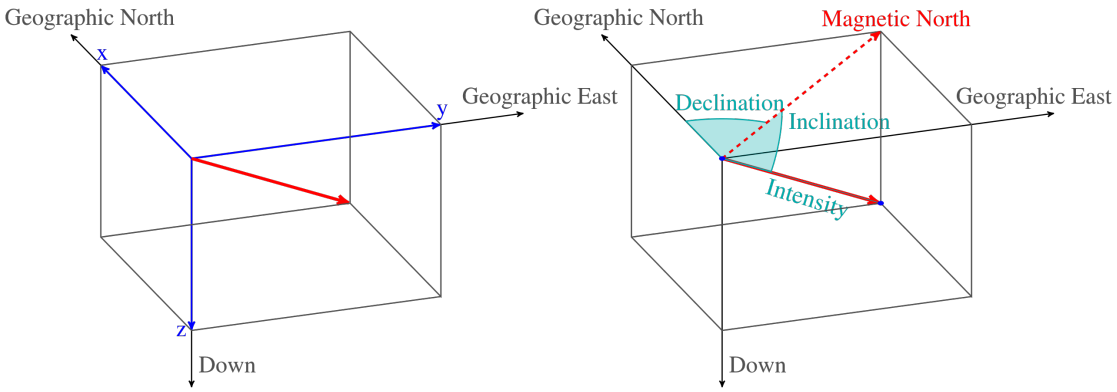


Figure 2.2: Common coordinate systems used for representing the Earth's magnetic field. Image gracefully taken from Wikimedia.[13]

In spherical coordinates, the dipole magnetic field of the Earth can be described as follows.[14]

$$\begin{aligned} B_r &= -2B_0(\frac{R_E}{r})^3 \cos\theta \\ B_\theta &= -B_0(\frac{R_E}{r})^3 \sin\theta \end{aligned} \tag{2.6}$$

Where B_0 (typically $B_0 = 3.12 * 10^{-5}$ T) is the mean value of the magnetic flux density at the magnetic equator on the Earth's surface. R_E is the mean radius of the Earth (approximately 6370 km), r is the radial distance from the center of the Earth, and θ is the azimuth angle measured from the north geomagnetic pole.

This approximation is very practical for our purposes since it is easy to calculate the magnetic flux density for a given WGS84 position.

2.4 Quaternions

Quaternions are a number system that extends the complex numbers with two additional imaginary axis. In 1843, they were described by the Irish mathematician William Rowan Hamilton and applied to mechanics in three-dimensional space.[15]

Quaternions are generally represented in the form

$$\boldsymbol{q} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k} \tag{2.7}$$

where a, b, c, and d are real numbers, and $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are the fundamental quaternion units, similar to the imaginary unit $\boldsymbol{i}$.

The imaginary units are connected through multiplication by

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1. \tag{2.8}$$

Quaternions are used in theoretical mathematics and applied mathematics. Applications are calculations involving three-dimensional rotations such as in computer graphics, computer vision, and crystallographic texture analysis. In practical applications, they can be used alongside other methods, such as Euler angles and rotation matrices, or as an alternative to them.

The unit quaternions can be thought of as a choice of a group structure on the 3-sphere S^3 that gives the group Spin(3), which is isomorphic to SU(2) and also to the universal cover of SO(3).

The conjugate of q is given by $q^* = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k}$.

The norm by $\|q\| = \sqrt{qq^*} = \sqrt{q^*q} = \sqrt{a^2 + b^2 + c^2 + d^2}$.

The inverse by $q^{-1} = \frac{q^*}{\|q\|^2}$.

2.4.1 Quaternions as rotations

To express a rotation of an angle θ counterclockwise around a unit vector $\boldsymbol{u} = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}$ we compose a unit quaternion with the extended Euler's formula

$$\boldsymbol{q} = e^{\frac{\theta}{2}\boldsymbol{u}} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\boldsymbol{u}\ . \tag{2.9}$$

In order to apply this rotation to a point $\boldsymbol{p} = p_x\boldsymbol{i} + p_y\boldsymbol{j} + p_z\boldsymbol{k}$ one has to compute

$$\boldsymbol{p}' = \boldsymbol{qpq}^{-1}\ . \tag{2.10}$$

Such a rotation is illustrated in Figure 2.3.

Rotations can be composed by multiplying quaternions and reversed with the inverse quaternion $\boldsymbol{q}^{-1}$.

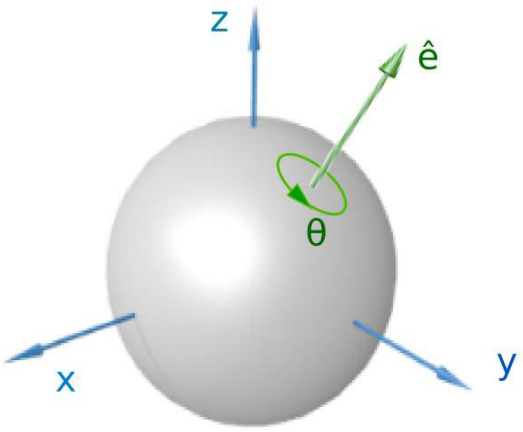


Figure 2.3: Illustration of a rotation represented by an Euler axis and angle. Image gracefully taken from Wikimedia.[16]

2.5 Bayesian statistics

Bayesian statistics is a theory based on the Bayesian interpretation of probability where it expresses a degree of belief in an event. The degree of belief can be based on prior knowledge about the event, like results of previous experiments, or on personal beliefs about the event. This differs from the frequentist interpretation which views probability as the limit of the relative frequency of an event after many trials.[17]

Bayes' theorem is used in Bayesian methods to update probabilities after obtaining new data. Given two events A and B, the conditional probability $P(A|B)$ – the probability for A given that B is true – is expressed as follows:[17]

$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)} \text{ with } P(B) \neq 0 \tag{2.11}$$

Bayes' theorem is a fundamental result of probability theory, but it has a specific interpretation in Bayesian statistics. In Equation 2.11, A represents a proposition (prior information like "this coin lands on heads fifty percent of the time") and B represents new data we want to take into account (e.g. the result of a series of coin flips). $P(A)$ is the prior probability of A, which expresses our beliefs about A before evidence is taken into account. $P(B|A)$ is the likelihood function, which can be interpreted as the probability of the evidence B given that A is true. The likelihood represents the model of the stochastic process and quantifies the extent to which evidence B supports the proposition A. $P(A|B)$ is the posterior probability –

the probability of A after taking the evidence B into account. Essentially, Bayes' theorem updates our prior beliefs $P(A)$ after considering new evidence B.[17]

In contrast to the frequentist interpretation, which views probability as the limit of the relative frequency of an event after many trials, Bayesian statistics does not rely on repeatable experiments. There is no true universal value of probability that we want to approach with many trials. Probability is a degree of belief based on data, a model and prior beliefs.

3 Hard Iron Effect

In this chapter, we will formalize measurement biases of magnetometers. First we give an overview, followed by a physical explanation. Then, we provide a model that includes major biases and introduce a desirable decomposition of the magnetic field. At the end, we summarize different approaches for magnetometer calibration which were developed in the past.

3.1 Overview

Measurement biases of magnetometers in smartphones are caused by magnetic materials inside the phone which interfere with the external magnetic field. The measurement bias is usually decomposed into two different effects: the hard iron and the soft iron effect. Figure 3.1 is an example of how hard and soft iron effects influence the measurement of the magnetometer after a full rotation of the sensor along an arbitrary axis in a homogeneous magnetic field.

The hard iron effect of magnetometers in smartphones is due to magnetic materials inside the phone which retain their magnetism even after the removal of the external magnetic fields. A permanent magnet of the speaker inside the phone is an example for that. Since these components are firmly attached to the magnetometer, the hard iron effect will cause an offset between the external and the measured magnetic field. The magnetization of the materials in the phone can still change in strength and direction in presents of external fields which adds dynamics to the hard iron effect.

The soft iron effect is due to inhomogeneous magnetization of materials in the smartphone in presence of external magnetic fields. This effect distorts the measurement of the magnetometer depending upon which direction the field acts relative to the sensor. In contrast to the hard iron effect, the soft iron effect does not bias measurements in absence of external magnetic fields.

Usually, the hard iron effect has a much larger contribution to the total uncorrected error than soft iron effect.[1]

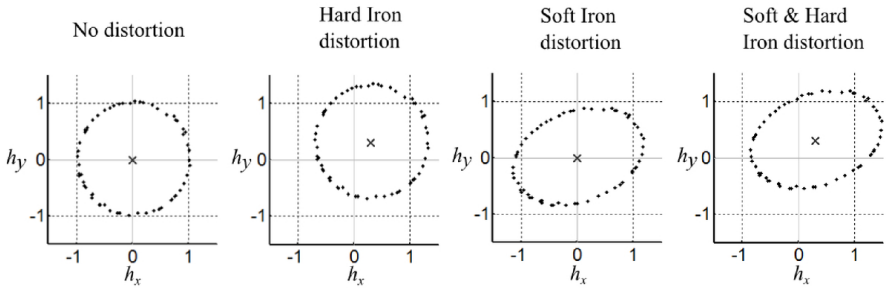


Figure 3.1: Illustration of the hard and soft iron effect. Image gracefully taken from Sawicki's paper.[18]

As discovered during working on this thesis, iOS and Android only provide magnetometer measurements without hard iron calibration but with soft iron and temperature calibration. This suggests that the soft iron effect and temperature dependencies are calibrated by the sensor hardware and the hard iron effect is calibrated by software of the OS or manufacturer. In this case the calibration depends on different algorithms across different devices. Treating their estimates equally is an additional source of error in cross-device scenarios, such as IPS (see Sections 1.4 for more details). Moreover the operating system does not quantify the quality of the hard iron calibration and requires unnatural movements of the phone to perform the calibration.

3.2 Physical origin

The soft and hard iron effects can be explained by magnetic hysteresis. It occurs when a ferromagnet, such as iron, is exposed to an external magnetic field and the magnetic dipoles of the atoms align with it. Some of the alignments will remain even after removing the external field which is called *remanence*. The material has become magnetized and will remain so indefinitely. In order to demagnetize it one has to apply heat or a sufficiently strong external magnetic field in the opposite direction.[19]

Magnetic hysteresis can be visualized with a $\boldsymbol{H}$-$\boldsymbol{M}$ or $\boldsymbol{H}$-$\boldsymbol{B}$ diagram. An example is given in Figure 3.2. In general the relationship between the external magnetic field strength $\boldsymbol{H}$ and the magnetization $\boldsymbol{M}$ is not linear. The origin of the diagram ($\boldsymbol{H} = \boldsymbol{M} = \boldsymbol{0}$) denotes the demagnetized state of the material in absence of external fields. Increasing the external field $\boldsymbol{H}$ in one direction leads to an increasing magnetization $\boldsymbol{M}$ until saturation is reached. All dipoles point in the same direction. After removing the external field the material is still magnetized. This

is called *remanence* and is denoted by $\boldsymbol{B}_r$ in the diagram. Reversing the external field will weaken the magnetization until it vanishes at $\boldsymbol{H}_c$, called *coercivity*.

Note that this is a one dimensional simplification of a more complicated three-dimensional situation.

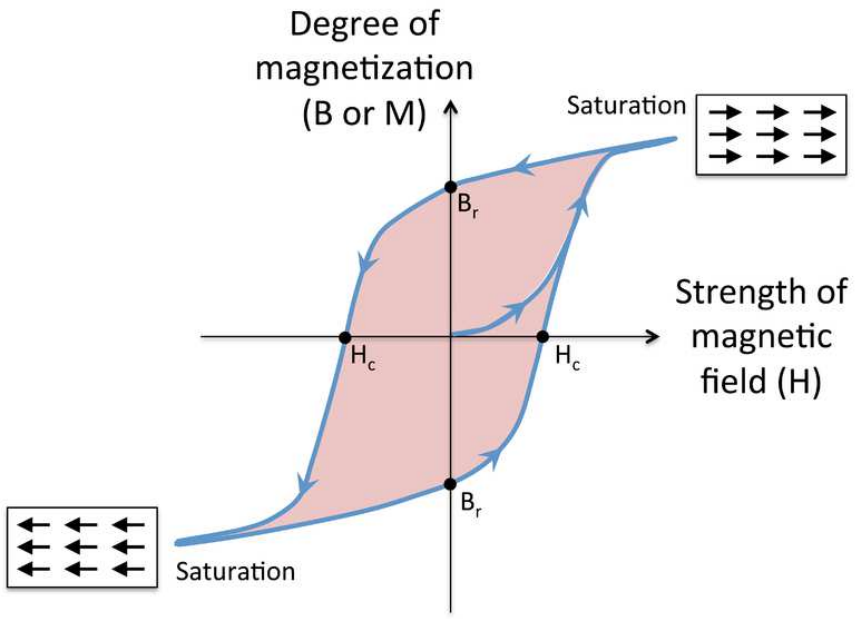


Figure 3.2: Illustration of the magnetic hysteresis. Image gracefully taken from Wikimedia.[20]

Ferromagnetic materials can be divided into magnetically soft and hard materials. Soft materials can be magnetized but do not tend to stay so because of low coercivity. Hard materials tend to stay magnetized because of high coercivity and can be used as permanent magnets.

The soft iron effect is caused by magnetically soft materials which magnetize in the presence of external fields and therefore amplify it. In general this effect is anisotropic because of the geometry of the material and its relative position to the magnetometer.

The hard iron effect is caused by magnetically hard materials which remain magnetized in absence of external fields. Since the magnetometer is firmly attached to the magnetically hard materials, this effect will cause an offset of the measurement, independent of the orientation of the sensor. Because of the magnetic hysteresis, the hard iron offset will change over time in presence of sufficiently strong and altering external magnetic fields.

3.3 Magnetometer sensor model

Equation 3.1 is a model for magnetometer measurements that includes soft and hard iron effects. The soft iron effect is parameterized with 6 values (s_x, s_y, s_z, s_{xy}, s_{xz}, s_{yz}) which scale and shear the magnetic field $\boldsymbol{B}$. The hard iron effect is parameterized with 3 values (h_x, h_y, h_z) which shift the external magnetic field vector $\boldsymbol{B}$. $\boldsymbol{B}_{\text{noise}}$ is the measurement noise of the magnetometer, which could be modeled as zero-centered Gaussian noise.[1]

Equation 3.2 is the inverse of 3.1 which can be used estimate the magnetic field $\boldsymbol{B}$ from a measurement $\boldsymbol{B}_{\text{measure}}$, given the soft and hard iron parameters.

$$\boldsymbol{B}_{\text{measure}} = \begin{bmatrix} 1 & s_{xy} & s_{xz} \\ 0 & 1 & s_{yz} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \left(\boldsymbol{B} + \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} \right) + \boldsymbol{B}_{\text{noise}} \tag{3.1}$$

$$\hat{\boldsymbol{B}} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & \frac{1}{s_z} \end{bmatrix} \times \begin{bmatrix} 1 & s^*_{xy} & s^*_{xz} \\ 0 & 1 & s^*_{yz} \\ 0 & 0 & 1 \end{bmatrix} \times \boldsymbol{B}_{\text{measure}} - \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} \tag{3.2}$$

In case of a misalignment between device and sensor axes an additional rotation is required with 3 parameters.

Temperature dependencies can be modeled with polynomials for each parameter.

3.4 Decomposition

Neglecting the soft iron effect, the measurement of the magnetometer can be expressed by the following decomposition.

$$\boldsymbol{B}_{\text{measure}}(t, \boldsymbol{x}) = \boldsymbol{B}_{\text{phone}}(t) + \boldsymbol{B}(t, \boldsymbol{x}) + \boldsymbol{B}_{\text{noise}} \tag{3.3}$$

$$\boldsymbol{B}(t, \boldsymbol{x}) = \boldsymbol{B}_{\text{earth}}(t, \boldsymbol{x}) + \boldsymbol{B}_{\text{env}}(t, \boldsymbol{x}) \tag{3.4}$$

Where $\boldsymbol{B}_{\text{measure}}(t, \boldsymbol{x})$ is the measurement of the magnetometer at a given point in time and space, $\boldsymbol{B}_{\text{phone}}(t)$ the magnetic field created by the smartphone, $\boldsymbol{B}(t, \boldsymbol{x})$ is the external magnetic field, $\boldsymbol{B}_{\text{earth}}(t, \boldsymbol{x})$ the magnetic field of the earth, $\boldsymbol{B}_{\text{env}}(t, \boldsymbol{x})$ is the magnetic field of the environment, and $\boldsymbol{B}_{\text{noise}}$ the noise produced by the magnetometer.

In pedestrian navigation, localization and wayfinding scenarios one can neglect the time dependency of $\boldsymbol{B}_{\text{earth}}$ and $\boldsymbol{B}_{\text{env}}$ because they vary on much longer timescales.

The decomposition given in Equation 3.4 is relevant for unbiased estimates of the horizontal orientation with $\boldsymbol{B}_{\text{earth}}$ and for localization techniques that rely on the magnetic field of the environment $\boldsymbol{B}_{\text{env}}$.

Let us imagine to rotate the phone in any direction to understand how the individual components in Equation 3.3 change. Assuming that $\boldsymbol{B}_{\text{noise}}$ is isotropic, it will not change. In the frame of reference of the phone, $\boldsymbol{B}$ will be rotated in the opposite direction. The magnetic field of the device $\boldsymbol{B}_{\text{phone}}$ is created by components of the interior which are firmly connected to the magnetometer. Therefore $\boldsymbol{B}_{\text{phone}}$ does not change in the frame of reference of the device.

Since $\boldsymbol{B}_{\text{earth}}$ and $\boldsymbol{B}_{\text{env}}$ in Equation 3.4 behave in the same way under rotation, we can guess already that the decomposition of $\boldsymbol{B}$ is a non trivial problem. $\boldsymbol{B}_{\text{earth}}$ only depends on the horizontal orientation of the device, assuming we know its horizontal and vertical components (those can be calculated with the dipole approximation of the Earth's magnetic field, see Equation 2.6 in Subsection 2.3.1). Without any model for $\boldsymbol{B}_{\text{env}}$ the decomposition is ambiguous for all directions. Such a model would have to make assumptions about the magnitude (for example $\|\boldsymbol{B}_{\text{env}}\| \ll \|\boldsymbol{B}_{\text{earth}}\|$) or direction (for example random directions) of the environmental magnetic field.

3.5 Related work

Common calibration techniques are:

- **Static calibration.** The device is rotated in multiple directions while measuring the magnetic field and possibly the orientation of the device. A numerical fit can be used to estimate the hard iron effect and these values will later be used in the application to correct the measurements.[21][22][23]
- **Calibration by significant rotations.** An algorithm detects when the phone is rotated significantly in multiple directions, collects batches of sensor data and uses the same approach as above or a filtering algorithm to estimate the hard iron effect.[24][25][26]
- **Online calibration.** A real-time algorithm is updated with consecutive observations iteratively. Specific assumptions might be introduced depending on the context of the application (e.g. automotive, aircraft, IPS).[27][28]

4 Particle Filter

This chapter is a short introduction to Sequential Monte Carlo (SMC) methods and specifically particle filters. First, we give an overview about particle filters and state-space models. Then we will derive the recursive filtering equations by following Bayes' theorem. At the end, we will introduce resampling and point out why it is a crucial feature of SMC methods.

4.1 Overview

The term *filtering* in this context means extracting information about a signal from partial and noisy observations in dynamical systems. In contrast to other estimation problems, filtering is about estimating the current state of a system given only observations up to this point in time.[29]

Particle filters were originally developed for object tracking and time series analysis for nonlinear, non-Gaussian state-space models.[30] The term particle filter was first used by Del Moral in 1996.[31]

Particle filters are SMC methods and a subset of Monte Carlo (MC) algorithms. They can be used to solve filtering problems arising in signal processing and Bayesian statistical inference. The goal is to compute the posterior distributions of states in a Markov process, given the prior distribution of states and some partial and noisy observations.[32]

The particle filter uses a set of particles to represent the current state of the system. There is no limitation to the state-space model, the initial state, and the noise distribution of observations. However, in practice the particle filter does not perform well in very high-dimensional systems.[32]

Each particle has a weight (or likelihood) that represents an approximate probability measure of that particle being sampled from the usually analytically unknown probability density function. The weights are updated when new evidence about the true state is available. A resampling step is necessary to avoid weight disparity leading to weight collapse. There are several adaptive resampling criteria

that can be used to reduce noise, including the variance of the weights and the relative entropy with respect to the uniform distribution. In the resampling step, the particles with negligible weights are replaced by new particles in the proximity of the particles with higher weights.[31][33]

4.2 Filtering with state-space models

According to Hans Künsch[34], in the last 50 years filtering has been mainly studied in the framework of state-space or hidden Markov models, assuming a Markovian time evolution of the signal and observations, which are instantaneous functions of the signal, and subject to white observation noise.

The objective of a filter is to estimate the hidden state variables X_k (also called signal) given the observation variables Y_k at iteration k. The observable variables Y_k are related to the hidden variables X_k by some conditional probability density $g(y_k|x_k)$ that is known. Similarly, the dynamics of the state process is also known $f(x_k|x_{k-1})$. A generic state and observation process is illustrated below.[34]

$$\begin{array}{ccccccccc} X_0 & \rightarrow & X_1 & \rightarrow & X_2 & \rightarrow & X_3 & \rightarrow & \cdots \quad \text{signal} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \cdots \\ Y_0 & & Y_1 & & Y_2 & & Y_3 & & \cdots \quad \text{observation} \end{array}$$

Given the sequential observations $Y_0, \cdots, Y_k$ at any time step k, the filtering problem is to estimate the signal X_k.

Bayesian estimates of X_k follow from the posterior density $p(x_k|y_0, \cdots, y_k)$. Particle filters provide an approximation of these conditional probabilities by weighted samples. In contrast to SMC methods, Markov Chain Monte Carlo (MCMC) methods would have to deal with the full posterior distribution $p(x_0, \cdots, x_k|y_0, \cdots, y_k)$ in each iteration.

4.3 Filtering recursion

In order to start the recursion, a prior distribution $p(x_0)$ of the signal X_0 is required. Selecting samples of the prior distribution is called initialization. One has to be careful choosing a prior distribution because it might cause bias. The prior could depend on the first observation Y_0 for example.

At time step $k = 1$ we observe Y_1 and want to update the filter. Following Bayes' theorem (see Equation 2.11 in Section 2.5) we derive the following:

$$p(x_0, x_1|y_1) \sim L(y_1|x_0, x_1)\ p(x_1) = g(y_1|x_1)\ f(x_1|x_0)\ p(x_0) \tag{4.1}$$

Where $L(y_1|x_0, x_1)$ is the likelihood function given by the observation function $g(y_k|x_k)$ and $p(x_1)$ is the prior for X_1 given by the signal prediction function $f(x_k|x_{k-1})$ and $p(x_0)$. Following this process recursively and applying normalization yields the following:

$$p(x_{1:k}|y_{1:k}) = \frac{g(y_k|x_k)\ f(x_k|x_{k-1})\ p(x_{1:k-1}|y_{1:k-1})}{p(y_k|y_{1:k-1})} \tag{4.2}$$

$$p(y_k|y_{1:k-1}) = \int_S g(y_k|x_k)\ f(x_k|x_{k-1})\ p(x_{1:k-1}|y_{1:k-1})\, dx_k \tag{4.3}$$

With the abbreviation $x_{1:k} = x_1, \cdots, x_k$, the normalization $p(y_k|y_{1:k-1})$ in Equation 4.3, and the state space volume S.

By marginalization of 4.2 (integration over $x_{1:k-1}$) we derive the following:

$$p(x_k|y_{1:k}) = \frac{g(y_k|x_k)\ p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})} \tag{4.4}$$

$$p(x_k|y_{1:k-1}) = \int_V f(x_k|x_{k-1})\ p(x_{k-1}|y_{1:k-1})\, dx_{k-1} \tag{4.5}$$

Equation 4.5 is usually called the prediction step and 4.4 the update step. However, according to Arnaud Doucet[32] most particle filtering methods rely on numerical approximations of 4.2 instead of 4.5 and 4.4.

Analytical solutions can be given for two important special cases. First, when the state space S is finite, the integrals above can be reduced to finite sums. Second, when the system is linear and $p(x_0)$, $g(y_k|x_k)$, and $f(x_k|x_{k-1})$ are Gaussian, the filtering problem can be solved by a Kalman filter exactly.[35]

4.4 Resampling

An algorithm that follows the initialization, prediction, and update steps sequentially by representing the distributions with N samples is called Sequential Importance Sampling (SIS). It can be shown that the variance of the weights is growing exponentially with the number of iterations. The important and defining feature

of particle filters is the resampling step, which removes particles with low weights and replaces particles with high weights by multiple offspring particles in close proximity. After a resampling step, all particles have the same weight and the distribution is only represented by the number of particles in the different states.[32]

Common algorithms for resampling are **systematic resampling** and **multinomial resampling**. Both require a summation of all weights which can be a bottleneck for parallelization. Other techniques were discussed by Murray, Lee, and Jacob in 2016.[32][36]

Adaptive resampling can be used to reduce noise and avoid unnecessary resampling steps in case of a sufficiently even weight distribution. A popular criteria is the effective number of particles.

$$\hat{N}_{eff} = \frac{1}{\sum_{i=1}^{N} (w_i)^2} \leq N_{thr} \tag{4.6}$$

With normalized weights $\sum_{i=1}^{N} w_i = 1$ and the threshold parameter N_{thr}.[33]

5 Implementation

This chapter walks through the chosen technologies, shows the architecture, frames the implementation effort, and explains the decisions taken while working on this thesis. First we will introduce the different technologies and how they play together. Second we will shortly define the frames of reference and the architecture of the implementation. Then the particle filter implementation will be explained in detail, followed by the filter routine and its parameters. Afterwards a few intermediate steps and solved issues that should arise while working on this thesis are discussed. Finally the platform implementation for Android and the effort made on visualization is described.

In order to design, implement and evaluate an online hard iron calibration, two different approaches were taken into consideration. Approach *One*: using existing tools to collect the required sensor data and to process and evaluate it offline. Approach *Two*: designing a system from scratch that collects, processes and visualizes sensor data in real time. Since the outcome of this thesis should be an online algorithm and it seemed to be more target-oriented, the prototyping was created with the paradigm of approach *Two*. Moreover, the orientation estimation (see Section 5.8) required qualitative evaluation which was preferable in the online scenario because of the immediate response in the visualization. Approach *One* was later used for the quantitative evaluation of the hard iron calibration.

5.1 Technology

The major criteria for the choice of technology was *platform independence* and *real-time processing*. Platform independence because we are targeting two different mobile platforms with Android and iOS and the need of an evaluation framework that runs on a desktop environment like Ubuntu. Since our calibration should be online, real-time processing is necessary and therefore a critical performance limit is attached to our problem: When one second of real-time has passed we have to process at least one second of input data.

Visualization was planned for prototyping and evaluation purposes. Similar to our calibration algorithm, the visualization was planned to be platform independent

with the benefit of displaying it on the smartphone and on the desktop with the same implementation. The first attempt was to use OpenGL from C++ but that introduced a lot of overhead to the implementation. Since all target platforms have an HTML5 compatible browser, the second and final attempt was to use JavaScript for that purpose. To communicate with C++ in real-time, WebSocket was used to exchange Protocol Buffers (see Subsection 5.1.4).

The target platform for the demo application was chosen to be Android. It was developed in Java, responsible for reading the sensors and providing a small user interface.

Table 5.1 gives a quick overview about the chosen technologies and the implementation effort by lines of code.

Language	**Total lines of code**
C++	3092
Java	366
JavaScript	450
Python	179
Protocol Buffers	55
CMake	203

Table 5.1: Total lines of code by language written for this thesis.

5.1.1 C++

The programming language that was used the most in this project is C++ for various reasons. C++ is platform independent and can reach major smartphone platforms like Android and iOS. Performance was critical for the implementation which is a major purpose of C++. Since C++11 there is a sufficient standard library for common data structures, random variables and common algorithms. Besides that, C++ is a modern programming language with multiple paradigms for example it is object-oriented and generic.

Along with C++ multiple tools, frameworks, and libraries were used which are listed and briefly described below.

- **boost** is a popular C++ library that comes in handy where the standard library is missing functionality. This thesis only depends on **boost asio** for networking and **boost beast** for HTTP and WebSocket.
- **Eigen** is a rich library for linear algebra and comes with implementations for matrices, vector, and quaternions.

- **gtest** is a unit test framework from Google and was used to write small self-contained (unit) test cases.
- **CMake** was used as dependency management and build system.

5.1.2 Android

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software. It was primarily designed for mobile devices with touchscreen such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google. It was unveiled in November 2007, with the first commercial Android device launched in September 2008.

Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013. As of May 2017, it has over two billion monthly active users, which is the largest installation base of any operating system.

Android is one of the target platforms for this thesis. Native applications can be developed with Android Studio in Java or Kotlin. Moreover, C++ can be integrated and called through Java Native Interface (JNI) for shared library access, enabling low-level functionality and high performance applications.

5.1.3 JavaScript

JavaScript was used to realize platform independent visualization that can be displayed on desktop and mobile devices. In contrast to the original approach via OpenGL it was more straight forward to use with **plotly**[1] and a **matplotlib**[2] background in Python. Moreover, modern browsers come with a WebSocket implementation which reduced the complexity of communication to C++ code.

Along with JavaScript multiple tools, frameworks, and libraries were used which are listed and briefly described below.

- **webpack** is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser.
- **plotly** is a rich plotting library for JavaScript and Python and was used for visualization purposes.

[1]High level plotting library for Javascript and Python. https://plotly.com
[2]High level plotting library for Python. https://matplotlib.org

- **THREE** is a library for linear algebra and comes with implementations for matrices, vector, and quaternions.

5.1.4 Protocol Buffers

Protocol Buffers[3] (short Protobuf) is Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data. It is useful for programs which communicate over a wire or for storing data. Protocol Buffers involve an interface description language which defines the structure of the data. Additionally it comes with a program that generates source code from that definition. The generated code deals with the encoding and decoding of byte streams which represent the structured data.

The data can be serialized in binary representation which takes up less space compared to a text representation like JSON. Therefore it needs less bandwidth while communicating over network.

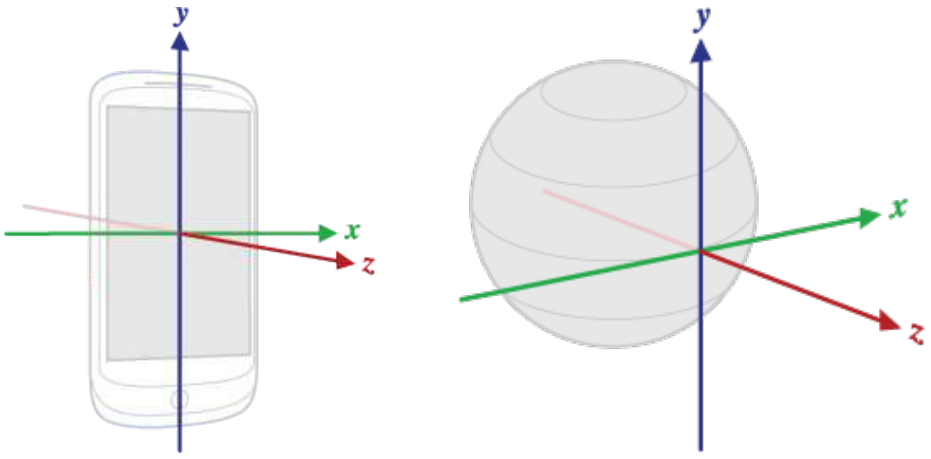
5.2 Frame of reference

An important choice while dealing with sensors that have spatial orientation is the frame of reference. Two frames had to be chosen. A local one for the phone and its sensors and a global one for the user and his environment. The chosen axes for the frames of reference are shown in Figure 5.1 and align with those defined in the Android sensor SDK. While magnetometer readings are taken in the phone's frame of reference it is important to transform those measurements to the global frame of reference since there the magnetic field is expected to be constant in close proximity.[2]

The transformation between those two frames is estimated by the orientation filter which forms an IMU based on the accelerometer and gyroscope. While the accelerometer provides information about two orientation angles, the gyroscope can be used to make the estimate more reliable. Additionally, with the gyroscope we can give an relative estimate about the horizontal orientation between two points in time. That angle will drift due to measurement errors which accumulate over time essentially generating a random walk.

In this thesis an algorithm designed by Madgwick was used to estimate the orientation of the phone. See Section 5.8 for further details.

[3] https://developers.google.com/protocol-buffers



(a) (b)

Figure 5.1: Chosen frames of reference. With (a) the *local frame of reference* or *device coordinates* and (b) the *global frame of reference* or *world coordinates*. Images gracefully taken from Google's Android SDK documentation.[2]

5.3 Software architecture

Since the implementation was growing fast in complexity, it was important to structure the project into logical and modular components. Many thoughts went into code architecture and design decisions which will be shortly discussed here.

The desired architecture had to deal with multiple programming languages and potential concurrency issues. Moreover, algorithms should be modules that only depend on their inputs and generate outputs, which makes it easy to exchange them and test them in a generalized way. Another advantage could be the ability to systematically detect bottle necks.

The various algorithms ended up being pluggable components in a pipelining scheme. Sensor and filtered data is passed through pipes from processing entity to processing entity. These processing entities were called nodes. The architecture was applied to the C++ code and the messaging protocol defined with Protocol Buffers. It was also applied in JavaScript but in a more simplified fashion. On Android, Java was used to call JNI wrapper functions to C++ and therefore did not require any particular architecture.

The overall architecture and different modules are shown in Figure 5.2.

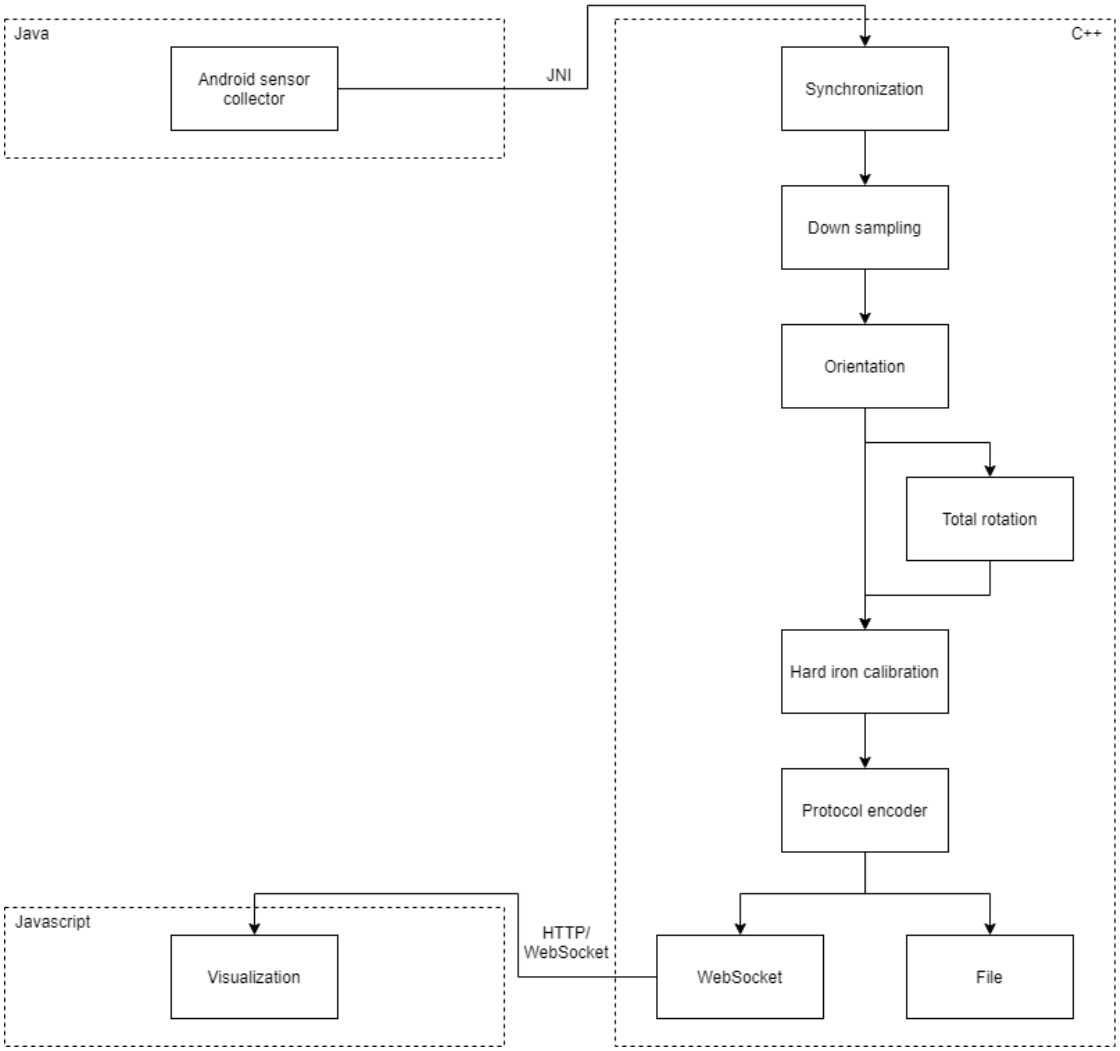


Figure 5.2: Illustration of the software architecture grouped by programming language.

5.4 Particle filter

In the following subsections we will walk through the different components of the particle filter. First, potential numerical issues with likelihoods are discussed. Second, the representation of a single particle in the filter is presented. Afterwards, the initialization, propagation, update, and resampling steps are discussed. Finally, we will see how the hidden states are estimated.

The bootstrap particle filter[32][34] was chosen as a template for the following implementation.

Initially, the particle filter was designed to model the complete decomposition given in Equation 3.3 and 3.4. Since $\boldsymbol{B}_{\text{earth}}$ and $\boldsymbol{B}_{\text{env}}$ transform in the same way under rotation of the device and we do not have a model for $\boldsymbol{B}_{\text{env}}$ this approach failed. The particle filter was further simplified to only decompose Equation 3.3.

5.4.1 Likelihood

In Bayesian statistics, likelihood is the key to model the update from a prior to a posterior distribution. Since we do this numerically, we have to be careful with the numerical limitations of the underlying machine. We expect single-precision floating-points as IEEE 754-1985 with the following properties: 1 sign bit, 8 exponent bits, 23 significant precision bits. This leads to an exponent int the range $[-126, 127]$. Since likelihood arithmetic involves multiplications with very small numbers we have to use the log likelihood instead.

To calculate the weights from the log likelihood we used the largest log likelihood involved and subtract it from all log likelihoods. This is possible because the weights are invariant under scalar multiplication as a relative measure and can be renormalized if necessary. Additionally this technique maps the weights to numbers in the range $[0, 1]$.

5.4.2 One particle

One particle represents the state of one realization of the hard iron calibration. Table 5.2 summarizes the properties of one particle. The calibration can be parameterized with a three dimensional vector representing the offset of the measurement from the true value. Another vector keeps the estimated external field, denoted by $\boldsymbol{B} = \boldsymbol{B}_{\text{measure}} - \boldsymbol{B}_{\text{phone}}$ with $\boldsymbol{B}_{\text{phone}}$ as the hard iron offset. The external field will later be compared to the next estimate to check for consistency. We also store the LOGLIKELIHOOD later also denoted by ℓ and the WEIGHT later also denoted by $\mathcal{W}$. They are related to each other but it is handy to keep them separate. Moreover, this might have performance advantages since we do not need to convert between them all the time.

5.4.3 Initialization

The particle filter has to be initialized with a prior distribution for the hard iron calibration. Pseudocode for the initialization step is given in Algorithm 5.1. Without making any further assumptions, a Gaussian distribution $\mathrm{N}(0, \sigma^2)$ was chosen

Field	Description
HARDIRON	The hard iron offset represented by this particle.
EXTERNAL	The external magnetic field $\boldsymbol{B}$ consistent with the last measurement.
LOGLIKELIHOOD	Logarithm of the likelihood ℓ of this particle.
WEIGHT	The weight $\mathcal{W}$ of this particle which describes how likely this realization is. It is a relative measure to the total weight among all particles.

Table 5.2: Description of the properties of one particle.

to sample each component of the hard iron vector. In order to enforce the constraint $\boldsymbol{B}_{\text{measure}} = \boldsymbol{B}_{\text{phone}} + \boldsymbol{B}$ (see Equation 3.3), we need an initial magnetometer observation $\boldsymbol{B}_{\text{measure}}$.

Two potential improvements to the initialization were identified. First, if the device was calibrated in the past, we could choose that as the center of our distribution or mix it with the one at the origin. Second, the current calibration of the operating system might also be a reasonable starting point of our calibration.

Algorithm 5.1: Initialization of the particle filter as pseudocode.

Input: population size N, hard iron variance σ^2, magnetic field measurement $\boldsymbol{B}_{\text{measure}}$, orientation $\boldsymbol{q}$
Output: initialized and populated particle filter P

```
1 P ← AllocateParticles(N)
2 foreach particle in P do
3 |   B_phone ← GaussianDrawThree(0, σ²)
4 |   B ← q × (B_measure − B_phone)
5 |   ℓ ← − log N
6 |   W ← 1/N
7 end
8 return P
```

5.4.4 Prediction

In order to predict future states of our particles we have to apply the state transition. Pseudocode for the propagation step can be found in Algorithm 5.2. Since we do not have a physical model for the time evolution of the hard iron effect,

the idea was to apply a random walk for each particle. This approach allows the particle filter to do recalibration over time.

Algorithm 5.2: Prediction step of the particle filter as pseudocode.

Input: population P, time passed t, hard iron variance drift rate σ^2
Output: population P

```
1 foreach particle in P do
2 |  B_phone ← B_phone + GAUSSIANDRAWTHREE(0, t * σ²)
3 end
4 return P
```

5.4.5 Update

We update our particle filter by feeding it with new observations from the magnetometer and new estimates of the orientation filter (see Section 5.8). Pseudocode for the update step is given in Algorithm 5.3. We predict the measurement of the magnetic field $\boldsymbol{B}_{\text{measure}}$ by transforming the previous estimate of the external field $\boldsymbol{B}$ into the local frame of reference and adding the hard iron vector $\boldsymbol{B}_{\text{phone}}$. Based on that prediction $\hat{\boldsymbol{B}}_{\text{measure}}$ and the measurement $\boldsymbol{B}_{\text{measure}}$, a likelihood can be calculated. That likelihood was modeled by a Gaussian distribution $N(0, \sigma^2)$. σ^2 can be used to parameterize the noise of the sensor and the variation of the magnetic field between two different points in space.

Algorithm 5.3: Update step of the particle filter as pseudocode.

Input: population P, magnetic field measurement $\boldsymbol{B}_{\text{measure}}$, orientation $\boldsymbol{q}$, prediction variance σ^2
Output: population P

```
1 foreach particle in P do
2 |  B̂_measure ← B_phone + q⁻¹ × B
3 |  ℓ ← ℓ+ GAUSSIANLOGPDF(B̂_measure, B_measure, σ²)
4 |  B ← q × (B_measure − B_phone)
5 end
6 return P
```

5.4.6 Resampling

The resampling step is the most important part of the particle filter. Without it the likelihood would converge to zero and the variance of the weights would only

increase. The idea is to remove unlikely states and replace them with more likely ones. The chosen algorithm, called multinomial resampling[32][36], is a common resampling algorithm with a time complexity of $\mathcal{O}(n \ln n)$ and memory complexity of $\mathcal{O}(n)$.[37]

Pseudocode for the resampling step is given in Algorithm 5.4. First we calculate the weight of each particle $\mathcal{W}$ by its likelihood ℓ. Then we accumulate these weights into an array of partial sums w_i. Now we can draw uniformly distributed samples x from the range 0 to total weight $\mathcal{W}_{sum}$ and lookup the lower bound elements in the array. Thereby we get the indices i of the particles we are going to sample from. We copy the state of those particles P^i_{in} into a second array and afterwards exchange it with the previous array of particles (double buffering).

Since the propagation step applies a random walk to the hard iron offset, we do not have to worry about state degeneracy.

Algorithm 5.4: Resampling step of the particle filter as pseudocode.

Input: population P_{in}, population size n
Output: population P_{out}

$\ell_{max} \leftarrow \text{MAX}(\ell)$
$\mathcal{W}_{sum} \leftarrow 0$
$w \leftarrow \text{ALLOCATEFLOATS}(n)$
foreach p *in* P_{in} **do**
    $\mathcal{W} \leftarrow e^{\ell - \ell_{max}}$
    $\mathcal{W}_{sum} \leftarrow \mathcal{W}_{sum} + \mathcal{W}$
    $w_i \leftarrow \mathcal{W}_{sum}$
end
foreach p *in* P_{out} **do**
    $x \leftarrow \text{RANDOM}(0, \mathcal{W}_{sum})$
    $i \leftarrow \text{LOWERBOUNDS}(w, x)$
    $p \leftarrow P^i_{in}$
end
return P_{out}

5.4.7 Estimation

In the estimation step we want to measure the hidden state of our system. The state of the particle filter is represented by its population, the particles, and their states and weights.

Pseudocode for the estimation step is given in Algorithm 5.5. In order to get an estimate of the hard iron vector and its error, a weighted average and weighted covariance matrix is calculated from the population of the filter. The covariance matrix can later be used to quantify the convergence of the filter.

Algorithm 5.5: Estimation step of the particle filter as pseudocode.

Input: population P

Output: hard iron vector $\hat{\boldsymbol{B}}_{phone}$, hard iron covariance $\hat{\boldsymbol{\Sigma}}_{phone}$, external magnetic field vector $\hat{\boldsymbol{B}}$, external magnetic field covariance $\hat{\boldsymbol{\Sigma}}$

1 $\hat{\boldsymbol{B}}_{phone} \leftarrow$ WEIGHTEDAVERAGE(P, $\mathcal{W}$, $\boldsymbol{B}_{\text{phone}}$)
2 $\hat{\boldsymbol{\Sigma}}_{phone} \leftarrow$ WEIGHTEDCOVARIANCE(P, $\mathcal{W}$, $\boldsymbol{B}_{\text{phone}}$)
3 $\hat{\boldsymbol{B}} \leftarrow$ WEIGHTEDAVERAGE(P, $\mathcal{W}$, $\boldsymbol{B}$)
4 $\hat{\boldsymbol{\Sigma}} \leftarrow$ WEIGHTEDCOVARIANCE(P, $\mathcal{W}$, $\boldsymbol{B}$)
5 **return** $\hat{\boldsymbol{B}}_{phone}$, $\hat{\boldsymbol{\Sigma}}_{phone}$, $\hat{\boldsymbol{B}}$, $\hat{\boldsymbol{\Sigma}}$

5.5 Filter routine

Our filter routine for the hard iron calibration is a combination of all the steps described in Section 5.4. The parameters of the filter are summarized in Table 5.3.

Parameter	Description
SEED	Seed for the pseudorandom number generator.
POPULATION	The total number of particles.
DELTATIME	The time difference in seconds between two filter iterations.
INITIALVARIANCE	The initial hard iron variance for the initialization procedure in $(\mu T)^2$.
DRIFTRATE	The hard iron calibration drift rate for the propagation procedure.
PREDICTIONVARIANCE	The prediction variance for the update procedure in $(\mu T)^2$.
MINIMALROTATION	Minimal rotation angle to perform an update in radians.
RESAMPLINGRATE	Relative amount of effective particles to trigger resampling.

Table 5.3: Parameter description of the filter.

Pseudocode of the filter routine is given in Algorithm 5.6. We initialize the filter with the first observations of the magnetometer and orientation filter. Then we

update and propagate based on time passed and newly incoming observations. The filter will only update after significant rotations which is parameterized by MINIMALROTATION. This has the benefit of saving Central Processing Unit (CPU) consumption and filtering noise of the orientation estimation which could result in biased hard iron estimates. Adaptive resampling (see Equation 4.6 in Section 4.4) is used to reduce noise, might spare CPU usage and can be parameterized with RESAMPLINGRATE.

Algorithm 5.6: The filter routine as pseudocode.

Input: magnetic field measurement $\boldsymbol{B}_{\text{measure}}$, orientation $\boldsymbol{q}$
Output: calibrated magnetic field $\hat{\boldsymbol{B}}$, calibrated magnetic field covariance $\hat{\boldsymbol{\Sigma}}$

```
if NOTINITIALIZED() then
    P ← INIT(B_measure, q)
    ESTIMATE()
end
if SIGNIFICANTROTATION() then
    UPDATE(B_measure, q)
    B̂_phone, Σ̂_phone ← ESTIMATE()
    if EFFECTIVEPARTICLES() < POPULATION × RESAMPLINGRATE then
        RESAMPLE()
    end
end
B̂ ← B_measure − B̂_phone
Σ̂ ← Σ̂_phone
return B̂, Σ̂
```

5.6 Synchronization

Our work targets mobile devices with Android and iOS. Since those are no real-time systems one has to deal with unpredictable latency and concurrency. These effects play a role while receiving data of multiple sensors, possibly through multiple threads. The receiver might observe that some events are out of order. Recursive filters rely on the causal order of the events because the state of the filter evolves with each observation and its timestamp. They cannot include events that are older than the current state. When we combine multiple sensor streams into one result, we have to be careful with potential latency differences given by each stream.

This problem was solved with a synchronization step at the beginning of the pipeline (see Figure 5.2). Buffers are used to hold all the incoming sensor data

until it is guaranteed that all streams reached the same point in time. Additionally, this guarantees that any other component after the synchronization will only receive events in order and does not have to deal with synchronization by itself.

5.7 Down sampling

As mentioned in Section 5.6, we cannot rely on the properties of a real-time system. In case of Android, the requested sampling rate passed to the the sensors Application Programming Interface (API) is treated as a hint and can differ greatly. Since the orientation estimation (see Section 5.8) relies on a constant sampling rate, this problem had to be resolved by implementing a down sampling algorithm. The chosen algorithm was a moving average.

The sampling interval was measured during the evaluation in Section 6.2.

5.8 Orientation estimation

We saw that our hard iron calibration highly depends on the orientation estimation of the device in Section 5.4. Errors attached to the orientation estimates will directly propagate into errors of the hard iron calibration. Therefore it was crucial to use a well suited algorithm to estimate the orientation with high precision. These algorithms typically use the accelerometer and the gyroscope to form an IMU.

In our implementation we relied on Madgwick's IMU algorithm[38] which estimates the orientation by fusing accelerometer and gyroscope. It claims to have low computational effort and higher accuracy than Kalman-based algorithms. The algorithm is illustrated in Figure 5.3. The orientation filter has only a single parameter BETA, apart from the update interval, which models the reliability proportion between accelerometer and gyroscope measurements.

Implementations for various programming languages are available online[4], including an implementation for C. This C implementation was adopted for C++ and slightly modified. One of the modifications was the replacement of the "fast inverse square-root" by a more conventional implementation. The "fast inverse square-root" was producing biases and the computational performance was not too critical in our case anyway.

[4]https://x-io.co.uk/open-source-imu-and-ahrs-algorithms

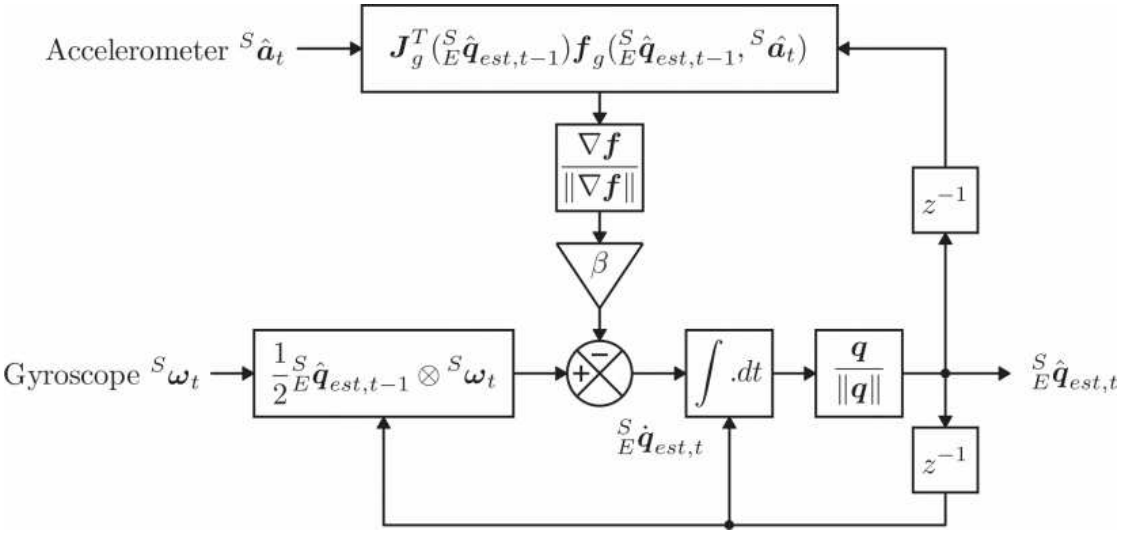


Figure 5.3: Block diagram representation of the orientation estimation algorithm. Image gracefully taken from Madgwick's paper.[38]

With an IMU based on accelerometer and gyroscope, the x and y-axis in world coordinates (see Figure 5.1) will be rotated by an unknown angle around the z-axis. This is due to missing information of the horizontal orientation, which can only be observed by the magnetometer (in case of those three sensors). Moreover, this angle will be drifting due to accumulated errors coming from the gyroscope.

5.9 Total rotation

Since our hard iron calibration can only progress if the device is rotated, we might want to quantify the total rotation and use it instead of time as an axis for our prototyping and evaluation plots. This was achieved by calculating the Euler axis and angle between two orientation estimates and a sum over these angles.

The quaternion $\boldsymbol{\Delta q}_i$ that transforms the previous orientation $\boldsymbol{q}_{i-1}$ to the current orientation $\boldsymbol{q}_i$ can be expressed by the following equation.

$$\boldsymbol{q}_i = \boldsymbol{\Delta q}_i \times \boldsymbol{q}_{i-1} \iff \boldsymbol{\Delta q}_i = \boldsymbol{q}_i \times \boldsymbol{q}_{i-1}^{-1} \tag{5.1}$$

The Euler angle can then be extracted with the extended Euler's formula.

$$\Delta\omega_i = 2 \arccos \mathrm{Re}(\boldsymbol{\Delta q}_i) \tag{5.2}$$

With $\mathrm{Re}(\boldsymbol{\Delta q}_i)$ as the real component of the quaternion $\boldsymbol{\Delta q}_i$.

The summation over $\Delta\omega_i$ is our desired quantity "total rotation". This quantity was also used to filter for significant rotations in the filter routine of our hard iron calibration.

5.10 Android

Java was used to develop a small demo application that is able to read sensor data and pass it down to C++ through JNI. A WebView was used to display the visualization written in JavaScript. The whole life cycle of the application was managed in Java.

The Android sensor API offers a high-level interface to read sensor values by registering to a specific sensor type, like accelerometer, providing a sampling rate, and a callback for incoming data. The API is presenting calibrated and uncalibrated sensor readings in the same way and is distinguishing between them by different identifiers. TYPE_MAGNETIC_FIELD can be used to read the system calibrated magnetic field and TYPE_MAGNETIC_FIELD_UNCALIBRATED for the uncalibrated magnetic field, which is required by this thesis.[39]

From experience we know that the sensor API will not sample the sensors at a constant rate. Since a constant sampling rate is a requirement for the orientation filter, this problem had to be resolved as discussed in Section 5.7.

A screenshot of the demo application is shown in Figure 5.4.

5.11 Visualization

One goal of the visualization was to make it platform independent and to use the same code for desktop and mobile devices. This was beneficial because the visualization could also be driven by simulated and real-time sensor data.

The initial idea was to use OpenGL ES for visualization purposes. Android and common desktop OS support OpenGL. One visualization idea was a three-dimensional scatter plot for the acquired magnetic field sensor data to build up intuition for the data we are going to filter. This might require to plot thousands of data points in real time. Since OpenGL is executed by the Graphics Processing Unit (GPU) there were no performance concerns.

Since implementation progress was rather slow and the effort was growing quickly

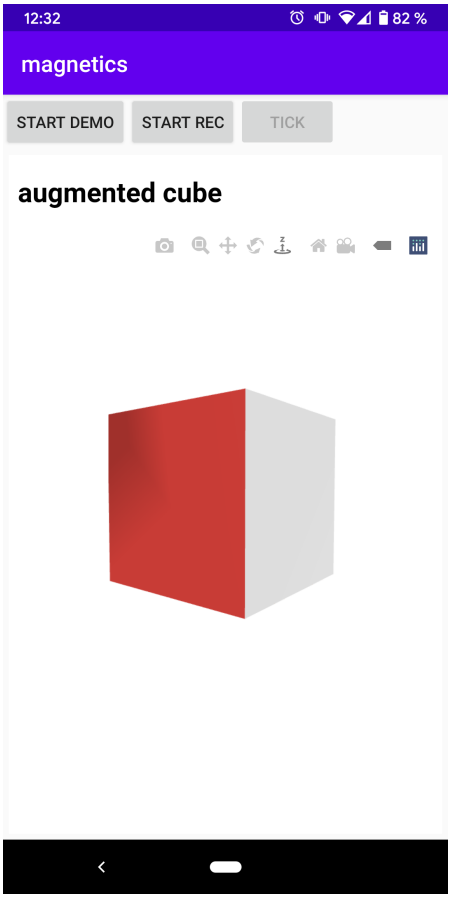


Figure 5.4: Screenshot for the Android demo application.

the desire for an alternative was growing. JavaScript with plotly seemed to be a good solution for platform independence. It also claimed to have good performance with a WebGL backend for 3D plots.

The HTML and JavaScript code was stored as an asset along the demo application and was displayed by a WebView on Android. On the desktop it is sufficient to use any modern browser to run the visualization.

6 Empirical Evaluation

In this chapter, we evaluate our implementation of the hard iron calibration presented in Chapter 5. First, we give an overview of the hardware used for evaluation. Then we take a look at the behaviour of the sensor data collection and a qualitative evaluation of the orientation filter. Finally, we will evaluate the quality of our hard iron calibration in comparison to the one of the system, the error estimation, and its performance.

6.1 Hardware

Four different devices were available for evaluation. They are listed in Table 6.1. It was important to compare the results between multiple devices since sensors and hard iron calibration algorithms vary between them. Also, we need to be sure that the performance of our implementation is good enough for real-time processing across different devices to achieve maximum compatibility.

All these devices run a fairly recent Android version and have been released during the last 5 years.

Name	Release year	Android version
Google Pixel 3	2018	11
Google Pixel 2	2017	11
Samsung Galaxy S7	2016	8
LG Nexus 5X	2015	8.1

Table 6.1: Devices used for testing and evaluation.

Another possible test device was the Nokia 3 released in 2017 but unfortunately it does not provide uncalibrated magnetic field sensor readings. Up to this point we are not sure about the market share of devices that support uncalibrated readings. Alternatives will be discussed in Chapter 8.

6.2 Data collection

The validation of the sensor data collection was a first and important step for evaluation, since all the following steps depend on it. Moreover, it was beneficial for building up intuition for the data which is used for filtering.

This experiment was carried out on all the Android test devices listed in Table 6.1. The requested sampling interval was 1 ms.

As a first step, only the timestamps of the magnetometer readings were used to calculate the measurement interval. Since Android is not a real-time system the interval will not be constant. A histogram of the interval is shown in Figure 6.1. Statistics are given in Table 6.2.

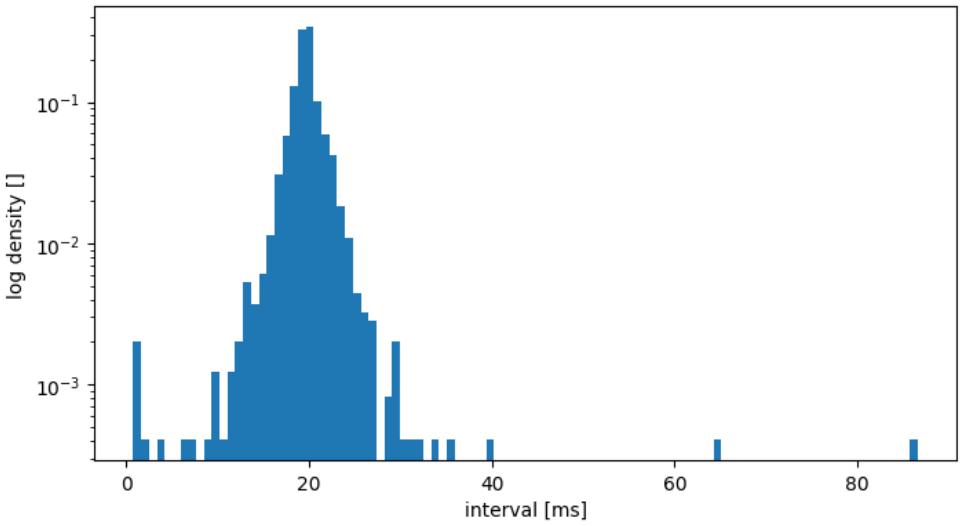


Figure 6.1: Histogram of the of the sampling interval for the magnetometer in the LG Nexus 5X.

Device	mean	std	min	Q_1	median	Q_3	max
Google Pixel 3	10.0	3.2	0.1	9.0	10.0	11.2	57.4
Google Pixel 2	9.6	1.6	0.0	9.2	9.6	9.9	63.9
Samsung Galaxy S7	10.0	2.6	0.3	9.3	10.0	10.7	115.5
LG Nexus 5X	19.7	2.6	0.8	19.0	19.7	20.4	86.5

Table 6.2: Statistics of the magnetometer reading interval in milliseconds.

Secondary, the readings of the magnetometer were displayed as a three-dimensional scatter plot to reveal the hard iron effect. For this purpose the device has been

rotated in multiple directions while reading the magnetic field. Two-dimensional projections of such a three-dimensional scatter are shown in Figure 6.2 for the Samsung Galaxy S7.

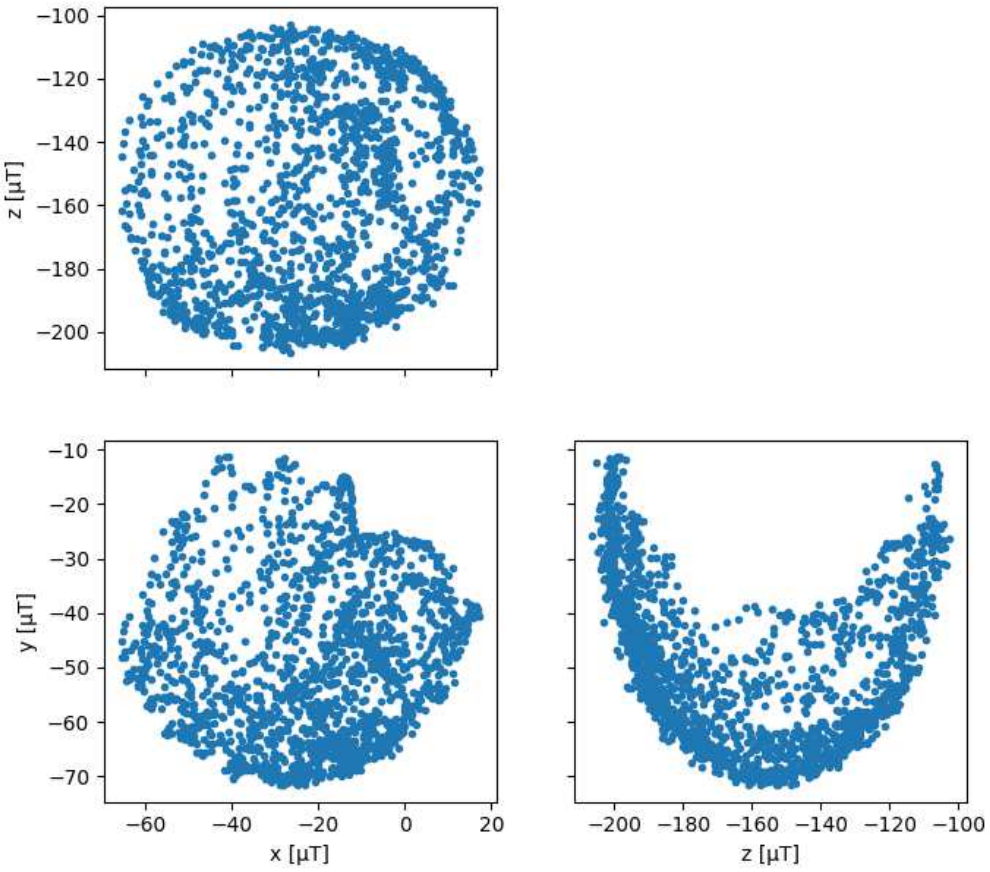


Figure 6.2: Scatter plot of the magnetometer readings while rotating the Samsung Galaxy S7 in multiple directions.

Without the hard iron effect we would expect the center of the plots at the origin. The maximal distance from the origin should be the magnitude of the Earth's magnetic field if other environmental sources are neglectable. Measuring the horizontal orientation involves $\arctan2(y, x)$ for a two dimensional projection of the magnetic field. As we can see in Figure 6.2, without considering the hard iron effect, the estimation of the horizontal orientation would be meaningless since (x, y, z) are not zero-centered.

6.3 Orientation filter

Since our hard iron calibration depends highly on the orientation estimation, it was necessary to validate the implementation and to evaluate its performance. A quantitative evaluation would require reference values for the orientation of the device for comparison. This could be achieved by tracking visual landmarks with a camera, rotating the device with a robotic arm or simulation of sensor data. Such experiments were carried out at the Department of Geodesy and Geoinformation at TU Wien in order to evaluate their Kalman-based orientation estimation.[40] However, using their equipment and data was not part of this thesis.

The implementation of the orientation filter required at least a qualitative evaluation to guarantee its functionality. This was carried out with an augmented reality demo application. A cube was placed at the origin of a 3D scene and the position and orientation of the camera in this scene were calculated based on the orientation of the phone in such a way, that the cube seems to be stationary in world coordinates. This approach validates the transformation between the local and global frames of reference and qualitatively evaluates the response to sensor updates.

The 3D scene is illustrated in Figure 6.3. Pictures of the augmented cube rendered by the LG Nexus 5X are show in Figure 6.4.

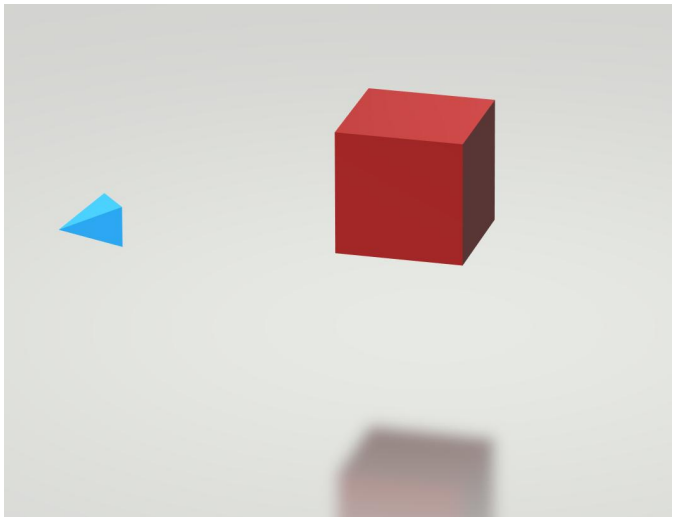


Figure 6.3: Illustration of the cube and the camera in the 3D scene.

The accelerometer and gyroscope have been down sampled to a 50 ms interval. Therefore, some delay and low refresh rate was expected. However, the experience was very positive.

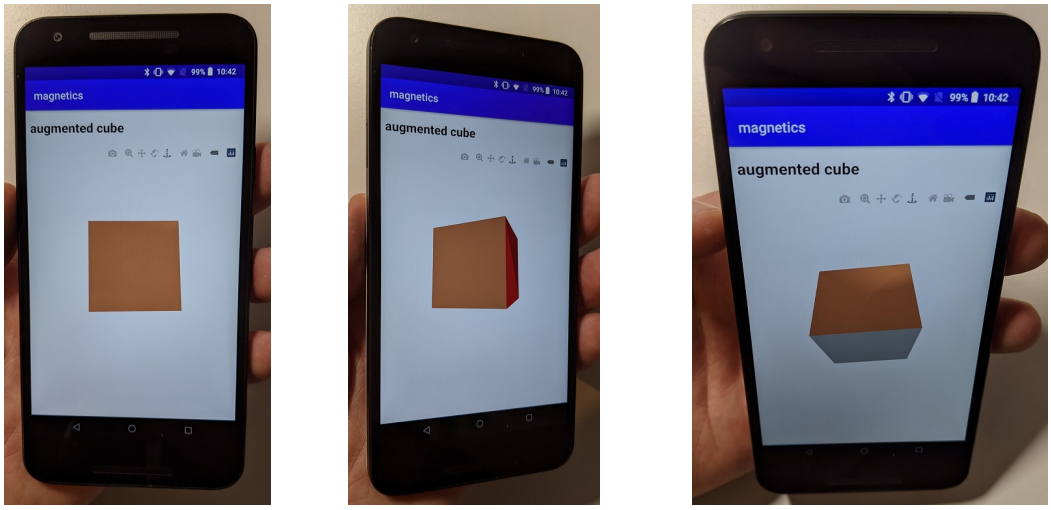


Figure 6.4: Pictures of the augmented cube after rotating the device.

The orientation filter has only a single parameter BETA, apart from the update interval. BETA models the reliability proportion between accelerometer and gyroscope measurements. This parameter has to be chosen carefully and in the best case individually for each sensor composition or smartphone model, since quality of the estimates depends highly upon that. Choosing one parameter across multiple devices might result in low accuracy or biased estimated.

Our demo application can be used to hand-tune BETA for a given device. One could chose a parameter to start with and test it in the application. By observing the performance one might decide that the parameter is not optimal and change it. Following this process iteratively until the results are satisfying will lead to an optimized value for BETA.

6.4 Hard iron calibration

A test track was set up in order to evaluate our hard iron calibration. It is illustrated in Figure 6.5. Visual landmarks were placed at each turning point as a spatial reference for the collected sensor data across different devices. The track had a total length of approximately 24 meters with 4 turns and a total horizontal rotation angle of approximately 540 degrees. A button in the test application, called TICK (see Figure 5.4), was used to mark the passing of each landmark.

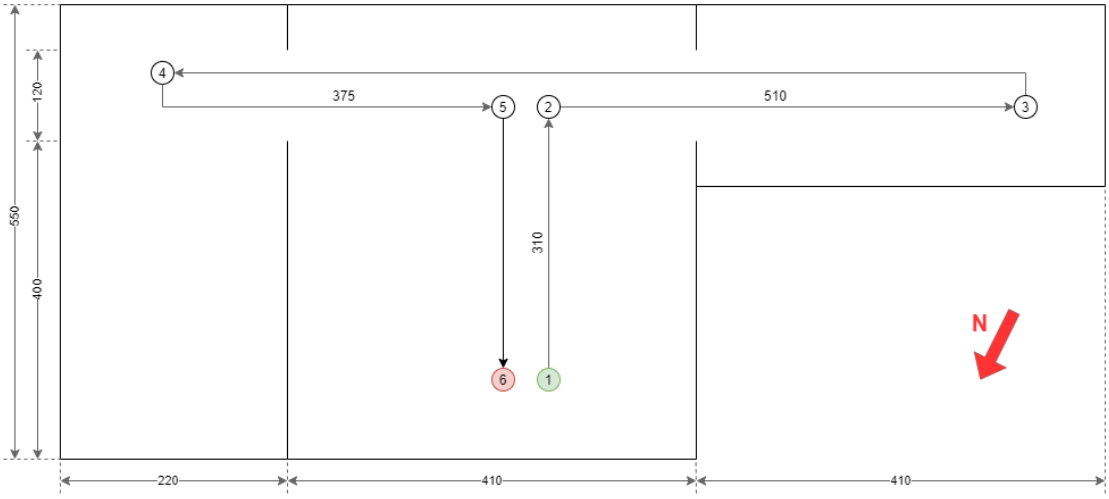


Figure 6.5: Illustration of the test scenario for the hard iron calibration.

Each test run was a sequence of three different phases. In the first phase the test device was placed on a table. Thereby, the orientation estimation would initialize in the same way across different test runs. Then a magnet was used in order to manipulate the hard iron effect of the test device. The second phase started by pressing the TICK button in the application, followed by walking the test track. The average walking speed was approximately 1 m/s. The third phase started with a last press of the TICK button at landmark 6. Then the test device was rotated in multiple directions on the same position. With the third and last phase we collected a large amount of reference data for an optimal magnetometer calibration.

The test application would be started with the button START REC (see Figure 5.4), an abbreviation for "start recording". By pressing this button, our application starts to record all the incoming data from the sensors. Our TICK button is also treated like a sensor in this case. Each sensor reading is then stored in a file, along with its timestamp. The recorded data was later used to simulate the sensor input and to evaluate the output of our filter. Table 6.3 contains the parameters of our filter used during the simulation.

Parameter	Value
SEED	random
POPULATION	10^5
DELTATIME	50 ms
INITIALVARIANCE	$(100\ \mu T)^2$
DRIFTRATE	1.0
PREDICTIONVARIANCE	$(5\ \mu T)^2$
MINIMALROTATION	0.1
RESAMPLINGRATE	0.01

Table 6.3: Chosen parameters for the simulation.

The data of the third phase of our test runs was used to estimate an independent reference for the hard iron calibration. The chosen estimator was a least squares fit for a sphere with an unknown origin and radius.[41]

Figures 6.6, 6.7, 6.8, and 6.9 show the estimated hard iron bias and error of our particle filter over time compared to the least squares estimate. The vertical lines represent the time when a landmark was passed on the test track.

Tables 6.4, 6.5, 6.6, and 6.7 contain the values of the estimated hard iron bias and error across different methods after passing the landmarks on the test track. PF is our real-time particle filter estimate, SYS is the system's estimate, and LS the post processed least squares estimate. Phase 2.x denotes passing the landmark x on the test track. Phase 3 denotes the end of the experiment after rotating the phone in multiple directions.

In all our test cases the OS was not able to calibrate the hard iron effect while walking on the test track. Only in the last phase of the experiment, when the device was rotated in multiple directions. The estimates of the systems were very close to the least squares estimates.

Figure 6.6 and Table 6.4 show the convergence of the hard iron estimation for the Google Pixel 3. Our particle filter calibration performed very well, but picked up a bias on the z-axis after landmark 4 (phase 2.4 in the table). During the calibration phase at the end of the experiment there was a bias on the x-axis.

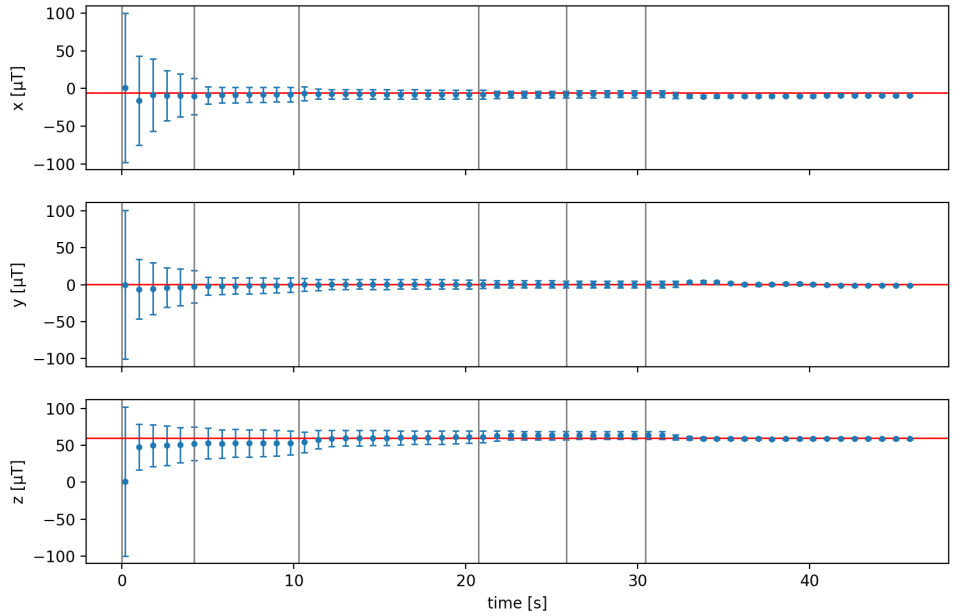


Figure 6.6: Estimated hard iron bias and error over time on the Google Pixel 3.

Method	Axis	Phase 2.1	Phase 2.2	Phase 2.3	Phase 2.4	Phase 2.5	Phase 2.6	Phase 3
PF	X	-0.5 ± 100.5	-9.3 ± 24.0	-5.8 ± 7.2	-5.3 ± 4.6	-5.3 ± 3.9	-4.9 ± 3.7	-8.4 ± 1.1
SYS		1.0	1.0	1.0	1.0	1.0	1.0	-6.4
LS								-5.4
PF	Y	-0.6 ± 100.4	-3.7 ± 21.6	-3.3 ± 8.8	-2.4 ± 5.0	-2.3 ± 4.1	-2.4 ± 4.0	-0.9 ± 1.1
SYS		32.6	32.6	32.6	32.6	32.6	32.6	1.3
LS								0.3
PF	Z	1.2 ± 100.8	52.1 ± 22.5	53.4 ± 15.4	61.5 ± 8.1	63.3 ± 5.8	63.7 ± 5.4	58.9 ± 1.1
SYS		8.4	8.4	8.4	8.4	8.4	8.4	59.4
LS								60.1

Table 6.4: Estimated hard iron bias in μT with different methods on the Google Pixel 3.

Figure 6.7 and Table 6.5 show the convergence of the hard iron estimation for the Google Pixel 2. Our particle filter calibration looks unbiased until landmark 6 (phase 2.6 in the table) but picked up a bias during the calibration phase at the end. Here we got a small bias for the x- and y-axis.

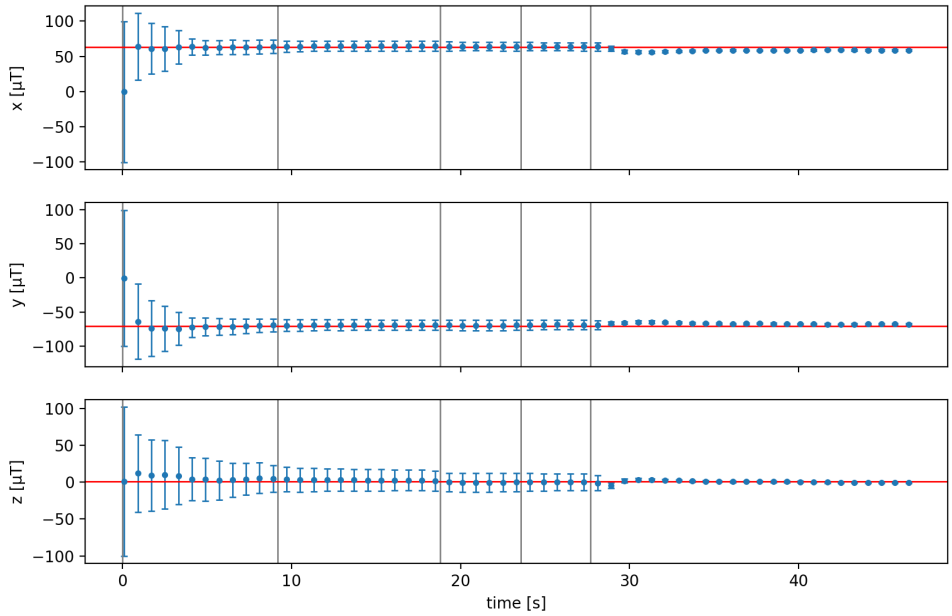


Figure 6.7: Estimated hard iron bias and error over time on the Google Pixel 2.

Method	Axis	Phase 2.1	Phase 2.2	Phase 2.3	Phase 2.4	Phase 2.5	Phase 2.6	Phase 3
PF	X	1.4 ± 99.7	62.3 ± 9.1	63.4 ± 6.7	62.7 ± 5.1	62.6 ± 4.7	59.8 ± 1.7	58.7 ± 1.0
SYS		63.6	63.6	63.6	63.6	63.6	63.6	62.0
LS								63.3
PF	Y	0.9 ± 100.1	-68.7 ± 11.4	-70.0 ± 7.6	-69.1 ± 6.2	-68.8 ± 5.8	-64.2 ± 1.9	-67.9 ± 1.0
SYS		−45.4	−45.4	−45.4	−45.4	−45.4	−45.4	−71.5
LS								−71.2
PF	Z	1.1 ± 99.9	1.4 ± 17.7	-1.2 ± 11.7	-0.5 ± 9.2	-0.0 ± 8.2	-3.1 ± 1.6	-0.2 ± 1.0
SYS		−27.4	−27.4	−27.4	−27.4	−27.4	−27.4	−0.6
LS								0.7

Table 6.5: Estimated hard iron bias in μT with different methods on the Google Pixel 2.

Figure 6.8 and Table 6.6 show the convergence of the hard iron estimation for the Samsung Galaxy S7. The hard iron effect of the z-axis was about 150 μT and therefore quite challenging. The filter would not initialize a lot of particles in close proximity since the INITIALVARIANCE is just $(100\ \mu T)^2$. The y-axis picked up a bias most likely due to the same reason as the z-axis. During calibration phase the accuracy increases but a small bias remained on the x- and y-axis.

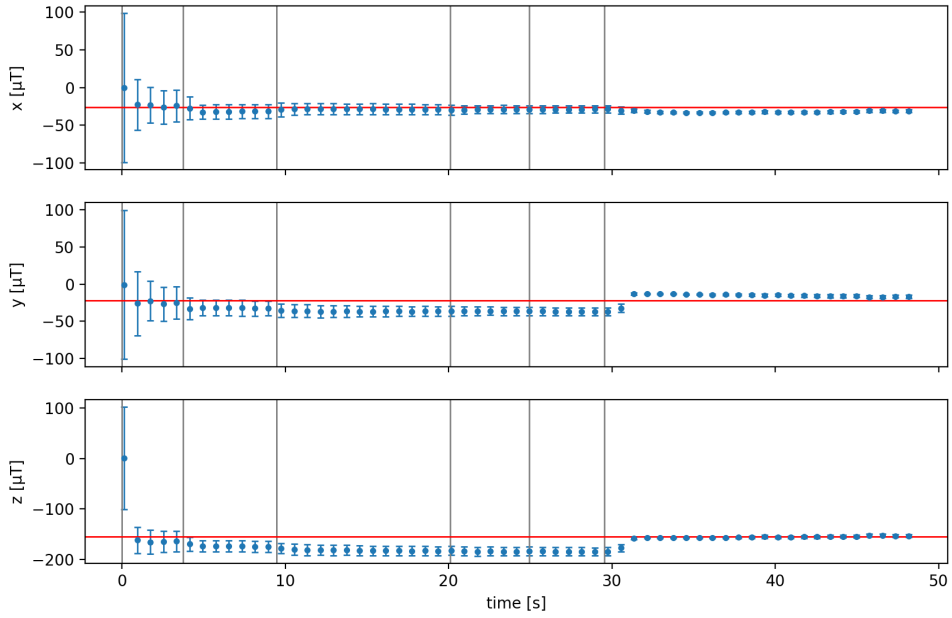


Figure 6.8: Estimated hard iron bias and error over time on the Samsung Galaxy S7.

Method	Axis	Phase 2.1	Phase 2.2	Phase 2.3	Phase 2.4	Phase 2.5	Phase 2.6	Phase 3
PF	X	1.1 ± 100.0	-20.9 ± 17.2	-24.1 ± 7.0	-24.9 ± 5.5	-25.8 ± 4.4	-26.1 ± 4.2	-27.2 ± 1.1
SYS		0.0	0.0	0.0	0.0	0.0	0.0	-25.6
LS								-26.1
PF	Y	1.8 ± 100.2	-27.0 ± 19.8	-32.4 ± 7.6	-31.9 ± 4.5	-32.0 ± 3.3	-32.1 ± 3.2	-20.3 ± 1.1
SYS		0.0	0.0	0.0	0.0	0.0	0.0	-23.6
LS								-21.9
PF	Z	0.9 ± 99.9	-163.8 ± 16.7	-167.7 ± 8.9	-169.8 ± 6.4	-170.7 ± 5.1	-171.1 ± 4.9	-154.9 ± 1.2
SYS		0.0	0.0	0.0	0.0	0.0	0.0	-154.8
LS								-154.6

Table 6.6: Estimated hard iron bias in μT with different methods on the Samsung Galaxy S7.

Figure 6.9 and Table 6.7 show the convergence of the hard iron estimation for the LG Nexus 5X. Our calibration performed reasonably well and remained within $\pm 1\sigma$ until calibration phase. Our particle filter seemed to pick up oscillations from the rotations of the devices. This might be due to a bias in the orientation estimation. BETA might not have been chosen optimal in this case.

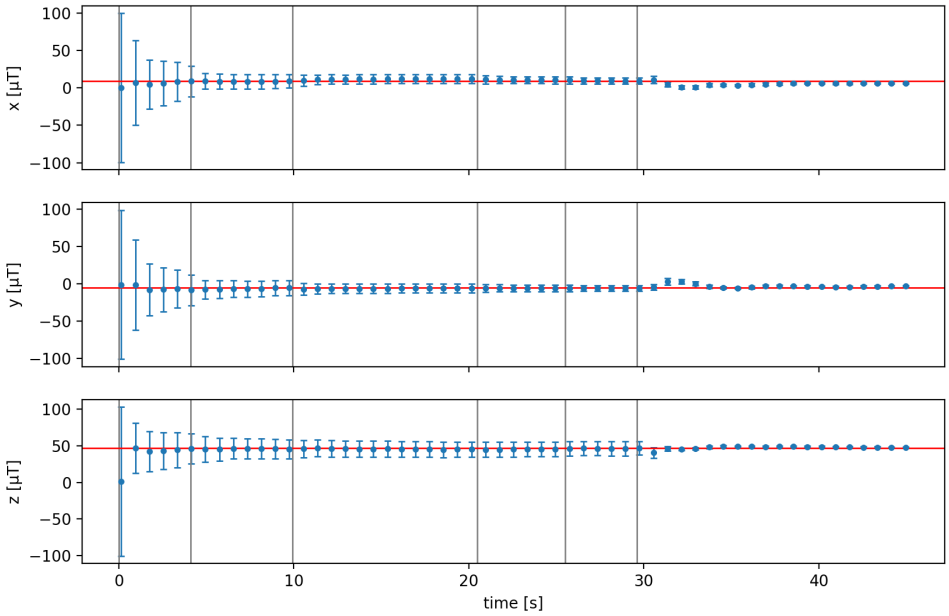


Figure 6.9: Estimated hard iron bias and error over time on the LG Nexus 5X.

Method	Axis	Phase 2.1	Phase 2.2	Phase 2.3	Phase 2.4	Phase 2.5	Phase 2.6	Phase 3
PF	X	0.4 ± 99.4	9.9 ± 21.1	7.1 ± 9.6	11.6 ± 6.1	11.3 ± 4.4	10.4 ± 4.2	2.9 ± 1.1
SYS		-4.4	-4.4	-4.4	-4.4	-4.4	-4.4	10.6
LS								9.2
PF	Y	2.0 ± 99.8	-8.8 ± 20.9	-5.9 ± 10.6	-6.4 ± 6.6	-6.7 ± 5.1	-6.3 ± 4.6	-2.6 ± 1.2
SYS		-47.8	-47.8	-47.8	-47.8	-47.8	-47.8	-6.7
LS								-5.1
PF	Z	-1.5 ± 100.1	45.9 ± 20.7	44.8 ± 14.3	40.6 ± 10.3	39.2 ± 8.2	39.8 ± 7.7	48.1 ± 1.4
SYS		42.8	42.8	42.8	42.8	42.8	42.8	47.2
LS								47.0

Table 6.7: Estimated hard iron bias in μT with different methods on the LG Nexus 5X.

In all our test cases the particle filter was able to converge. The estimated error dropped to approximately $\pm$ 6 μT after 20 seconds while producing biases of about $\pm$ 10 μT in the worst case. In phase 3, the accuracy of our filter was worse compared to the OS and least squares estimates. However, our particle filter produced results when the other methods were not able to. The biases might come from a previous signal processing step like the moving average or the orientation

filter. Compared to the earth magnetic field the errors are reasonably small and should not affect the compass too dramatically.

6.4.1 Computational performance

Since we are targeting mobile devices and want to run our particle filter in real-time, the evaluation of the performance is indispensable. A fast algorithm has the benefit of being compatible with older and slower devices and will consume less battery which is a limited resource for mobile devices. Depending on the use case, the performance can be tweaked with the amount of particles. However, reducing the population of the particle filter will also reduce its stability. For a practical use case it is important to find a sweet spot with optimal performance and optimal stability.

The parameter MINIMALROTATION was set to zero in order to measure an upper bound for the CPU consumption. Otherwise the parameters were the same as shown in Table 6.3.

The performance measurements are summarized in Table 6.8. Each value is the result of a 10 second measurement of the CPU time spent in the particle filter divided by real-time passed.

Device \ Population	10^3	10^4	10^5
Google Pixel 3	0.00	0.06	0.51
Google Pixel 2	0.01	0.11	0.53
Samsung Galaxy S7	0.01	0.14	0.79
LG Nexus 5X	0.01	0.10	0.59

Table 6.8: CPU time spent in the particle filter divided by real-time passed per device and particle filter population.

7 Conclusions

The goal of this thesis was to overcome limitations of the OS hard iron calibration of smartphones in pedestrian navigation, localization and wayfinding scenarios, like IPS (more details in Section 1.4). We identified the following major limitations: different algorithms across different devices, dependence on gestures that have to be communicated to and performed by the user, no continuous calibration, and no quantified error estimation.

We proposed a particle filter to calibrate the hard iron effect of three-axis magnetometers in smartphones by sensor fusion with the accelerometer and gyroscope. Our algorithm can be updated iteratively and takes previous estimations into account. It does not depend on specific gestures that have to be communicated to the user. Any kind of rotations of the device are taken into account. Additionally, we can obtain a quantified error estimate for the hard iron calibration which can be used for error propagation in other filters that depend on the magnetic field (e.g. a compass).

A test plan was developed with a predefined track and different calibration phases to make the measurements of the different test devices comparable. We compared the results of the particle filter to those of the OS and to least squares estimates for reference.

Our particle filter is producing promising results with a reasonable amount of CPU usage. It was able to calibrate the hard iron effect just by walking on the test track while the OS was not able to produce any estimates. This promotes our algorithm to a viable solution for online calibration of magnetometers in smartphones in navigation, localization and wayfinding scenarios. There is still a lot of room for improvements which are discussed in Chapter 8.

As discovered during the work on this thesis, not all Android devices with a magnetometer will offer uncalibrated readings. Unfortunately, this limits the scope of application for our particle filter. Possible solutions are discussed in Chapter 8.

8 Outlook

In this chapter we want to discuss possible future work connected to this thesis. Besides implementation details that might improve performance directly, there is room for further research on the model and applications.

In the evaluation our particle filter was biased in some cases. Depending on the application this need further investigation. Animating the time evolution of the state space might help to understand the underlying problem.

Currently the time evolution of the hard iron effect is modelled as a random walk. A possible improvement would be to have an adaptive speed for the random walk depending on the strength of the external magnetic field. More advanced models might even be able to predict the influence of the external field on the hard iron effect.

One application of this work is to develop an unbiased compass for smartphones in an indoor environment. Because of magnetic construction materials, the measured magnetic field can differ highly from the Earth's magnetic field. Since the variation of the building's magnetic field acts on much smaller scales than the one of the Earth, it might be possible to distinguish between them. Calibrating the hard iron effect is a preliminary step that removes one bias from the Equations 3.3 and 3.4. Simple models for the building's magnetic field might be that it is small compared to the Earth's field or that it is very chaotic and might cancel out in summation over a long path.

As discovered during the work on this thesis, not all Android devices with a magnetometer will offer "uncalibrated" readings, which do not contain hard iron compensation. Unfortunately, this limits the scope of application for our particle filter. This is true for at least the Nokia 3 and the Motorola Moto G3. Using the system's calibrated magnetometer readings for further calibrations requires caution, since it can be a new source of bias. It seems that the system calibration generates discontinuous jumps in the signal. It should be possible to filter these jumps and to propagate them into the state of the particle filter.

8.1 Computational performance

The particle filter is highly parallelizable because the particles are mostly independent. Only the resampling step usually requires interaction since a sum over all weights is performed.[36]

Modern CPUs come with special instructions for vectorization. Multiple operations, like vector addition, can be performed at the same time. This can speed up the execution time significantly if this happens on a critical path. Compiler Explorer[42] can be used to check if such instructions are generated by the compiler.

Modern smartphones usually come with four processor cores or more. Those can be utilized by multithreading effectively. Usually that also requires synchronization which is error-prone and makes the program less readable.

Another possible optimization is to run the particle filter completely, or at least parts of it, on a GPU. The GPU is made for parallel and independent workloads and seems to fit well for this situation.[36]

Performance is also related to battery consumption which is a critical resource for mobile devices. Depending on the application it might not be affordable to run a particle filter which is computational heavy by design.

List of Figures

List of Tables

List of Algorithms

Glossary

API Application Programming Interface. 35, 37

CPU Central Processing Unit. 34, 50, 54

GNSS Global Navigation Satellite Systems. 3

GPS Global Positioning System. 4

GPU Graphics Processing Unit. 37, 54

IMU Inertial Measurement Unit. 2, 4, 26, 35, 36

IPS Indoor Positioning System. 1, 3, 4, 14, 17, 51

JNI Java Native Interface. 25, 27, 37

MC Monte Carlo. 19

MCMC Markov Chain Monte Carlo. 20

MEMS Microelectromechanical Systems. 7

OS Operating System. 1, 2, 14, 25, 37, 45, 49, 51

SIS Sequential Importance Sampling. 21

SMC Sequential Monte Carlo. 19, 20

Bibliography

[1] Vectornav. *Educational Material, 3.5 Magnetometer Errors & Calibration*. URL: https://www.vectornav.com/resources/magnetometer-errors-calibration (visited on 10/18/2020).

[2] Google. *Android SDK SensorEvent*. URL: https://developer.android.com/reference/android/hardware/SensorEvent (visited on 10/18/2020).

[3] Apple. *iOS Documentation CMMagneticField*. URL: https://developer.apple.com/documentation/coremotion/cmmagnetometerdata/1616084-magneticfield (visited on 10/18/2020).

[4] Wolfgang Demtröder. *Experimentalphysik 2*. Springer, 2012, p. 136. ISBN: 978-3-642-29943-8.

[5] Willie D. Jones. *A Compass in Every Smartphone*. 2010. URL: https://spectrum.ieee.org/semiconductors/devices/a-compass-in-every-smartphone.

[6] Fan-Yu Sung, Shih-Hau Fang, and Ying-Ren Chien. „An experimental study of MEMS-based magnetometers on Android mobile phones“. In: *Digest of Technical Papers - IEEE International Conference on Consumer Electronics* (May 2014), pp. 227–228. DOI: 10.1109/ICCE-TW.2014.6904071.

[7] HyperPhysics. *Illustration of the Hall effect*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/Hall.html (visited on 11/08/2020).

[8] Edwin Hall. „On a New Action of the Magnet on Electric Currents“. In: *American Journal of Mathematics* 2 (1879), pp. 287–292. DOI: 10.2307/2369245.

[9] Wolfgang Demtröder. *Experimentalphysik 2*. Springer, 2012, p. 99. ISBN: 978-3-642-29943-8.

[10] C. C. Finlay et al. „International Geomagnetic Reference Field: the eleventh generation“. In: *Geophysical Journal International* 183.3 (Dec. 2010), pp. 1216–1230. ISSN: 0956-540X. DOI: 10.1111/j.1365-246X.2010.04804.x. eprint: https://academic.oup.com/gji/article-pdf/183/3/1216/1785065/183-3-1216.pdf. URL: https://doi.org/10.1111/j.1365-246X.2010.04804.x.

[11] Ronald T. Merrill, Michael W. McElhinny, and Phillip L. McFadden. *The magnetic field of the earth: paleomagnetism, the core, and the deep mantle*. Academic Press, 1996. ISBN: 978-0-12-491246-5.

[12] Chulliat Arnaud et al. *The US/UK World Magnetic Model for 2015-2020*. URL: https://www.ngdc.noaa.gov/geomag/WMM/data/WMMReports/WMM2015_Report.pdf.

[13] Wikimedia user Chymæra. *Illustration of magnetic field coordinates*. URL: https://commons.wikimedia.org/wiki/File:XYZ-DIS_magnetic_field_coordinates.svg (visited on 10/18/2020).

[14] Martin Walt. *Introduction to Geomagnetically Trapped Radiation*. Cambridge University Press, 1994, pp. 29–33. ISBN: 0-521-61611-5.

[15] Yan-Bin Jia. *Quaternions and Rotations*. 2013. URL: http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf (visited on 10/18/2020).

[16] Wikimedia user Juansempere. *Illustration of euler axis and angle*. URL: https://commons.wikimedia.org/wiki/File:Euler_AxisAngle.png (visited on 10/18/2020).

[17] Andrew Gelman et al. *Bayesian Data Analysis, Third Edition*. Chapman and Hall/CRC, 2013. ISBN: 978-1-4398-4095-5.

[18] Aleksander Sawicki, Zdenek Slanina, and Arturas Linkel. „Compensation of hard- and soft-iron distortions is magnetometer measurement data“. In: (2017). Ed. by Ryszard S. Romaniuk and Maciej Linczuk, pp. 1715–1723. DOI: 10.1117/12.2280794. URL: https://doi.org/10.1117/12.2280794.

[19] Wolfgang Demtröder. *Experimentalphysik 2*. Springer, 2012, p. 111. ISBN: 978-3-642-29943-8.

[20] Wikimedia user Tem5psu. *Illustration of the magnetic hysteresis*. URL: https://commons.wikimedia.org/wiki/File:Magnetic_hysteresis.png (visited on 10/18/2020).

[21] MathWorks. URL: https://www.mathworks.com/help/nav/ref/magcal.html (visited on 11/08/2020).

[22] Ales Kuncar, Martin Sysel, and Tomas Urbanek. „Calibration of low-cost triaxial magnetometer“. In: *MATEC Web of Conferences* 76 (Oct. 2016), p. 05008. DOI: 10.1051/matecconf/20167605008.

[23] M. Kok and T. B. Schön. „Magnetometer Calibration Using Inertial Sensors“. In: *IEEE Sensors Journal* 16.14 (2016), pp. 5679–5689. DOI: 10.1109/JSEN.2016.2569160.

[24] Pengfei Guo et al. „The soft iron and hard iron calibration method using extended kalman filter for attitude and heading reference system“. In: (2008), pp. 1167–1174. DOI: 10.1109/PLANS.2008.4570003.

[25] Demoz Gebre-Egziabher et al. „Calibration of Strapdown Magnetometers in Magnetic Field Domain“. In: *Journal of Aerospace Engineering - J AEROSP ENG* 19 (Apr. 2006). DOI: 10.1061/(ASCE)0893-1321(2006)19:2(87).

[26] José Vasconcelos et al. „Geometric Approach to Strapdown Magnetometer Calibration in Sensor Frame“. In: *Aerospace and Electronic Systems, IEEE Transactions on* 47 (May 2011), pp. 1293–1306. DOI: 10.1109/TAES.2011.5751259.

[27] Y. Wu et al. „Dynamic Magnetometer Calibration and Alignment to Inertial Sensors by Kalman Filtering“. In: *IEEE Transactions on Control Systems Technology* 26.2 (2018), pp. 716–723. DOI: 10.1109/TCST.2017.2670527.

[28] Cao, Shicai Xu, and Xiang Xu. „Real-Time Calibration of Magnetometers Using the RLS/ML Algorithm“. In: *Sensors* 20 (Jan. 2020), p. 535. DOI: 10.3390/s20020535.

[29] The Analytic Sciences Corporation. *Applied Optimal Estimation*. The MIT Press, 1974. ISBN: 978-0-262-57048-0.

[30] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. „Novel approach to nonlinear/non-Gaussian Bayesian state estimation“. In: *IEE Proceedings F - Radar and Signal Processing* 140.2 (1993), pp. 107–113. DOI: 10.1049/ip-f-2.1993.0015.

[31] Pierre Del Moral. „Nonlinear filtering: Interacting particle resolution“. In: *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 325.6 (1997), pp. 653–658. ISSN: 0764-4442. DOI: https://doi.org/10.1016/S0764-4442(97)84778-7. URL: http://www.sciencedirect.com/science/article/pii/S0764444297847787.

[32] Arnaud Doucet and Adam M. Johansen. „A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later“. In: (2011).

[33] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. „On Adaptive Resampling Procedures for Sequential Monte Carlo Methods“. In: (2012), pp. 252–278. DOI: 10.3150/10-bej335. URL: http://hal.inria.fr/docs/00/33/25/83/PDF/RR-6700.pdf.

[34] Hans R. Künsch. „Particle filters“. In: *Seminar für Statistik* (2013). DOI: 10.3150/12-BEJSP07.

[35] Rudolph Emil Kalman. „A New Approach to Linear Filtering and Prediction Problems“. In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[36] Lawrence M. Murray, Anthony Lee, and Pierre E. Jacob. „Parallel Resampling in the Particle Filter“. In: *Journal of Computational and Graphical Statistics* 25.3 (2016), pp. 789–805. DOI: 10.1080/10618600.2015.1062015. URL: https://doi.org/10.1080/10618600.2015.1062015.

[37] Kaijen Hsiao, Henry de Plinval-Salgues, and Jason Miller. „The particle filters and their applications“. In: (2005). URL: http://web.mit.edu/16.412j/www/html/Advanced%20lectures/Slides/Hsaio_plinval_miller_ParticleFiltersPrint.pdf (visited on 09/30/2019).

[38] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. „Estimation of IMU and MARG orientation using a gradient descent algorithm“. In: (2011), pp. 1–7. DOI: 10.1109/ICORR.2011.5975346.

[39] Google. *Android SDK SensorManager*. URL: https://developer.android.com/reference/android/hardware/SensorManager (visited on 10/18/2020).

[40] Andreas Ettlinger, Hans Neuner, and Thomas Burgess. „Development of a Kalman Filter in the Gauss-Helmert Model for Reliability Analysis in Orientation Determination with Smartphone Sensors“. In: *Sensors* 18.2 (Jan. 2018), p. 414. ISSN: 1424-8220. DOI: 10.3390/s18020414. URL: http://dx.doi.org/10.3390/s18020414.

[41] Charles F Jekel. *Digital Image Correlation on Steel Ball*. 2016. Chap. Appendix A, pp. 83–87. URL: https://hdl.handle.net/10019.1/98627.

[42] Matt Godbolt. *Compiler Exploerer*. URL: https://godbolt.org (visited on 11/08/2020).