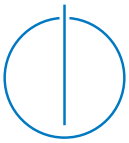
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Pedestrian Augmented Reality Navigator

Anton Moritz Rohr



DEPARTMENT OF INFORMATICS

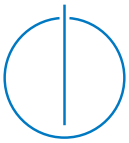
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Pedestrian Augmented Reality Navigator

Fußgänger Augmented Reality Navigator

Author:	Anton Moritz Rohr
Supervisor:	PD Dr. habil. Christian Prehofer
Advisor:	Dipl. -Ing Georgios Pipelidis
Submission Date:	13.12.2018



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 13.12.2018 Anton Moritz Rohr

Abstract

Navigation is often regarded as one of the most interesting use cases for Augmented Reality (AR). Current AR head-mounted displays (HMDs) are rather bulky and cumbersome to use and therefore do not offer a satisfactory user experience for the mass market yet. However, latest generation smartphones offer AR capabilities out of the box, with sometimes even pre installed apps. Apple framework *ARKit* is available on iOS devices, free to use for developers. Android similarly features a counterpart *ARCore*. Both systems work well for small spatially confined applications, but they lack global positional awareness. This is a direct result of one limitation in current mobile technology. Global Navigation Satellite Systems (GNSS) are relatively inaccurate and often cannot work indoors due to the limitation of the signal to penetrate through solid objects, such as walls.

This work presents the Pedestrian Augmented Reality Navigator (PAReNt) iOS App, as a solution to this problem. The App implements a data fusion technique to increase accuracy in global position and showcases AR navigation as one use for the improved data. ARKit provides data about the smartphones motion which is fused with GNSS data and a Bluetooth indoor positioning system via a Kalman Filter (KF). Four different KFs with different underlying models have been implemented and independently evaluated to find the best Filter. The evaluation is measuring the accuracy of the App under controlled circumstances against a ground truth. Two main testing methods are introduced and applied to determine which KF works best. Depending on the evaluation method, this novel approach improves accuracy by 50% or 32% over the raw sensor data.

Contents

List of Abbreviations

AR Augmented Reality

avg Average

BLE Bluetooth Low Energy

CLI Command-line interface

CPU Central processing unit

EKS Extended Kalman Filter

GNSS Global Navigation Satellite System

GPS Global Positioning System

GUI Graphical user interface

HMD Head-mounted display

IMU Inertial measurement unit

IOT Internet Of Things

JSON JavaScript Object Notation

KF Kalman Filter

MSE Mean squared error

MVC Model-view-controller

PAReNt Pedestrian Augmented Reality Navigator

PDA Personal Digital Assistant

PDR Pedestrian dead reckoning

PF Particle Filter

RSSI Received signal strength indicator

SDK Software development kit

SIMD Single instruction multiple data

UKS Unscented Kalman Filter

UUID Universally unique identifier

VR Virtual Reality

WGS 84 World Geodetic System 1984

ZUPT Zero velocity update

1 Introduction

1.1 AR State of the Art

The idea of Augmented Reality (AR) can be summarized in the simplest way as: Augmenting the real-world with computer-generated information. This poses an interesting base problem on all systems that attempt to provide an AR experience. To augment the real-world, it must be perceived at least in some ways.

As technological innovation progresses, Augmented Reality (AR) becomes an increasingly interesting topic. What used to be futuristic field of research is slowly getting to the mass market. HMDs for AR are a heavily researched field, but not yet ready for the average consumer. Smartphones on the other hand are prevalent and current generation devices offer video see-through AR capabilities. Even though a smartphone AR experience feels less natural, it showcases the potential of AR and can be seen as a precursor to HMD AR.

Both Apple and Google have recently published frameworks for their mobile operating systems that enable AR apps. They are called ARKit and ARCore and offer both roughly the same features. Apple ships every sold phone with an app called *Measure*, that is powered by ARKit and can measure the size of real-world objects.

The smartphones perceive the real-world by precisely tracking the pose (position and orientation) in space and detecting elements of the environment with the camera. This enables digital content to be attached to the real world. Tracking the motion of the smartphone is vital, so digital content stays attached to its real-world position. Local positioning works reasonably well for small confined areas but there is currently no capability to connect the local position precisely to a global position. The only option of getting a global position is GPS, which is only accurate to a certain degree (about 5 meters)

1.2 Precise Positioning for AR

This work solves the problem of inaccurate global position by fusing local motion information with global positioning data. The motion information is gathered by ARKit via camera and the inertial measurement unit (IMU). The global positioning data comes from GPS and a custom Bluetooth based positioning system. The later is necessary for indoor scenarios, where GPS does not work. As one application this work presents the

PAReNt iOS app, that showcases pedestrian navigation via AR. Navigation is based on global coordinates and therefore a well suited application.

Not just this example is enabled by the results of this work, but a lot of other AR applications become possible that could improve the quality of life. In principle digital information can be attached to coordinates, and be displayed at a certain defined position. Recommender systems based on location and viewing angle come to mind. The outside wall of a museum could display a preview of whats inside. This is of course of generally related to the field of advertising. If precise positioning is connected with Internet Of Things (IOT) systems, smart decisions can be made by gadgets, like turning of the light when the user leaves the room. One example of evacuation with the help of AR is the MARINS [1] system, which is talked about in section 2.1. Some further opportunities of AR with locational awareness can be found at the end of this work in section 8.2.

1.3 Outline of this work

The remaining work is structured in the following way. Chapter 2 explores related work in the fields of AR navigation, GNSS and Bluetooth based positioning approaches. Chapter 3 explains background knowledge that this work is based on and the details of the data fusion method Kalman Filter (KF). Chapter 4 explains details of how the described problem of precise positioning is approached. Chapter 5 shows some important implementation details. Chapter 6 explains the evaluation that is done to assess the quality of the results. Chapter 7 talks about limitations of this work and how they can be approached by future work. Chapter 8 concludes the work and gives a outlook about the future.

2 Related Work

2.1 Augmented Reality Navigation

The idea of navigation as one of the applications for Augmented Reality systems exists for a while. The highly cited paper with the title "a survey of augmented reality" [2] mentions in 1997 the possible benefits of AR Navigation in Military Aircraft. Information about basic navigation, flight information and possible targets could be superimposed in helmet mounted sights of pilots. Two years later Höllerer et al. [3] present a wearable augmented reality system that is location aware and can annotate the users view with geographical information. The system consist of multiple separate sensors and computing devices that are portable, but rather cumbersome to use (Figure 2.1 left). The goal was to develop a system which displays digital location or object based information indoors and outdoors, as well as giving the user the possibility to enhance and extend this information. One part of the application was a user guidance system that displayed a path in the users view. An example of this way of navigating can be seen on the right in Figure 2.1.

Figure 2.1: Examples from [3]; left: hardware setup; center & right: user guidance

In 2003 Narzt et al. [4] present an AR Navigation system for cars and a modified version for pedestrian navigation. The system consists of a Personal Digital Assistant (PDA) that acts as a video see through display, a camera as a visual input, a GPS sensor, an orientation tracker, a navigational device and a computational unit (laptop). In the car setup the camera is mounted behind the front mirror (Figure 2.2 left) and the display close to the dashboard (Figure 2.2 center). In the pedestrian handheld setup the camera is mounted directly behind the PDA (Figure 2.2 right). The computational unit collects all sensor data, the video stream and path data from the navigational device. After combining those, the visual navigation data in form of semitransparent

lane markings is rendered onto the camera image and presented on the PDA display. In 2006 [5] Narzt et al. publish a slightly modified version, that uses data of additional sensors to improve accuracy.

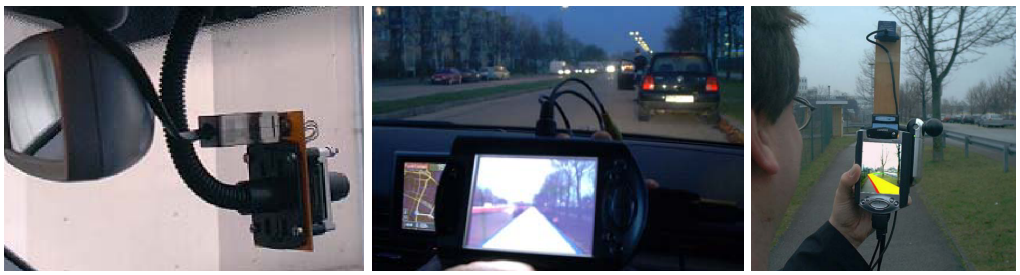


Figure 2.2: Examples from [4]; left: camera car; center: display car; right: mobile setup

More recent projects use the growing capabilities of smartphones and tablets. In 2015 Low and Lee [6] present SunMap+, a prototype available on the Android operating system for indoor AR navigation. The application uses custom created 3D maps of university buildings and facilitates the Vuforia software development kit (SDK). The landmark recognition capabilities of the later is used to determine the users position. Additionally, a pedestrian dead reckoning (PDR) system is used when the Vofuria SDK fails. The PDR system uses the accelerometer to detect step patterns, as well as the magnetometer to get the direction of movement. This system, as all IMU based systems, is prone to accumulating errors, often called drift. The combination of both subsystems compensates their limitations. The application deployed on an android tablet device is using the build-in camera and sensors that can enable the algorithm to determine the users position. This is then used to display augmented reality overlays on the live camera image in a video see through display fashion.

Figure 2.3: Examples from [1]; left: maze setup (illuminated); right: Screenshot of ARKit app with projected arrows

One of the latest AR navigation projects from October 2018 by Diao and Shih [1] is MARINS: A Mobile Smartphone AR System for Pathfinding in a Dark Environment. The paper presents a smartphone based guidance system to aid evacuation in potentially dark and hazardous environments. The researchers constructed an artificial maze (Figure 2.3 left) in an pitch black environment which is only illuminated by the flashlight

on the smartphone. The ARKit based app is able to create a 3D model of the maze on the fly, which is simultaneously used for path planning. The user starts by walking into the maze in a random fashion. As soon as the app detects a dead end, it starts guiding the user with the help of projected arrow markers on the floor (Figure 2.3 right). First back to the last junction and then in the direction of the not yet chosen branch. This is repeated until the end of the maze is reached. The system was evaluated against a control group of users equipped with a map. It was shown that the users with help of the MARINS system where able to find their way out of the maze faster and with less distance traveled than the control group.

2.2 GNSS Improvements

Global positioning and navigation has been a field of research since human traveled the oceans on boats. What used to be a major hassle, is nowadays available to everyone thanks to Global Navigation Satellite Systems (GNSS). In 1978 the Global Positioning System (GPS) was launched as the first GNSS. Since then the usable accuracy for layman seems to improve, mainly because additional systems like Galileo. Even though GNSS are accurate enough for many problems, higher accuracy and precision seems always better than lower. To achieve that one common approach is to integrate IMUs with GNSS receivers. There are two Books, one by Grewal et al. [7] and another one by Groves [8] which discuss this in detail. Since in the past GPS receivers and IMUs have been way bigger then they are today, the main focus on those books lays in the applications for vehicles such as ships, cars and aircraft. Besides other techniques, both books explore the Kalman Filter (KF) [9] as a method to fuse GNSS and IMU data.

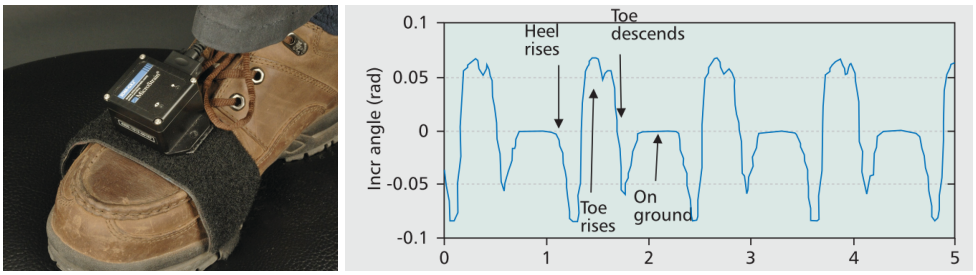


Figure 2.4: Examples from [10]; left: shoe mounted IMU; right: IMU data example

As IMUs and GPS receivers got smaller over time, better localization for pedestrians became more viable. Not just more accuracy, but also continuous tracking even with GPS shortages sparks research with body worn devices. The data of an IMU worn on the body is fundamentally different compared to vehicles. Because of a vehicles inert mass, changes in velocity and direction are never spontaneous, but happen over longer time spans. There are several examples ([11], [10], [12], [13]) of mounting an IMU on the users shoe (see Figure 2.4 left). This has the advantage of zero velocity updates

(ZUPT), when the foot stands still for a moment during the normal walking motion. This can be used to detect distinct steps easier. An example of data from the gyroscope of an IMU can be found in Figure 2.4.

In 2010, Hide et al. [14] present a body worn system (Figure 2.5 left) that uses vision in addition to GPS and a low cost IMU. After successful fusion of all three sensor inputs, their main goal is to build a system that can work even without GPS. They show that the used IMU suffers from dramatic drift. After 60 seconds of GPS outage, the error grew already over 200m. As the vision component they add a camera looking at the ground, similar as a smartphone would have been hold. A FAST feature detector is used in combination with the BaySAC algorithm to match and compute a homography matrix of the camera movement. An example for this optical flow like procedure can be seen in Figure 2.5 (right). The computed information is then integrated with the IMU captured data and results in a drastic improvement. Its shown that the vision and IMU combination results in a approximately 1m error after 60 seconds and less than 3m error after 6minutes. Its concluded that the improves standalone IMU navigation radically, but uses a lot computing power for the computer vision part.

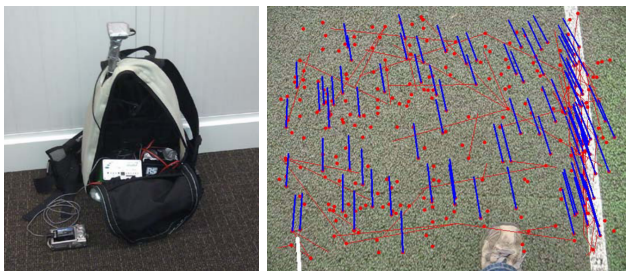


Figure 2.5: Examples from [14]; left: wearable backpack; right: example optical flow

As the technical capabilities in smartphones evolve continuously, it allows more research just on those devices. They usually have GPS receivers, IMUs and remarkable more computing power every generation. Ivanov et al [15] present a paper in 2018 that combines GNSS and IMU in smartphones. Emphasis is put on situations with partial GNSS availability, like entering and leaving a building. The focus lays in comparing different kinds of filters all based on the Kalman Filter (KF) and test them in simulated and real scenarios. They develop and implement four Kalman based methods, which they abbreviate with KS for Kalman Smoother. The first is a classic KS that incorporates position, heading and step count with a fixed step length. The second (KS SL) extends the first with an heuristic to incorporate variable step length. The third and fourth are they non linear Kalman Filter extensions, namely the Extended Kalman Filter (EKS) and Unscented Kalman Filter (UKS). The later two incorporate the step length naturally and don't need an additional heuristic.

The results of their evaluation are very interesting. In Figure 2.6 the result from their simulations are displayed. With full GPS coverage (left) none of the Filters stands out

$\sigma_z, \sigma_\theta, \sigma_l$	KS	KS SL	EKS	UKS
$10, 1°, 0.1$	2.45	3.07	2.16	2.24
$15, 2°, 0.2$	3.37	5.66	4.73	3.21
$20, 3°, 0.3$	4.08	3.75	2.87	3.64
$25, 4°, 0.4$	5.56	3.85	4.4	3.82
$30, 5°, 0.5$	6.02	5.43	6.56	5.59
average error	4.30	4.35	4.14	3.70

$\sigma_z, \sigma_\theta, \sigma_l$	KS	KS SL	EKS	UKS
$10, 1°, 0.1$	11.4	5.94	3.21	3.65
$15, 2°, 0.2$	11.13	6.39	3.12	3.53
$20, 3°, 0.3$	16.32	8.14	4.45	5.83
$25, 4°, 0.4$	32.65	18.94	19.95	20.68
$30, 5°, 0.5$	39.09	11.18	12.54	16.31
average error	22.11	10.11	8.65	10

Figure 2.6: Results from simulated runs in [15]; left: full GPS coverage; right: partial GPS

with significantly better results then the other. It could be argued that the UKS has a slight advantage. With only partial GPS coverage it becomes clear that the inclusion of a variable step length makes a different, because the error of KS is about double then the others. Which is also what is concluded by the authors. But even when used variable step length, none of the remaining filters sticks out significantly. If a winner needs to be elected, its the EKS this time.

	KS	KS SL	EKS	UKS
track 1	36.9608	14.1578	21.2462	28.3694
track 2	27.8456	12.6445	25.2197	32.1426
track 3	35.1727	11.4269	22.2905	31.3898
track 4	21.0482	17.6744	12.2995	51.0284
average error	30.2568	13.9759	20.2640	35.7325

Figure 2.7: Results of real data with partial GPS coverage from [15]

The error values of the real world example with partial GPS coverage can be found in Figure 2.7. The walk was conducted by starting indoors, leaving the building, walking outdoors, turning around and reentering the building. The generally high error levels are regarded by the authors to high GPS errors in the transition phases from outdoor to indoor. In this table, the Kalman Filter with the adjusted step length (KS SL) performs significantly better than all other filters.

2.3 Indoor Positioning via Bluetooth

Using wireless networks to determine indoor position has been researched a lot. In 2000 Bahl and Padmanabhan [16] developed RADAR one of the earliest systems, based on WaveLan a predecessor of WiFi. Boukerche at al. [17] summarize in 2007 existing approaches in positioning via wireless sensor networks. The focus lays on concepts independent of used wireless technology. On of the concepts is trilateration, a schematic example can be found in Figure 2.8. In 2011 Subhan at al. [18] looks at Bluetooth

based indoor location approaches. Since no standard for any distance based measures exists in Bluetooth, they review several properties of Bluetooth signals. In the end they decide on an received signal strength indicator (RSSI) trilateration approach. To convert RSSI to distance in meters they test their Bluetooth devices. Several measurements in different distances are made and combined with the radio propagation model. With the results of the trilateration they deploy a gradient filter and are able to reduce their average error from 5.87m to 2.67m. Since this approach takes often average values over certain periods of time, it is not very suited for moving targets.

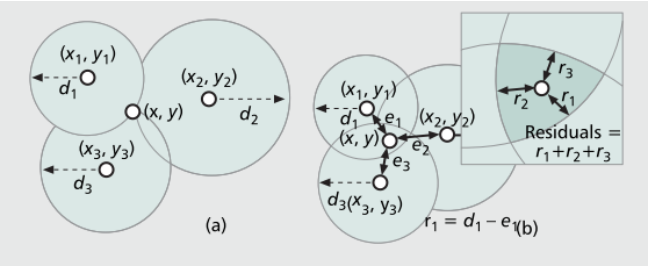


Figure 2.8: Schematic example of trilateration in [17]

Over time and with new Bluetooth standards more systems were developed. Röbesaat et al. [19] present a system that combines Bluetooth Low Energy (BLE) and dead reckoning for positioning via smart phone in an office scenario. Low cost BLE Beacons are used and as a first step their RSSI behavior is examined. It is noticed that quite large variances in RSSI exist and filtering is needed, to counter that a KF is used. the filtered values are then converted to distance in meters using the path loss model. The resulting conversion diagram can be found in Figure 2.9 on the left. For positioning the signals of three or more beacons are fed into a multilateration model that uses the least square method to minimize residuals. In the end another average filter is added to the result of the lateration.

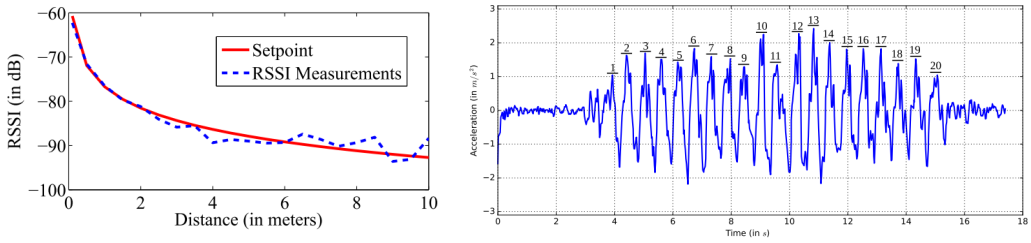


Figure 2.9: Examples from [19], left: conversion RSSI to distance; right: step counting via accelerometer

The PDR is developed based on an accelerometer step detection, that registers peaks in acceleration with a fixed step length of 74cm. An example of 20 detected steps can be found in Figure 2.9 on the right. The magnetometer is used to detect direction of

walking of the user. Several test measurements show an average magnetometer error of less than 6 degrees.

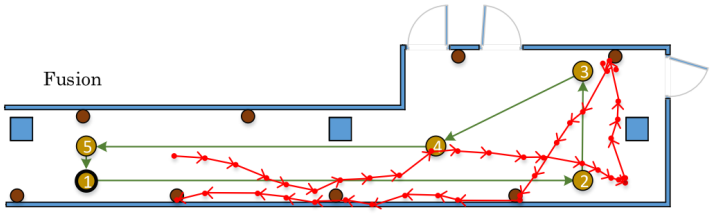


Figure 2.10: Fusion result from [19]; green: path; red: fusion result

As a last step the results from the trilateration are combined with the results of the dead reckoning with a KF. The researchers evaluate the developed system in an office corridor. 8 Beacons are deployed and the path has a length of about $30m$. An example of the fusion result marked on an image of the test setup can be found in Figure 2.10. As an error measure the distance to the end point is taken and a average error of $74cm$ is reached. As another addition context information in form of constraints is added, but cant add significant accuracy.

3 Background

3.1 ARKit

ARKit [20] is a framework developed and maintained by Apple on the iOS platform, free to use for developers. It is supported by devices running iOS 11 and onwards and requires the devices to have an Apple A9 central processing unit (CPU) or newer. This means iPhones from 6S, 6S Plus, SE and more recent are capable, as well as all iPads from 2017 (sometimes called 5th generation) and newer. Even though it is free to use, the internals of how ARKit works exactly are proprietary and not publicly available.

Apple is however providing documentation for the several features of ARKit, which are all either connected to the front or back camera. Front camera features are things like face tracking, pose and expression detection and more. This work only uses the back camera for which ARKit provides features like tracking of the environment, a coordinate system to register objects and horizontal surface detection and more.

The world tracking capabilities are powered by *visual-inertial odometry* [21], which combines information from the IMU with information gathered by analyzing the camera video stream. The later is done by extracting features from each video frame and tracking their positions across frames. All together it results in the capability to compute how the camera and therefore the whole smartphone moves in space across time.

This means, as long as ARKit runs, the position (in meters) and orientation (as a vector) of the device is available, but only in comparison to the start position and orientation. Its realized in the following way, as soon as ARKit starts, it puts the origin of a coordinate system at the position in space where the camera is located at that moment. Because its in no way connected to a global coordinate system, throughout this work it will be referred to as a local coordinate system. The developer can decide on how the local coordinate system is orientated. There are 3 options.

Camera The coordinate system is oriented as the camera, with the x-Axis pointing to the right, y-Axis pointing up and z-Axis points out the front of the device, towards the user

Gravity The coordinate system the same as in the *Camera* option, with the exception that the y-Axis is not pointing up like the smartphone, but up as the opposite of the direction of gravity (Figure 3.1 left). This results in the x-z-Plane being parallel to the worlds surface.

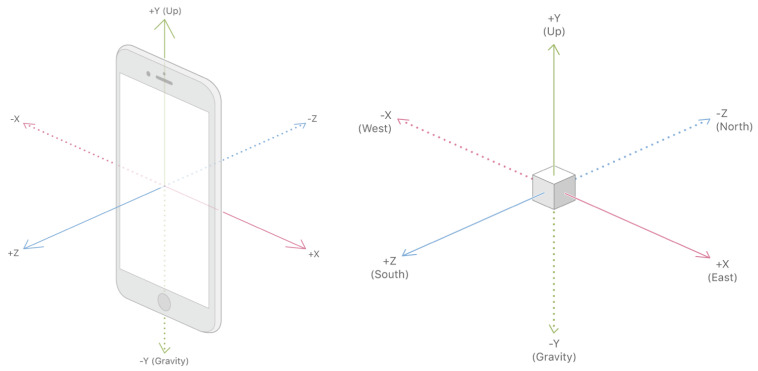


Figure 3.1: ARKit coordinate system orientation; left: *Gravity*; right: *GravityAndHeading*

GravityAndHeading The coordinate systems y-Axis is the same as in the *Gravity* option, but the x-Axis points in direction East and the z-Axis points in direction North (Figure 3.1 right).

The coordinate system gets set the moment ARKit is started. For the configurations *Gravity* and *GravityAndHeading* this involves values of the accelerometer and magnetometer sensors, which might be inaccurate. Its important to note that the directions of the Axises are subject to error. The position and orientation of the coordinate system is not changed automatically during an ARKit session, but can be changed by the app developer.

Note: When in this work the term *local reference frame* is used, it is meant that the coordinates in the ARKit system are only valid in the local coordinate system and are not connected to the global position of the user (*global reference frame*).

3.2 Bluetooth Beacons

Bluetooth Beacons are small devices that broadcast an identifier using the Bluetooth protocol. The broadcasting is implemented via Bluetooth low energy proximity sensing and is completely independent from receiving devices, it does not need any connecting mechanism. The identifier of a beacon is programmable and consists of a universally unique identifier (UUID), a major and a minor value. The later two being 16 bit each. The typical applications for beacons proximity based, for example tracking of objects that have a beacon attached to them. With a stationary beacon a smartphone can detect if the user is in the proximity of a beacon. The distance of a beacon to the Bluetooth signal receiver is usually approximated by using the RSSI in combination with a path loss model.

3.3 Trilateration

Trilateration is a geometrical term to describe the process of determining a location based on distances to three known locations. In two dimensions, any location is always unambiguously defined byt those three distances. To calculate that distance the geometry of circles is used. A circle around a location describes all points that are exactly the radius distance away from that location.

In two dimensions a circle with radius r around the origin can be mathematically described with $r^2 = x^2 + y^2$. When talking about multiple circles with different radii and centers different than the origin, the equation changes to Equation 3.1.

$$r_i^2 = (x - x_i)^2 + (y - y_i)^2 \tag{3.1}$$

The circles are distinguished by index i and one distinct circle is therefore defined by three parameters r_i, x_i and y_i.

When the distances to the unknown location x, y are precisely known it looks like the left example in Figure 2.8 on page 8. And there is only one solution that can be computed analytically. When the distances are not known precisely the problem can be visualized like the right example of that Figure 2.8. In the real world, measurements are usually somewhat imprecise and prone to noise up to a certain level, which makes a solution to the second image necessary.

Robesaat et al. [19] for example tackle that problem in the following way. The difference between the real radius r_i and the measured radius m_i can be set as $v_i = r_i - m_i$. The location x, y is the result that assumes a minimal squared measuring error, mathematically expressed in Equation 3.2

$$(x, y) = \min(\sum^{i}(v_i)^2) \tag{3.2}$$

This equation cannot be solved analytically anymore, thus requires a numerical solver. This problem can also contain local minima, which needs to be taken into consideration.

It is possible to use this also with more than three circles. Solving this problem is then often called *Multilateration*.

3.4 Global Navigation Satellite Systems

Global Navigation Satellite Systems (GNSS) are satellite systems that send radio signals down to earth, which can be used for location and navigation purposes. the signal receiving devices use the information transmitted from multiple satellites to calculate the current global location of the device. The most prominent and first GNSS is the Global Positioning System (GPS) owned by the United States government. Besides GPS there are also the Russian GLONASS, Chinese BeiDou and European Galileo system. BeiDou and Galileo do not have full global coverage yet, but are expected to be

completed by 2020. There are vast number of receiver devices, like smartphones and automotive navigation systems.

The location calculation that needs to be done in the receiving devices is similar to the Trilateration problem but in three dimensions.

$$d_i = (t - b - s_i)c \quad d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} \tag{3.3}$$

The distances to the satellites are calculated with the help of the speed of light c. The distance is basically c multiplied with the time it took for the signal to arrive. Measuring the exact time is impossible when the satellites clock s_i and the receivers clock t are not perfectly synchronized. Therefor the term b is added to express this bias. The satellites clocks are assumed to be synchronized to each other. Since there are four unknown variables (x, y, z, b) a signal from at least 4 satellites is necessary to solve the equations.

There are different levels of accuracy for the different satellite systems and different methods of computing the location, but in general an accuracy of about 5 meters is presumed. The used Apple iPhone X supports GPS, GLONASS and Galileo. The received data is provided via a framework called *CoreLocation*

Note: In the following work the abbreviation GPS and GNSS are used interchangeably.

3.5 Kalman Filter

The Kalman Filter, named after Rudolf Emil Kálmán was published in the 1960 [9], since then it became a well known algorithm in multiple fields, including navigation and data fusion. The filter takes in measurements over time, as well as statistical noise or other inaccuracies and produces estimates of unknown underlying variables. The estimates are generally getting better over time. This is related to the fact that early estimates have higher uncertainty because little information was exposed yet.

The algorithm consists internally of two steps, prediction and update. The prediction step produces estimates and uncertainties based on the current state. The update step updates the state by taking the results from the prediction step, as well as a measurement and uncertainty information about that measurement into account. How much emphasis is put on the prediction or the update is determined by the Kalman gain, which is between zero and one for each state variable. A low Kalman gain results in a smoother estimation, but also less responsiveness to changes. The algorithm described in the following sections uses an optimal Kalman gain functionality, that adapts the gain on the fly, based on the current uncertainty values as well as the noise of prediction and measurement.

In KFs the state is a vector of n underlying variables that need to be estimated. The error or uncertainty information is encoded in a $n \times n$ covariance matrix.

3.5.1 Prediction

The prediction step is performed by two equations. One to compute a the prediction of the state, the second to change the error covariance.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{3.4}$$

Equation (3.4) shows how the state estimate is predicted. $\hat{\mathbf{x}}_{i|j}$ generally (for all i, j) represents the estimate of $\mathbf{x}$ at time i given all previous observations up until and including j. That means $\hat{\mathbf{x}}_{k-1|k-1}$ is the state estimate at time $k-1$ given all observations at times before and including $k-1$, or in simpler words, the state estimate before the prediction. $\mathbf{F}_k$ is the state transition model (at time k). $\mathbf{B}_k \mathbf{u}_k$ is added to the product of estimate before prediction and start transition model. $\mathbf{B}_k \mathbf{u}_k$ is the control part that consists of the control model ($\mathbf{B}_k$) and the control vector ($\mathbf{u}_k$). The result is $\hat{\mathbf{x}}_{k|k-1}$, the state estimate at time k using all the observations until $k-1$, or in other words: the newly predicted state.

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k \tag{3.5}$$

The second Equation 3.5 is about changing the error covariance matrix according in line with the prediction of the state estimate that was just explained. $\mathbf{P}_{i|j}$ (for all i, j) is analogous to the last section, the error covariance matrix at time i given all previous observations put until and including j. Therefore $\mathbf{P}_{k-1|k-1}$ is the error at $k-1$ with all information until that time $k-1$, or in other words: the error covariance matrix before the prediction. This covariance is multiplied with the state transition model ($\mathbf{F}_k$) from the left and the transposed state transition model from the right. To this product $\mathbf{Q}_k$ is added, which is the covariance of the process noise. All together this results in $\mathbf{P}_{k|k-1}$, which is the error covariance matrix at time k given all observations up until $k-1$, or in other words, the error covariance after the prediction step.

Because these estimations are predicted before the actual measurement, they are often called *a priori* estimates

3.5.2 Update

The update step is broken down into 5 different equations that result again in an updated state estimation and covariance matrix.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{3.6}$$

The first equation (equation (3.6)) contains $\mathbf{z}_k$, the measurement at time k. $\mathbf{H}_k$ the observation model, which transforms a state into the observation space. And $\hat{\mathbf{x}}_{k|k-1}$ the state estimate from the prediction step. The observation model is necessary because the internal state might have different dimensions than the observed measurement. As earlier, the state is denoted in n dimensions and from now on the measurement

dimension is denoted with m. Accordingly the observation model has $m \times n$ dimensions. First $\hat{\mathbf{x}}_{k|k-1}$ is transformed into the observation space with $\mathbf{H}_k$, which could be described as a prediction of the measurement. Then the difference between the real measurement and the predicted measurement is the result $\tilde{\mathbf{y}}_k$. This is often called the Innovation or measurement pre-fit residual and basically just means how far off is the prediction compared to the measurement.

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k \tag{3.7}$$

The next Equation 3.7 takes the predicted error covariance matrix $\mathbf{P}_{k|k-1}$ (see Equation 3.5) and transforms with the observation model ($\mathbf{H}_k$) from the left and the transposed observation model ($\mathbf{H}_k^{\mathrm{T}}$) from the right. In addition with the measurement noise $\mathbf{R}_k$ this result in $\mathbf{S}_k$, which can be described as the error covariance in observation space, that includes the uncertainty of the measurement. Since its the covariance in observation space, it has the observation dimensions $m \times m$.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^{\mathrm{T}} \mathbf{S}_k^{-1} \tag{3.8}$$

Equation 3.8 computes the optimal Kalman gain, by comparing two error covariance matrices. $\mathbf{P}_{k|k-1}$ contains the error before the prediction plus the prediction noise $\mathbf{Q}_k$. $\mathbf{S}_k$ is the same, plus the measurement noise $\mathbf{R}_k$. Combining them in the way Equation 3.8 does with the inverse of $\mathbf{S}_k$, results in a Kalman gain that is closer to 1, the closer $\mathbf{S}_k$ is to $\mathbf{P}_{k|k-1}$. In other words, if $\mathbf{R}_k$ is very small (a high confident measurement), then $\mathbf{K}_k$ is closer to 1. And vice versa if $\mathbf{R}_k$ is large, $\mathbf{K}_k$ is closer to 0. Since the Kalman gain describes the emphasis between prediction and update, a low gain results in higher belief in the prediction and lower belief in the measurement, which might be noisy.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{3.9}$$

Equation 3.9 uses the gain to update the state estimate. $\tilde{\mathbf{y}}_k$ describes the difference between prediction and measurement in observation space. Since $\mathbf{K}_k$ has the dimensions $n \times m$, (the opposite of the observation model $\mathbf{H}_k$), it can take a value from observation space into state space. All together Equation 3.9 takes the predicted state estimate $\hat{\mathbf{x}}_{k|k-1}$ and adds a Kalman gain sized portion of $\tilde{\mathbf{y}}_k$. Here can be seen how a small gain results in an estimate that is close to the prediction and a high gain result in an estimate close to the measurement.

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{3.10}$$

Equation 3.10 updates the error covariance based on information from the Kalman gain. If the gain is close to zero, $\mathbf{I} - \mathbf{K}_k \mathbf{H}_k$ is close to the identity ($\mathbf{I}$), resulting in an error covariance close to the prediction covariance error. A high Kalman gain close to one can only occur when the measurement uncertainty is very low (Equation 3.8), this then results in a lower error covariance than the predicted error covariance.

Since $\hat{\mathbf{x}}_{k|k}$ and $\mathbf{P}_{k|k}$ describing the state and error estimate after an update, they are ofter called *a posteriori* estimates.

4 Approach

This chapter gives an overview on how the proposed problem of improved positioning data for AR navigation is approached. The decision of using iOS over Android as a platform was made mainly because of the available personal resources (iPhone X). Even though the application is iOS specific, generality is kept in mind. This is approached with keeping functionality separate from platform specific code, with custom type as symbolized in figure 4.1. Furthermore an effort was made to decouple the modules as much as possible.

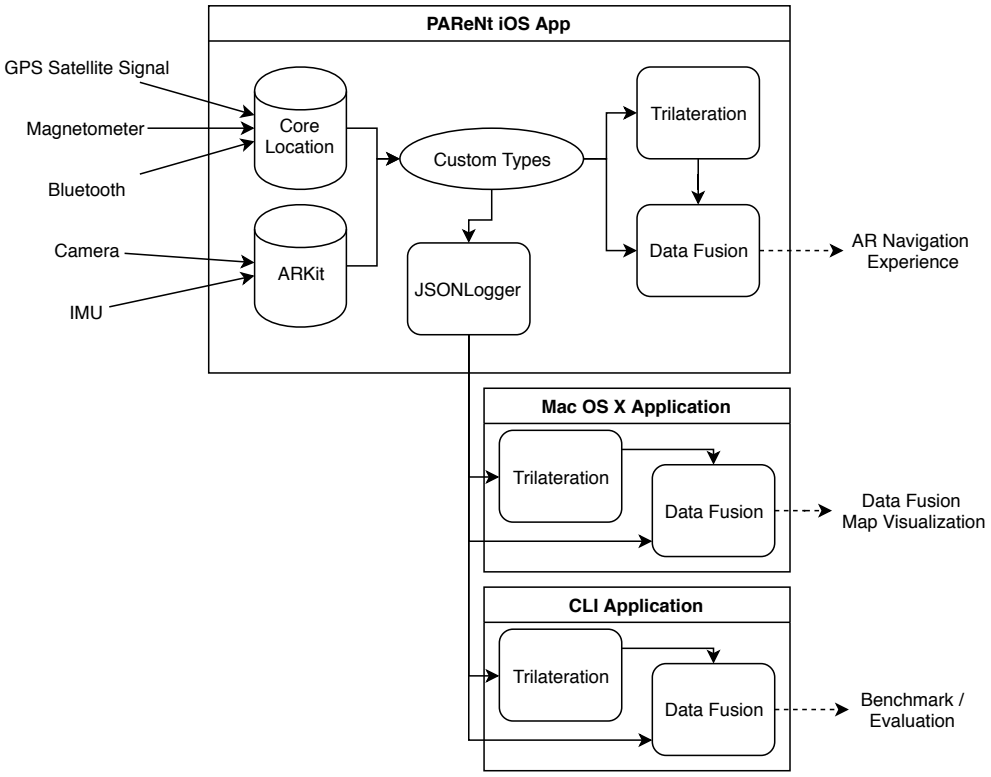


Figure 4.1: Overview Approach

There are three components that share some functionalities, but serve different purposes. The PAReNt iOS App is the main component and includes all functionality that is needed for the AR navigation experience. It captures sensor data, converts it to custom types, saves them via JSON and preforms the trilateration and data fusion.

The saving to JSON enables the two other components. The saved data is used by the Mac OS X and CLI Applications to, respectively, visualize and evaluate the data fusion. More specific information about the implementation can be found in chapter 5.

Several decisions are made because of the mobile nature and the accompanied limited computational resources. One popular approach besides Kalman Filter in data fusion is the Particle Filter (PF), which often results in better estimates. The downside of a PF is its high computational cost, which is why using KFs is chosen instead.

4.1 Bluetooth Beacon Trilateration

To approach the problem of positioning in GPS deprived environments, like indoors, a Bluetooth Low Energy Beacon Trilateration system is developed. Cheap battery powered short range beacons sold by the company Avvel [22] are used. Being battery powered and inexpensive, makes them easily deployable and cost effective even for large buildings. To convert from RSSI to distances, a simplified model is used, that approximates the signal propagation model with a second order polynomial. The details and parameters are stated in subsection 5.2.5.

The general functional principle of trilateration is explained in section 3.3, with one possibility how to solve the problem of imprecise distant measurements. This work uses a different approach to avoid a computation heavy numerical solver. This is done by utilizing properties of intersecting circles, the details are explained in subsection 5.2.9.

4.2 Data Fusion

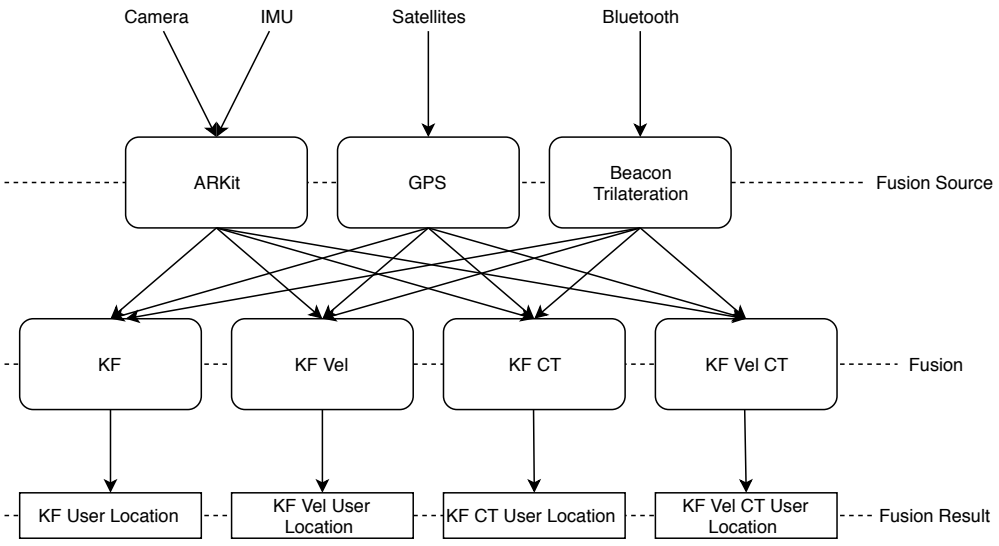


Figure 4.2: Overview data fusion

A general overview of how the different sensor data is fused together can be found in figure 4.2. There raw data comes in from the camera, the IMU, satellites and via Bluetooth. This data is then preprocessed in ARKit, CoreLocation and the Beacon Trilateration module. ARKit and CoreLocation are frameworks developed by Apple, the Trilateration module is a custom implementation with details in figure 5.8. Those three components categorized as fusion source, because the data they provide is used by the Kalman filters in the fusion step. The four independent filters all take in fusion source data and produce all their own result. How each KF works and how they are different is explained in the following sections 4.2.1 to 4.2.4.

4.2.1 Kalman Filter 1: KF

The first filter is the simplest one of the four. This Filter is in the following work often simply referred to as *KF*. The internal state consists only of two variables one for latitude and one for longitude. In the prediction phase, its simply assumed that the state stays the same and there is no control. The covariance of the process noise is determined by one parameter n which is set in meters. n_{lat} and n_{lng} compose the parameter n, converted in latitude longitude space. The conversion between latitude, longitude and meters is different depending on where the user is located on the world. Equation (4.1) summarizes all definitions for the prediction phase in this KF. The index k is not necessary for these definitions, since they are the same in each step.

$$\hat{\mathbf{x}} = \begin{bmatrix} lat \\ lng \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{B} = 0_{2,2} \quad \mathbf{u} = 0_{2,1} \quad \mathbf{Q} = \begin{bmatrix} n_{lat} & 0 \\ 0 & n_{lng} \end{bmatrix} \tag{4.1}$$

For the update step, it is separated between ARKit sensor data and GPS Sensor data. When there is new ARKit sensor data available, the measurement ($\mathbf{z}_k$) is composed of $\Delta ar_{k,lat}$ and $\Delta ar_{k,lng}$. The data is only available in a local reference frame (see section 3.1). Therefore the displacement from the last $\Delta ar_{lat,k-1}$ in meters is taken and converted back into latitude space to compute a new latitude value. The same is done for respectively for the longitude values. By doing this the observation model ($\mathbf{H}$) is very simple, just the identity. Similar to the prediction the measurement noise ($\mathbf{R}_k$) is determined by one parameter mA in meters, which is then converted into latitude longitude space. Equation (4.2) summarizes the definitions for the update phase with ARKit data.

$$\mathbf{z}_k = \begin{bmatrix} \Delta ar_{k,lat} \\ \Delta ar_{k,lng} \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} mA_{lat} & 0 \\ 0 & mA_{lng} \end{bmatrix} \tag{4.2}$$

For new GPS data the modeling is quite similar, but no Δ values need to be taken for the measurement. So $gps_{k,lat}$ just describes the latitude GPS sensor data at time k. The measurement noise is also similar by being defined as a meter value mG, but it can change depending on the data at time k. *CoreLocation* provides information

about accuracy for each measurement, depending on GPS signal strength and more. Equation (4.3) summarizes the definitions required for the GPS update step.

$$\mathbf{z}_k = \begin{bmatrix} gps_{k,lat} \\ gps_{k,lng} \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{R}_k = \begin{bmatrix} mG_{k,lat} & 0 \\ 0 & mG_{k,lng} \end{bmatrix} \tag{4.3}$$

4.2.2 Kalman Filter 2: KF Vel

The second filter (*KF Vel*) has a similar approach as the first, but uses a more complex model that included velocities in latitude and longitude direction. The introduction of velocity makes it important to keep track of the time, in order to properly compute the velocity function $\Delta s_k = \Delta t_k v_{k-1}$ with Δ meaning for any u: $\Delta u_k = u_k - u_{k-1}$. The function describes the distance in between two steps (Δs_k), as the time between the steps (Δt_k) times the last velocity (v_{k-1}). The state ($\hat{\mathbf{x}}$) is now four dimensional, because of the additional velocity variables. The state transition model ($\mathbf{F}_k$) implements this velocity function in latitude and longitude direction and is now different in each step k because of Δt_k. As in the simple KF filter, no control method (B, u) is used.

$$\hat{\mathbf{x}} = \begin{bmatrix} lat \\ lng \\ vLat \\ vLng \end{bmatrix} \quad \mathbf{F}_k = \begin{bmatrix} 1 & 0 & \Delta t_k & 0 \\ 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = 0_{4,4} \quad \mathbf{u} = 0_{4,1} \tag{4.4}$$

The process noise is assumed to be happening because of a constant acceleration $\mathbf{a}$ in latitude and longitude direction, based on Newtons laws of motion $\Delta s_k = \frac{1}{2} a \Delta t^2 + v_{k-1}$ and $v_k = a\Delta t + v_{k-1}$. So in matrix form all together this becomes $\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{G}_{k,lat}\mathbf{a}_{lat} + \mathbf{G}_{k,lng}\mathbf{a}_{lng}$, with $\mathbf{G}_{k,lat}$ and $\mathbf{G}_{k,lng}$ defined as:

$$\mathbf{G}_{k,lat} = \begin{bmatrix} \frac{1}{2}\Delta t_k^2 \\ 0 \\ \Delta t_k \\ 0 \end{bmatrix} \quad \mathbf{G}_{k,lng} = \begin{bmatrix} 0 \\ \frac{1}{2}\Delta t_k^2 \\ 0 \\ \Delta t_k \end{bmatrix} \tag{4.5}$$

When assuming a standard deviation σ_a for $\mathbf{a}$ and $\mathbf{G}_{k,lat}\mathbf{a}_{lat} + \mathbf{G}_{k,lng}\mathbf{a}_{lng} \sim N(0, \mathbf{Q}_k)$, this results in the following process noise covariance matrix $\mathbf{Q}_k$

$$\mathbf{Q}_k = \mathbf{G}_{k,lat}\mathbf{G}_{k,lat}^T\sigma_{lat}^2 + \mathbf{G}_{k,lng}\mathbf{G}_{k,lat}^T\sigma_{lng}^2 = \begin{bmatrix} \frac{1}{4}\Delta t_k^4\sigma_{lat}^2 & 0 & \frac{1}{2}\Delta t_k^3\sigma_{lat}^2 & 0 \\ 0 & \frac{1}{4}\Delta t_k^4\sigma_{lng}^2 & 0 & \frac{1}{2}\Delta t_k^3\sigma_{lng}^2 \\ \frac{1}{2}\Delta t_k^3\sigma_{lat}^2 & 0 & \Delta t_k^2\sigma_{lat}^2 & 0 \\ 0 & \frac{1}{2}\Delta t_k^3\sigma_{lng}^2 & 0 & \Delta t_k^2\sigma_{lng}^2 \end{bmatrix} \tag{4.6}$$

The update step for this filter is very similarly to the last filter, but since the state is including velocity, the observation model is a 2×4 matrix. For ARKit data and GPS

data, $\mathbf{z}_k$ and $\mathbf{R}_k$ are constructed the same way as for the last filter (see subsection 4.2.1). The definitions are summarized in Equation 4.7 and Equation 4.8.

$$\mathbf{z}_k = \begin{bmatrix} \Delta ar_{k,lat} \\ \Delta ar_{k,lng} \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{R}_k = \begin{bmatrix} mA_{lat} & 0 \\ 0 & mA_{lng} \end{bmatrix} \tag{4.7}$$

$$\mathbf{z}_k = \begin{bmatrix} gps_{k,lat} \\ gps_{k,lng} \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} mG_{k,lat} & 0 \\ 0 & mG_{k,lng} \end{bmatrix} \tag{4.8}$$

4.2.3 Kalman Filter 3: KF CT

The main difference between the first two filters and this one, is that they use the ARKit data as control input, instead of measurement input. This filter KF CT is designed similarly to KF, with a 2 dimensional state consisting just of latitude and longitude values. The control vector Δar_k, which is the displacement (difference) from the last ARKit value in meter ($\Delta ar_k = ar_k - ark_{k-1}$). This needs to be converted into latitude longitude space. The control model is a simple identity matrix since its only connecting the latitude and longitude values. Equation (4.9) summarizes the definitions for the KF CT prediction step.

$$\hat{\mathbf{x}} = \begin{bmatrix} lat \\ lng \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{B}_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{u}_k = \begin{bmatrix} \Delta ar_{k,lat} \\ \Delta ar_{k,lng} \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} n_{lat} & 0 \\ 0 & n_{lng} \end{bmatrix} \tag{4.9}$$

The update step only consist of one GPS part, because the ARKit data is already used in the prediction step. This results in exactly the same definitions as for the KF GPS update step, which can be found in Equation 4.3.

4.2.4 Kalman Filter 4: KF Vel CT

The KF Vel CT filter a mixture between KF Vel and KF CT, by using a 4 dimensional state similar to KF Vel, but using the ARKit data as control vector like KF CT. The state $\hat{\mathbf{x}}$, the state transition model $\mathbf{F}_k$ and the error covariance of the process noise ($\mathbf{Q}_k$) is defined exactly as in KF Vel, which can be found in Equation 4.4. The difference is found in the control model and the control vector. The AR data is used in the control vector to adjust a change in velocity in between the last step ($k-1$) and the current (k). The velocity during the step k is computed by $\Delta var_k = \frac{\Delta ar_k}{\Delta t_k}$, where Δar_k is again the meter difference detected by ARKit between the last step and the current. $\dot{\Delta} var_k$ shows how much the velocity changed, by computing $\dot{\Delta} var_k = \Delta var_k - \Delta var_{k-1}$. The control model propagates that changed velocity into the state estimation velocity and adjusts the position as well. Equation (4.10) shows the definitions of the control model and

control vector.

$$\mathbf{B}_k = \begin{bmatrix} 0 & 0 & \Delta t_k & 0 \\ 0 & 0 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ \dot{\Delta} var_{k,lat} \\ \dot{\Delta} var_{k,lng} \end{bmatrix} \tag{4.10}$$

The update step is similar to KF CT only considering GPS data, because the ARKit data is already handled in the update step. The definitions are identical to the KF Vel definitions for the GPS update, that can be found in Equation 4.8.

4.3 Single Instruction Multiple Data

The Kalman filter is based mainly on matrix operations like multiplications and inversions, and all four filters use only up to 4×4 dimensions. This work uses an approach to speed up the computations by computing matrix operations in parallel. This is done by using single instruction multiple data (SIMD) types. SIMD types perform one instruction like multiplication on a vector or matrix of data in parallel. This needs specialized hardware that is capable, which in this case is the ARM NEON architecture extension built into the iPhone CPU. This is not just used in the Kalman filters, but also in multiple geometry functions used by the Trilateration.

4.4 Augmented Reality Navigation

The navigation part in this work is approached by displaying path markers on the ground. The route information in form of global latitude longitude coordinates is obtained with the help of Apple's MapKit Framework. The user can search for a specific address or place of interest and then gets a list of possible options from the MapKit database. After choosing a destination MapKit provides a pedestrian route which is converted into a path. The path is displayed in the users Augmented Reality View, based on the users current location and orientation, which are itself based on the results of the Kalman filters. The path is visually *pinned* to the ground by setting the z-Axis coordinate (height) to same value as the ground. This information about the ground is obtained by using ARKits horizontal plane detection feature.

5 Implementation

5.1 Overview

This chapter is giving some insight on the actual implementation, a short architecture description in this section is followed by details of the main components.

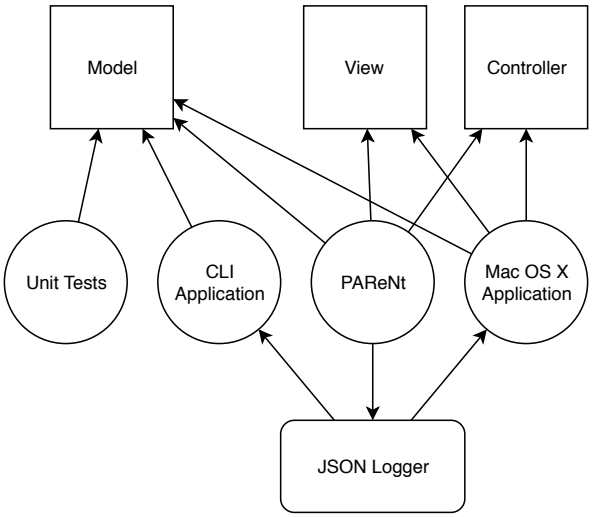


Figure 5.1: Overview Implementation

The whole project consists of three applications that share code, which are going to be shortly described in the following sections 5.1.1 to 5.1.3. In addition there is a unit test component (section 5.6) that can be run independently from all three applications. The code is organized in a model-view-controller (MVC) fashion. Figure 5.1 shows Model, View and Controller and four executables in circles under it. It can be seen that View and Controller are not used by the Tests and the CLI Application which makes sense because both don't need a graphical user interface or user input. The design pattern MVC was chosen mainly to decouple functionalities and therefore keep the codebase easily extendable and reusable. The connection between the three modules is realized in Apple's implementation of the observer pattern, named *NotificationCenter*. As one short example; when the Model computes something new, it notifies all observers. The View is setup as a Model observer and therefore is able to render the newly computed information. Details of Model, View and Controller are described in sections 5.2 to 5.4. The last component on figure 5.1 is the JSONLogger, which is responsible for sharing

data between the three applications. More on how that works can be found in the following sections 5.1.1 to 5.1.3 and 5.5.

5.1.1 iOS Application PAReNt

The main iOS app is called PAReNt and the only application of the three that is exposed to actual users. This app uses all almost all parts of the later described code, besides some specific views of the Mac Application and some evaluation methods of the CLI Application.

The app has two versions, one debugging version that is split in two parts. The top part is the actual AR View, explained in section 5.3.2. The lower part shows the debugging information in the Map View, as explained in section 5.3.1. In between the two parts are some controlling elements to enter a destination and for starting and stopping an evaluation run (some more details in section 5.4.3). This version also collects all sensor data via the JavaScript Object Notation (JSON)Logger, explained in section 5.5. Besides the debugging version, there is also a production version, which only consist of the AR View and the destination search bar and does not collect any sensor data. Example screenshot of the app can be found in figure 5.2. One the left and center the production version can be seen, on the right is an example for the debugging version that includes the map view.

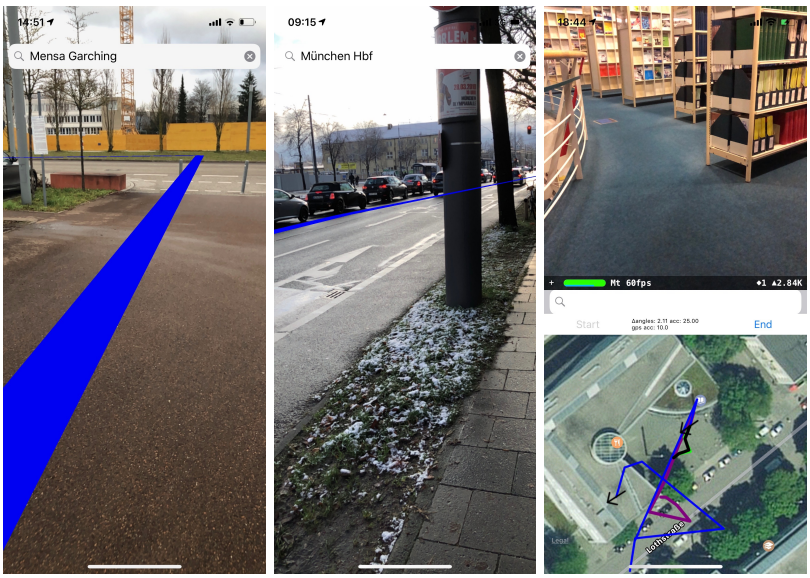


Figure 5.2: Example AR View; left, center: production; right: debug

5.1.2 Mac OS X GUI Application

This application is a native Mac OS X application that acts as a visualization and debugging tool. The via the JSONLogger collected sensor data from the iOS app can be fed into this tool. This was especially valuable during development, because the KFs can be tested on Mac OS X without actually doing a new walk again. Since this application and the iOS app share the same code, the results are the same. The screenshots in chapter 6 were produced on this tool.

5.1.3 CLI Application

The CLI Application is a benchmarking tool to measure the performance of the KFs. It has two main tasks, first using a batch of collected walks and evaluate all primary sensor inputs, as well as the KFs. This produces a lot of different tables and averaging values, that are used to assess the performance in chapter 6. The second task is optimizing the the various parameters of the four KFs.

5.2 Model

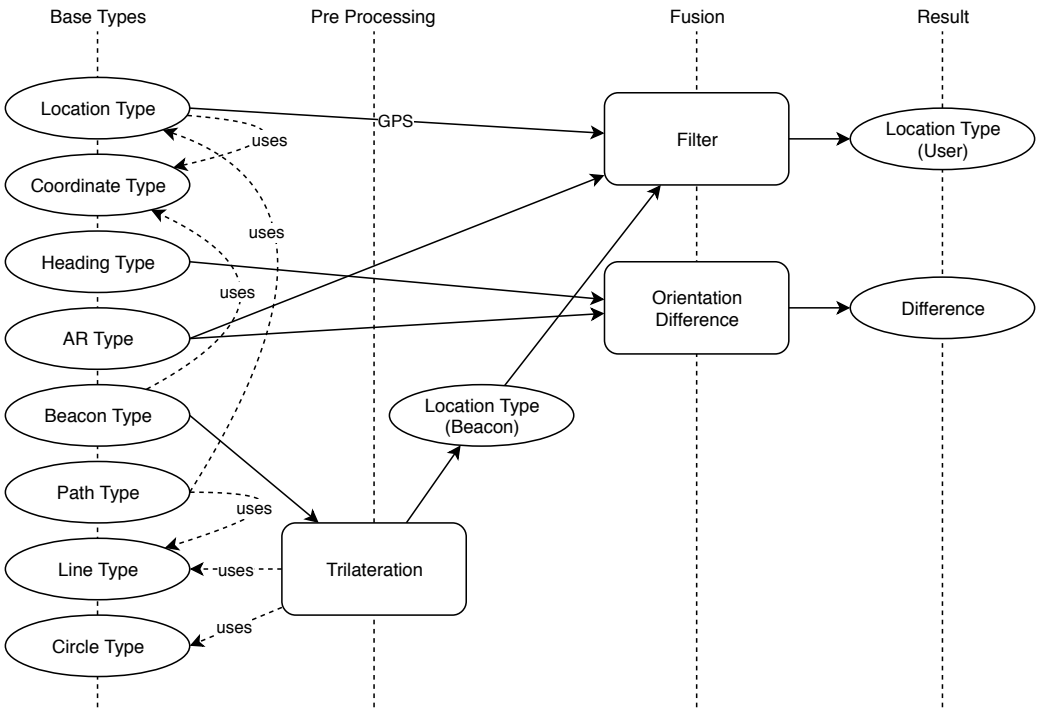


Figure 5.3: Overview Model

The model as part of the MVC is shared by all 3 applications and its internal

architecture can be seen in figure 5.3. The following sections start with the base types that are the heart of the model. The main focus of the base types is providing a custom implementation. This is opposed to reusing and extending Apple's internal types. Reusing is often recommended to avoid duplicate code and swift provides good and easy mechanisms to accomplish that. The downside is a tighter coupling to Apple's ecosystem. The decision to not do this was consciously made by using custom types. This way the code is decoupled and can be transferred to other operating systems or even other programming languages more easily. One example is the Location Type, which is very similar to Apple's type *CLLocation*. In this particular case, the benefit was not just decoupling, but also the ability to use custom distance functions. The second stage that is shown in figure 5.3 is pre processing and consists only of the Trilateration that takes Bluetooth beacon data and computes a location (section 5.2.9). The next step is the fusion which includes all four KFs that are in the figure only summarized as Filter (section 5.2.11) and the Orientation Difference module (section 5.2.12).

5.2.1 Location Type

The Location Type is representing a certain location on the earths surface. This is done via the most commonly used geographic coordinate representation, latitude and longitude. Additionally it holds information about altitude, as well as accuracy data that represents a certain confidence in the values. This type is initiated in various ways. Either directly from GPS measurements which are captured in the Main Controller. Or indirectly from computational results, mainly Trilateration and Kalman Filter.

Furthermore this type encapsules functionality needed to convert between geographic coordinates and *X, Y, Z* displacement coordinates, as described in Augmented Reality Type. A latitude and longitude representation as basis for a geographic coordinate system is used, because most current navigation products use this architecture. Prominent examples are *Google Maps* [23] and GPS itself. Also the used Frameworks *CoreLocation* and *MapKit* both use a latitude longitude representation.

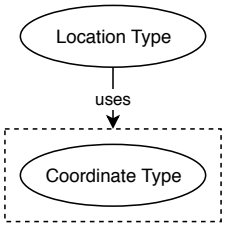


Figure 5.4: Overview Location Type dependencies

The explicit handling of latitude and longitude values, as well as all algorithms needed for geometry on a model of the earth are implemented in the Coordinate Type, which is the only other base type that the location type is using (figure 5.4).

5.2.2 Coordinate Type

The Coordinate Type caries latitude longitude values and provides algorithms for dealing with geometry on the earths surface. There are several models to describe the shape of the earth. Historically they developed towards higher precision and accuracy. But the higher accuracy comes at a cost. More complex models imply higher complexity in computation. Which results among other things in higher energy consumption and is especially important in this application because of the mobile nature. There are three models implemented which go from low complexity and low accuracy to high complexity and high accuracy. The decision of which model to use was done by testing the models against each other, as explained in section 5.6. The following three models have been implemented:

Equirectangular projection This model projects the earths surface on a rectangular plane. This model has several problems because it seemingly expands areas close to the poles. This results in straight lines on the globe not translating in straight lines on the projection. The same is true for distances. The resulting error grows with increasing distance, that means this model is practically only usable for very small distances. The geometric algorithms used in this model are based on the *Pythagorean Theorem*.

Earth as a Sphere Modeling the Earth as a Sphere results in a significant jump in accuracy, especially for larger distances. Only the radius is needed to describe all properties of the sphere. The radius is often approximated as 6371 kilometer, but testing as described in Tests showed that a radius of 6378137 meter results in higher accuracy. This precise value is derived from the World Geodetic System 1984 (WGS 84), which is the standard behind GPS. The geometric algorithms are based on *Haversine Formula* [24] and therefore computationally more complex than the Equirectangular Projection.

Earth as a Ellipsoid The Earths shape can be approximated even better by using a Ellipsoid instead of a Sphere. Defining a ellipsoid is done by having a couple of parameters that describe the deviation from a sphere. There are several different standards to do this for the earth. The standard WGS 84 was chosen because it is used by GPS. The geometric algorithms are based on *Vincenty's Formulae* [25], which are two iterative methods. The methods are giving a higher accuracy, but are computationally even more complex than the algorithms on a sphere.

Independent from the used model, there are certain geometric functions needed. Those are used mainly by Trilateration and all modules that convert between the geographic system and the meter displacements used in the Augmented Reality Type. The geometry functions can be put in the following categories:

Distance This computes the distance (line of sight) in meters between two Coordinates

Distance to two dimensions This category converts between a Coordinate and a north east meter displacement and vice versa.

Compute coordinate This computes a destination Coordinate, based on a start Coordinate, a direction of travel (bearing) and a distance to travel.

Compute bearing This computes the bearing between a start Coordinate and a destination Coordinate. An explanation for bearing can be found in Heading Type

Intermediate point This computes a point at a fraction on the line from a start Coordinate to a destination Coordinate

5.2.3 Heading Type

The Heading Type represents an orientation on a map. This is an angle between 0 and 360 degrees, with 0 being north, east being 90 degrees, south 180 and west 270. This is often also called bearing instead of heading. The type is directly initialized by the Main Controller with data coming from *CoreLocation*. The data is coming from the internal magnetometer sensor, which measures the magnetic field of the earth.

5.2.4 Augmented Reality Type

This type is a custom representation of the information that is provided by Apple's ARKit Framework. The Framework computes additional information based on camera and inertial measurement unit, as explained in more detail in section 3.1 This AR Type combines information about the camera position and orientation in reference to a local coordinate system. The position is a meter displacement in *X, Y, Z* coordinates. The orientation is a unit vector pointing in the pointing direction of the camera. Additionally the type computes a property called *xzOrientation* on initialization. Its an angle which represents the phones general looking direction on the earths surface. Since the ARKit coordinate system is roughly pointing north with the x-Axis and east with the z-Axis, this angle is comparable with the Heading Type. Comparing data from ARKit and the Magnetometer makes it possible to detect unnatural discrepancies that might be caused by failing measurements in either one of those systems.

5.2.5 Beacon Type

The Beacon Type represents all data that is received from Bluetooth beacons. For every beacon this is mainly an identifier and RSSI. The location of each beacon is supplied via a key value store, that assigns a coordinate to each beacon identifier (figure 5.5). The actual distance from the sensing device to a beacon is derived from the signal strength. This highly depends on which beacons are used and how strong their signal is. The beacons that were used have been calibrated by measuring the signal strength

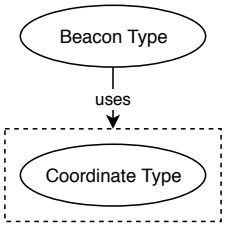


Figure 5.5: Overview Beacon Type dependencies

at close distance, 1 meter and 6 meter distance. The RSSI is -40 for close distance, -60 for 1 meter and -90 for 6 meter. To convert between RSSI (r) and distance (d), a second order polynomial of the form $d = ar^2 + br + c$ is used, which results in the parameters $a = 7$, $b = -11$ and $c = 36$.

5.2.6 Path Type

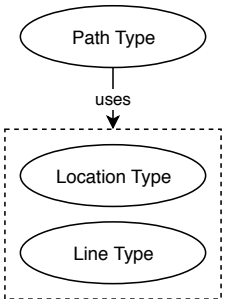


Figure 5.6: Overview Path Type dependencies

The Path Type holds navigation data about the route a user has to take to get to the destination. This is realized by a list of locations (figure 5.6) which embody checkpoints that the user is passing. Usually one of these checkpoints is an intersection or a curve on the route where the user has to change the direction of walking. This type is initialized by the Control Controller which gets the location data directly from *MapKit*.

The Path Type is also used during the Evaluation to have a Ground Truth. Details on the Evaluation can be found in chapter 6. Provided by this type are methods to calculate the distance between a Coordinate and a Path, as well as calculating an intermediate Coordinate at a certain fraction of the way between start and end of the Path.

5.2.7 Line Type

This Type represents a line between two Coordinates. It is directly used by Trilateration and provides methods to intersect two lines, resulting in a intersection Coordinate. Trilateration is only used with small distances below 100 meter. Since in this case

the resulting errors of an Equirectangular projection are very small, the algorithm to intersect lines is simplified for maximum performance. The Line Type is also used by the Path Type to calculate distances from a Coordinate to a line and calculating an intermediate point at a fraction on the line.

5.2.8 Circle Type

The Circle Type describes a circle with a Coordinate as origin and a radius in meters. It is a vital part of Trilateration, where it is used to represent the distance around a Beacon. The main functionality needed by Trilateration is the intersection of two circles resulting in a Line between the intersection points. There are some steps necessary to make this possible. First the distance between the two circles has to be calculated. This is then used to determine how the circles stand in relation to each other. They can be actually the same circle, entirely separate, one can be contained in the other, intersecting once or intersecting in two points. Only the last option is viable to calculate the intersection Line.

5.2.9 Trilateration

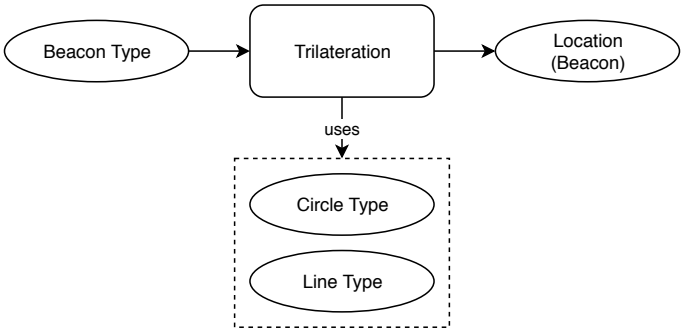


Figure 5.7: Overview Trilateration data flow

The general problem of trilateration is explained in section 3.3 and some details of how this problem is approached in this work can be found in section 4.1. In figure 5.7 the general black box mechanism of the trilateration can be seen. Beacon type data is the input and with the help of the circle and line type, a location is generated.

5.2.10 Circle Geometry

This work uses a properties of intersecting circles. Two circles can either be not intersecting, intersecting once, twice or one circle can be contained in the other. If they are intersecting twice than a line between the two intersection points can be drawn. If there are three circles that each intersect twice with each other, the resulting lines of the

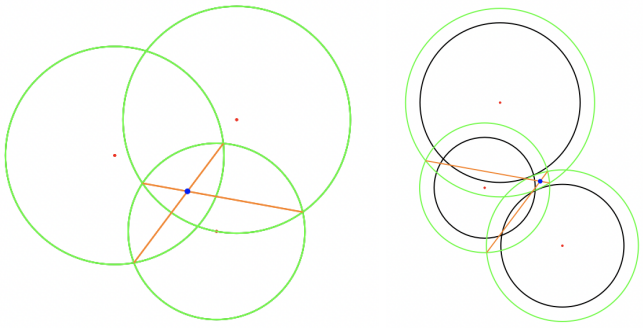


Figure 5.8: Trilateration examples; regular (left); with size increase (right)

intersection points are intersecting themselves in one point that is always well defined. Its actually only necessary to intersect two of these three lines, because the third will have the same intersection point as the first two. An example of such can be found in figure 5.8 on the left side. The intersecting circles are drawn in green, the lines in orange and the intersection point in blue.

In a perfect world with perfect distance values the three circles would actually intersect at one point, which is the same as the line intersection point. But because RSSI values are imperfect because of imprecise measurements, the resulting distance values are also imperfect. The circles are sometimes too large and sometimes too small. The former is not a problem, but the later makes the use the earlier described method impossible. To counter that the size (distances) is artificially expanded until all three circles intersect twice with each other. An example of this can be found in figure 5.8 on the right side, with the bigger intersecting circles drawn in black.

Beacon Location Module

The result from the Trilateration is giving one specific user location, but no accuracy value on how good the trilateration worked. The Beacon Location Module adds an accuracy value by taking the sum of the difference of the found solution to the three distance circles. So if something went wrong in the Trilateration this value will be very high and mean a bad accuracy. Its best be understood as: accurate to x meters. This module than broadcasts the trilateration solution with the accuracy value to everyone who is listening.

5.2.11 Kalman Filters

The general overview of how the data fusion is approached was given in section 4.2. figure 5.9 shows a simple overview on how each filter takes in GPS, Beacon and AR type data and fuses it together to one user location. There are four KFs implemented, according to the equations in sections 4.2.1 to 4.2.4. The specific filters are derived from

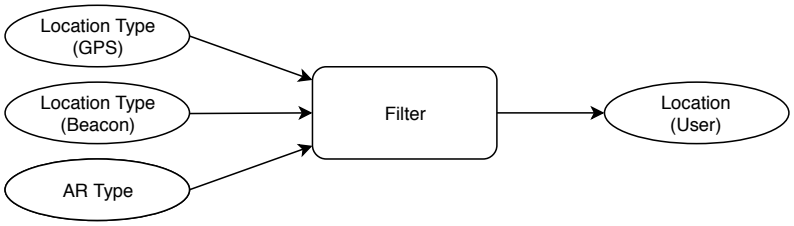


Figure 5.9: Overview Kalman Filters data flow

one generic filter class that implements the prediction and update step. The generic class uses a custom matrix type internally. The matrix type implements all operations needed, such as multiplication, inversion, transposing and more. It is implemented with the use of SIMD as explained in section 4.3

5.2.12 Orientation Difference

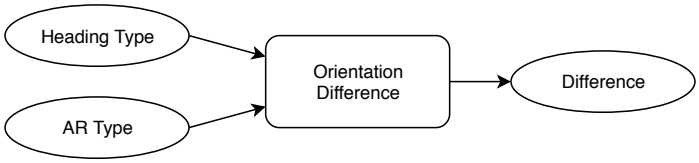


Figure 5.10: Overview Orientation Difference data flow

The Orientation Difference module is a very simple KF that holds a current orientation difference value. A simple black box overview on how it works cn be found in figure 5.10. Its fed by two inputs, the magnetometer (heading type) and the *xzOrientation* of AR type. It calculates the difference between those to orientations and tracks it over time. If the difference is greater than 0 it means the local coordinate system of ARKit got out of sync with the magnetometer data. This can occur because either one of two things. When ARKit is initialized it uses the magnetometer value at this moment in time to setup the local coordinate system. This value might have been inaccurate, because its just a single value that naturally has noise associated. The second reason could be a temporary failure of ARKit, which happens almost never. But when it happens it results in constant errors from that point on, which can be corrected with this method.

A KF was used here to smooth the difference data here, so the module does not react to temporary small differences. As soon a difference is persistent over a couple of frames, the coordinate system of ARKit is reset with the smoothed difference amount. This is done by telling the AR Controller the calculated difference.

5.3 View

5.3.1 Map View

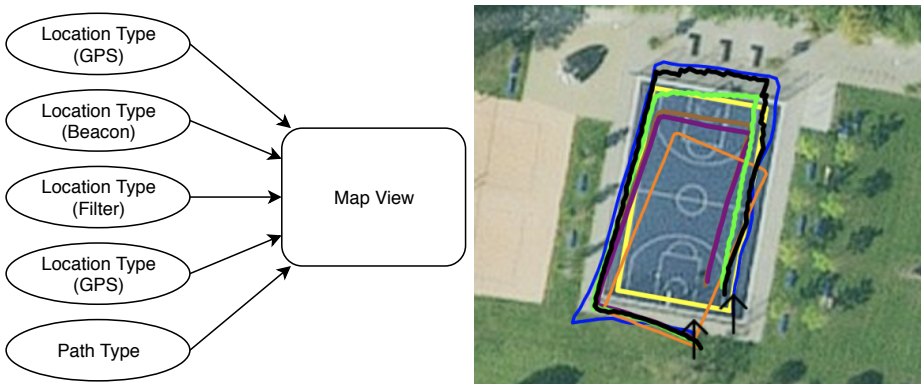


Figure 5.11: Overview Map View data flow (left); example Map View (right)

This view is showing the results of all four KFs on a satellite image of the current location. The raw GPS, Beacon AR data and path is also visualized. A schematic data flow diagram can be found on the left side of figure 5.11. The right side shows an example on how that map view looks like. The AR data is not inherently drawable on a map, because its captured in a local reference frame. Therefore its just connected to the first real location measurement that is made and the the north and east movement is drawn. This view is the main view of the Mac Application and the lower part of the debugging version of PAReNt.

5.3.2 AR View

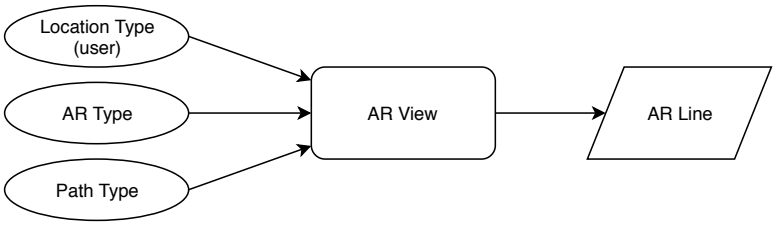


Figure 5.12: Overview AR View data flow

This view is used in the iOS app to display the path in the camera image depending on the user location (figure 5.12). Its basically the live image that the back-facing camera captures, augmented with a blue path. Its realized via ARKit internal functionalities and a custom type called *ARLine*. Screenshots of how that looks where shown earlier figure 5.2

5.4 Controller

The controller as part of MVC serves as a direct interface to operating system. The following controller described in sections 5.4.1 to 5.4.3 are only used by the iOS application and use functionalities specific to the platform. There is also one controller for the Mac OS application, which is not explained here because it does not contain any interesting functionality.

5.4.1 Main Controller

The Main Controller is the main interface between the underlying operating system and the application. Its the starting point of the application and sets up different parts. Those include the JSON Logger, Kalman Filter and the Beacon Trilateration Module. The Main Controllers most important task is configuring and managing the iOS Framework *CoreLocation*. The framework provides the connection to several hardware components, in this application the GPS chip, the magnetometer and the Bluetooth module. The configuration consists of steps like getting authorization from the user to use location services and requesting the highest level of precision and update intervals. Once the framework is started, the Main Controller captures all incoming data and processes it in the following ways.

Location Data The incoming location data is first converted from the Apple type *CLLocation* into the custom Location Type (the reasons for using custom types is explained in section 5.2). Then the data is distributed via mechanisms of the observer pattern to every module that is interested.

Magnetometer Data The data coming from the magnetometer hardware module is also converted to the custom Heading Type and similarly distributed.

Bluetooth Beacon Data Apple intends Bluetooth Beacons as a positioning mechanism where GPS is not available. Therefore beacons are also managed via *CoreLocation* and not by the *CoreBluetooth* Framework. *CoreLocation* collects all beacon data and once a second provides it to the application. The Beacon Type is then initialized with this data. Similar to the Location and Magnetometer Data, the distribution to theBeacon Trilateration Module takes place via the observer pattern.

The outcome is that the Main Controller provides GPS, Heading and Beacon data to the rest of the application. A simple visualization of that can be found in figure 5.13.

5.4.2 AR Controller

The AR Controller serves as an interface to ARKit. It initializes the AR session and registers the event handlers for incoming data sensed by ARKit. As soon as ARKit did

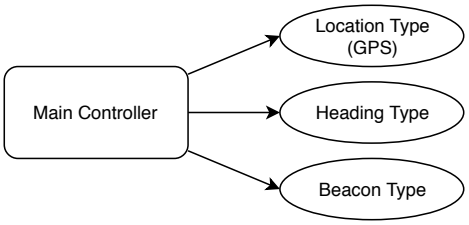


Figure 5.13: Overview Main Controller data flow

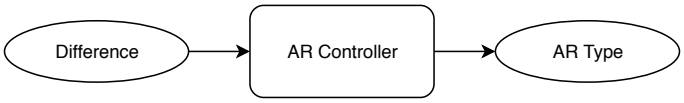


Figure 5.14: Overview AR Controller data flow

process a new frame, the resulting data is converted into a AR Type and distributed to everyone that is listening view the observer pattern (figure 5.14). Since ARKit is handled here, the coordinate system is also rotated here, if the Orientation Difference module detects a inconsistency. Besides handling ARKit, it also directs the AR View. If a new path is broadcasted by the (Navigation Controller), the path coordinates are converted into the AR space. Here is also the ground sensing handled. If ARKit senses a new vertical plane, the controller checks if its y-Axis (up) value deviates more than 10cm from the current ground height. If that is the case, the value is adjusted and directed to the AR View.

5.4.3 Navigation Controller

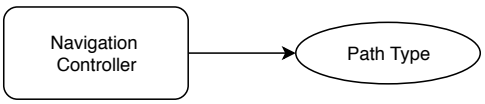


Figure 5.15: Overview Navigation Controller data flow

The Navigation Controller's main task is providing the search bar for searching and choosing a navigation destination. It acts in a way that is expected from a navigation app, meaning it provides suggestions while typing. When a navigation destination was chosen it finds the corresponding route, converts it into a path type and broadcasts it (figure 5.15). In the debugging version of PAReNt it also has buttons to start and end a evaluation walk.

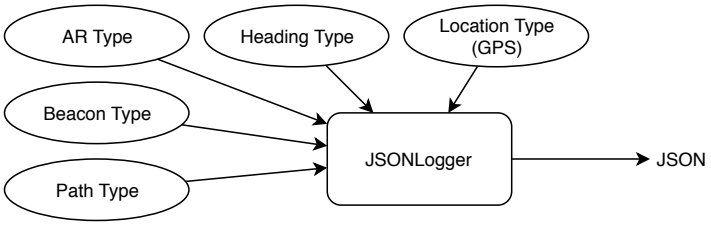


Figure 5.16: Overview JSONLogger data flow

5.5 JSONLogger

The JSONLogger listens to all raw sensor events as well as custom system messages and logs them, a black box view of that can be found in figure 5.16. The sensor events are location (GPS), magnetometer, AR type data and Beacons. The system message include start and ending of evaluation runs and path related data. The events are encoded into JSON objects and are then saved onto the phones persistent memory. Alternatively they can also be send to a network address.

5.6 Unit Tests

Additionally the section Tests gives an overview about the unit tests that are included.

Several critical functionalities of this project can not be solved analytically. The unit tests described in this section are mainly used to make sure correct results are produced, even with imperfect algorithms. Furthermore the performance of different algorithms which solve the same problems is compared. Having different options for algorithms enables a fine tuning between precision and computational complexity. In other words, the highest accuracy might not be preferable if it results in significantly more computations and therefore increased energy consumption.

All algorithms that used for computations on a geographic coordinate systems fall into this category. Especially those that convert from Augmented Reality measurements, into GPS or indoor location positioning. As described in AR Type, the camera position is generally given as a X, Y, Z displacement in meter values. By contrast the Location Type is based on a latitude and longitude system. The different algorithm options described in detail in Coordinate Type are also tested for accuracy and performance. With the results of these tests, a medium level of accuracy was chosen. The high accuracy option did not yield any significant accuracy benefit in smaller distances (below 1 kilometer), but resulted in a measurable performance decrease. The low accuracy option on the other hand decreased in accuracy with growing distances, but did not increase performance significantly.

6 Evaluation

The evaluation of the proposed work is described in the following sections. The first section describes the 2 used data sets that have been recorded and the testing methods that have been used on those data sets. The following four sections describe evaluations of different types of input data and the performance on the proposed KFs.

Note: All data mentioned in tables is in the unit meter, unless specified differently

6.1 Data Sets & Testing Methods

This section explains the details of the two data sets that were recorded. Both of them contain a ground truths and sensor data captured from walks. The walks were done in a fashion that seems natural for augmented reality navigation. The phone was hold in front of the chest with a slight tilt forward. In this position the user can see on the screen where he is walking and also see parts of the ground and the displayed navigation path. It was tried to walk in a steady pace without stopping in between and as little deviation from the path as possible. This is especially important for the testing methods described in subsection 6.1.4 and subsection 6.1.5. Of course this can't be done with perfect precision, because of humans being prone to error. All data was captured on an iPhone X. An example video of one evaluation run can be found here: `https://youtu.be/8fg1kPD92Q8`

The position of the ground truth for both data sets where taken from satellite images on Apple Maps. This was the only viable option since there are currently no alternatives besides taking location via GPS, which yields a worse precision. After the coordinates where taken from the satellite images, the distances between them were calculated and compared with real world measurements to ensure the satellite image is not distorted. The only inaccuracy left could be that the images are moved. If this is the case, all recorded and computed data is actually more accurate as evaluated in the following sections.

Note: The ground truth is internally realized via the Path Type. Therefore, the terms ground truth and path or path data is used interchangeably.

6.1.1 Data Set 1; GPS & AR

This is recorded data from walks around a basketball court on the university grounds. During the recording no beacons where deployed, hence the recording consists only

Figure 6.1: Ground Truth / Path of Data Set 1 (left) and Data Set 2 (right)

of GPS and Augmented Reality data. The ground truth is the outer boundary of the court. Each data set entry is the captured data of one walk starting from the south east corner, around the court in clockwise direction. Figure 6.1 (left) shows the basketball court with the ground truth marked in yellow.

6.1.2 Data Set 2; Beacons & GPS & AR

This data set was created on the same basketball court as Data Set 1, but only along one side line of the court. Additionally 9 beacons were deployed on the right and left of this path. The distance between each beacon and the path is 3 meter and the distance between the beacons on each side is 6 meter. Finally the left side has a 3 meter offset to the right side, resulting in a zig-zag pattern. This can be seen in Figure 6.1 (right) with the path in yellow and the beacons in white. The beacons were deployed in this specific way, to support the Trilateration, by having always three beacons in close range. A walk on this data set starts at the south end of the line and ends at the north end.

In comparison Data Set 2 is smaller than Data Set 1, because it is only one side of the Basketball court. Of course more data is always preferable and this limitation attributed to the limited amount of beacons that were available. One limitation is that the Testing Method Distance Start to End cannot be used on this data set, because the end point is different to the starting point. On the other hand this data set is very versatile in regard to available sensor data. Since it hat also been recorded outside, it contains not just valid beacon data, but also usable GPS data. So it can be used to fuse GPS, Beacon and AR data, which is discussed in section 6.5

6.1.3 Testing Method 1; Distance to Ground Truth

This method is measuring the distance of each recorded location to the Ground Truth. So for example if the first GPS location is one meter next to the start point of the Ground Truth, this results in an error of 1m. This is a simple and plausible measure to assess the accuracy of a signal, whether its GPS, or one of the Kalman Filters. For example, in

	mean	median	max	min	σ	MSE
avg GPS	1.74	1.29	8.46	0.04	1.70	7.20
avg AR	2.04	1.78	5.37	0.00	1.29	7.78
avg KF	1.23	1.00	4.64	0.00	1.01	3.12
avg KF Vel	1.33	1.05	5.37	0.00	1.15	3.63
avg KF CT	1.26	1.03	4.28	0.03	0.96	3.13
avg KF Vel CT	1.43	1.08	4.62	0.03	1.18	3.81

Table 6.1: Evaluation (Data Set 1); GPS & AR; Distance to Ground Truth

the first row of Table 6.1 it can be seen that the recorded GPS signal deviates on average $1.74m$ (averaged over all walks of Data Set 1). To assess the performance between lets say GPS and a Kalman Filter the mean squared error (MSE) in the last column of the mentioned table is used.

There is one downside of this method. If a filter produces results that are just staying at the start location and don't move at all, this method would result in a very low error, even though this filter actually really bad.

6.1.4 Testing Method 2; Distance to correct point on Ground Truth

	mean	median	max	min	σ	MSE
avg GPS	3.80	3.54	9.70	0.77	2.03	22.57
avg AR	4.80	4.60	7.18	2.58	1.33	30.29
avg KF	3.63	3.21	6.82	1.13	1.55	19.27
avg KF Vel	3.58	3.27	6.86	0.91	1.57	18.60
avg KF CT	3.46	3.16	6.96	1.01	1.70	18.13
avg KF Vel CT	3.42	3.19	7.33	0.89	1.66	17.25

Table 6.2: Evaluation (Data Set 1); GPS & AR; Distance to correct point on Ground Truth

To counter the just mentioned downside of just taking the distance to Ground Truth, this method is measuring the distance of a recorded location to the correct point on the Ground Truth. Because the Data Sets where recorded walking with a steady pace, it can be derived that the user is at 50% of the path after 50% of the elapsed time. Again the MSE of this measure is the best way of compare the Kalman Filters with each other and to the sensor signals like GPS. An example for this kind of error can be found in Table 6.2, where the last column shows MSE. It can be seen that MSE of all Kalman Filters is significantly lower than from GPS alone. More detail on fusion of GPS and AR follows later in section 6.3.

The downside of this method is that it is more dependent on the recording of a

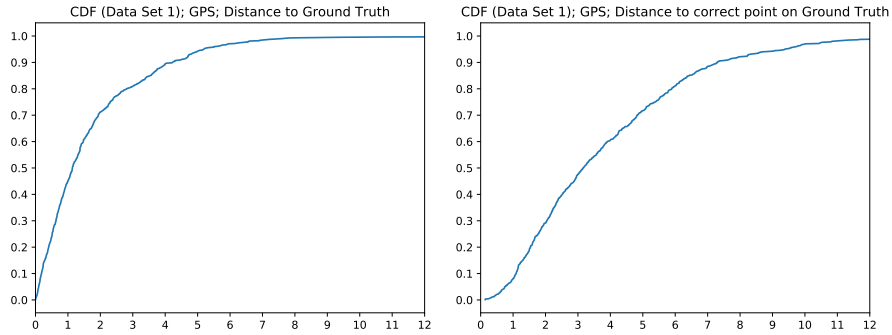


Figure 6.2: GPS error CDF of Distance to Path (left) and Distance to correct point on Grund Truth (right)

walk. Walking in a steady pace is necessary and since its done by a human, prone to error. A little delay after pressing the start button and actually starting to walk, introduces a constant error because the user is a moment behind the position where he actually should be. Similar, stopping the recording contributes to the same issue. This phenomenon can be seen when looking at the cumulative distribution functions (CDFs) of the error of the GPS signal in Data Set 1 (Figure 6.2). The left plot shows the CDF for the first method where the amount of errors decreases with the magnitude of them. Or in other words, the smallest errors occur most frequently. The slope of the function constantly declines. For the second method (Distance to correct point on Ground Truth), the talked about phenomenon can be seen in the right plot. The slope first is almost zero around the zero error mark, increases until approximately $2m$ and the declines again. This is a sign that with this method there is a error introduced, just by human error of the measurement.

Both methods have reasons to exist and it cant be said that one method is superior to the other.

6.1.5 Testing Method 3; Distance Start to End

	path length	**distance start end**	**%**
avg GPS	80.63	3.55	+4.40
avg AR	80.63	0.67	+0.83
avg KF	80.63	3.93	+4.88
avg KF Vel	80.63	3.75	+4.65
avg KF CT	80.63	4.02	+4.99
avg KF Vel CT	80.63	3.66	+4.54

Table 6.3: Evaluation (Data Set 1); GPS & AR; Distance between start and end

This method calculates the distance between the start and the end point. This testing method makes a lot of sense when used on Data Set 1, since those walks end at the same location as they start. The resulting distance is then set into relation to the length of the path. So as an example, a path length of $80.63m$ and a start to end distance of $67cm$ results in a 0.83% error. this is especially suited to assess systems that are not producing results in a global reference frame, but in a local. One example of a local reference frame system is actually ARKit, hence the evaluation of ARKit in subsection 6.2.3 is using this method.

Other local reference frame systems like pedestrian dead reckoning (PDR) systems are using this evaluation method as well. For example Jimenez et al. [12] reached a accuracy of smaller then 5% for an shoe mounted IMU.

Using this method on assessing the Kalman Filters developed in this work is less valuable. Kalman Filter as described in section 3.5 take bad estimates as initial values and improve with every following input value. The filters try to estimate the the users position in a global reference frame, not the exact movement. This should result in worse values than ARKit alone. Which is the case, as can be seen in Table 6.3

6.1.6 Testing Method 4; Length of recorded data

Figure 6.3: Jittering signal of KF Vel CT (purple) and smooth signal if KF CT (purple) in 2018-11-15 14:52:00.json

This method measures the length of the recorded data. It takes the distance between each location and accumulates it to a total length. This is then set into relation to the length of the path. As an example path length of $80.63m$ and a recorded length of 76.16 results in a -5.93% error (Table 6.10: avg AR). Similar to the last method, described in subsection 6.1.5, this method is valuable when looking at local reference frame systems and its used in ARKit Evaluation as well. But in contrast to the last method, it is also very interesting and valuable to look at the error of the KFs. Because a high error is an indication of jitter in the data. A low error on the other hand does shows a good result anyways, but also is a characteristic of a smooth estimation. An example of a

jittering filter signal can be found in Figure 6.3. The purple line shows the Filter next to the GPS line (blue) and the beacon line (red). An example for a smooth filter can be found is the brown line, also in Figure 6.3.

6.2 Raw Sensor Data

6.2.1 GPS

	mean	**median**	**max**	**min**	σ	**MSE**
2018-07-24 11:31:16.json	5.48	5.10	13.07	0.14	3.02	39.14
2018-07-24 11:32:52.json	4.44	4.05	10.03	0.25	2.80	27.55
2018-07-24 11:34:23.json	2.59	2.32	5.95	0.60	1.37	8.59
2018-07-24 11:35:52.json	3.93	4.22	7.13	0.45	1.78	18.59
2018-11-15 11:43:21.json	7.52	7.25	11.34	4.23	1.91	60.16
2018-11-15 11:44:49.json	2.08	1.51	6.12	0.14	1.61	6.92
2018-11-15 11:46:08.json	2.28	2.19	4.28	0.72	0.89	6.00
2018-11-15 11:48:33.json	1.96	1.59	4.10	0.15	1.11	5.06
2018-11-15 11:49:46.json	2.77	2.78	4.23	0.81	0.85	8.37
2018-11-15 11:54:54.json	4.74	3.72	31.94	0.43	4.94	46.90
2018-11-15 11:56:08.json	4.07	4.17	8.49	0.55	2.11	20.98
2018-11-15 14:35:08.json	1.80	1.97	3.35	0.30	0.98	4.19
2018-11-15 14:41:31.json	2.68	2.76	5.92	1.05	1.08	8.37
2018-11-15 14:50:04.json	2.81	2.70	4.68	1.76	0.75	8.48
2018-11-15 14:52:00.json	5.16	3.64	11.58	0.56	3.59	39.49
2018-11-15 14:52:53.json	6.56	5.77	13.85	1.17	4.34	61.79
2018-11-15 14:53:46.json	2.92	3.22	3.93	0.72	1.02	9.59
2018-11-15 14:54:34.json	2.92	1.93	6.40	1.11	1.84	11.91
2018-11-15 14:55:25.json	3.32	2.82	7.33	0.69	2.06	15.29
average	3.69	3.35	8.62	0.83	2.00	21.44
σ	1.56	1.46	6.33	0.90	1.19	18.38

Table 6.4: Evaluation (Data Set 1 & 2); GPS; Distance to correct point on Ground Truth

section 3.4 already talked about accuracy of GPS and in general GNSS systems. This evaluation produces values for all the four testing methods. The most precise one is distance to the correct point on path. A detailed listing of the data for each recordings can be Table 6.4. The mean error, averaged over all 19 recordings is $3.69m$, with an average standard deviation (σ) of $2m$. This shows a slightly better performance as the $5m$ GPS accuracy explained in section 3.4. The average distance between start and end, found in Table 6.3, which is $3.55m$ mirrors about the same accuracy. By taking a closer look at the detailed first mentioned Table 6.4, it can be seen that there are

Figure 6.4: GPS signal in 2018-11-15 11:43:21.json (left) and 2018-07-24 11:34:23.json (right)

maximal errors of up to 31.94m and also the median of each recording is lower than the mean. This is a sign for a generally good performance with somewhat rare outliers. These outliers seem to be often in connection to a bad GPS signal at the beginning of a recording, when there is not yet a good signal from more than a few satellites. An example of a somewhat bad and somewhat good GPS signal can be found in Figure 6.4. The left image shows a bad signal including an initial outlier in the right lower corner. The right image is showing a clearly better signal.

6.2.2 Bluetooth Beacons

	mean	**median**	**max**	**min**	σ	**MSE**
2018-11-15 14:35:08.json	2.65	1.59	14.16	0.23	3.55	19.59
2018-11-15 14:41:31.json	1.44	1.20	3.46	0.11	1.04	3.16
2018-11-15 14:50:04.json	0.78	0.63	1.78	0.11	0.50	0.86
2018-11-15 14:52:00.json	1.20	1.14	2.66	0.13	0.69	1.91
2018-11-15 14:52:53.json	0.99	1.17	1.57	0.00	0.47	1.21
2018-11-15 14:53:46.json	1.15	1.07	3.54	0.05	0.78	1.91
2018-11-15 14:54:34.json	1.94	1.68	4.25	0.51	1.04	4.86
2018-11-15 14:55:25.json	1.21	1.30	3.20	0.09	0.84	2.16
average	1.42	1.22	4.33	0.16	1.11	4.46
σ	0.56	0.30	3.81	0.15	0.94	5.84

Table 6.5: Evaluation (Data Set 2); Beacon; Distance to Ground Truth

For the evaluation of the Bluetooth Beacons Trilateration system, Data Set 2 is used. By looking at the simple distance to Ground Truth in Table 6.5, a mean error of 1.42m and a MSE of 4.46m averaged over all walks can be seen. Immediately the outlying MSE of 19.59m of *2018-11-15 14:35:08.json* attracts attention. In Figure 6.5 this walk

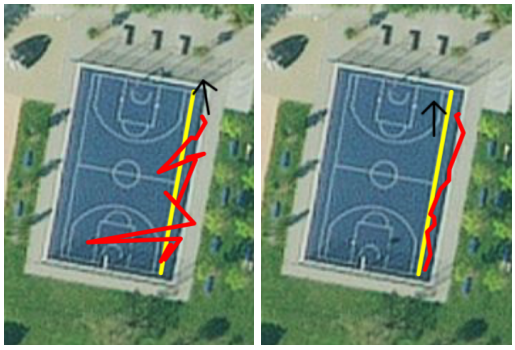


Figure 6.5: Beacon (red) signal in 2018-11-15 14:35:08.json (left) and 2018-11-15 14:52:00.json (right)

(left) is visualized together with a *2018-11-15 14:52:00.json* (right), which has a MSE of only 1.90m. The differences are striking and can be traced back to the one worst outlier of (14.16m) on the south west side. Why outliers like that may occur, as well as what is done against them is explained in section 5.2.10. The comparison with GPS on Data Set 2 can be found in Table 6.13. The developed beacon system is significantly better using this method of testing than GPS, with an average MSE of 4.46m versus 24.43m.

	mean	**median**	**max**	**min**	σ	**MSE**
2018-11-15 14:35:08.json	3.56	3.15	11.34	0.33	2.44	18.61
2018-11-15 14:41:31.json	3.56	3.55	6.41	0.33	1.64	15.34
2018-11-15 14:50:04.json	2.82	2.52	4.69	0.99	1.04	9.06
2018-11-15 14:52:00.json	4.97	4.23	10.20	0.78	3.62	37.78
2018-11-15 14:52:53.json	4.51	2.58	11.10	0.83	3.59	33.21
2018-11-15 14:53:46.json	3.18	2.89	4.99	1.81	1.08	11.26
2018-11-15 14:54:34.json	3.04	2.75	4.68	1.93	0.85	9.95
2018-11-15 14:55:25.json	4.11	4.45	5.14	2.07	0.84	17.55
average	3.72	3.27	7.32	1.13	1.89	19.10
σ	0.70	0.69	2.82	0.66	1.11	10.07

Table 6.6: Evaluation (Data Set 2); Beacon; Distance to correct point on Ground Truth

Upon looking on the second testing method: Distance to correct point on Ground Truth in Table 6.6, a totally different picture emerges. A the average values for mean error of 3.72m and MSE of 19.10m are about the same as for GPS (mean: 3.52m, MSE: 19.89, Table 6.14). The recent example of the right image in Figure 6.5, even has a MSE of 37.78. The discrepancy in data is an example of the the problems of the first method (explained in subsection 6.1.3), as well as added error in the second method (explained in subsection 6.1.4). Very little movement can result in a good score in the first method, even though its actually too little movement, which will only show

up in the second method. But the second method introduces error caused by human imperfection. Additionally it might be that there is a delay somewhere in the whole beacon system. This would mean that all computed locations are behind the actual position the user is. To test this hypothesis, more work needs to be done, which will be talked about in section 7.3

	path length	**recorded length**	**deviation (%)**
2018-11-15 14:35:08.json	25.90	75.88	+65.86
2018-11-15 14:41:31.json	25.90	25.54	−1.42
2018-11-15 14:50:04.json	25.90	23.45	−10.47
2018-11-15 14:52:00.json	25.90	24.13	−7.35
2018-11-15 14:52:53.json	25.90	21.93	−18.12
2018-11-15 14:53:46.json	25.90	23.64	−9.56
2018-11-15 14:54:34.json	25.90	25.50	−1.60
2018-11-15 14:55:25.json	25.90	27.75	+6.66
average	25.90	30.98	+3.00
σ	0.00	17.05	24.74

Table 6.7: Evaluation (Data Set 2); Beacon; Length related data

The results of the third testing method with length related data can be found in Table 6.7. The average length deviation of all walks is only +3%, but has a standard deviation of 24.74%. Even though the average is better than for example GPS on this data set (+9.48%, Table 6.15), the very high standard deviation means the data is very inconsistent. 6 out of 8 walks have a recorded length that is lower than the actual length, but outlier *2018-11-15 14:35:08.json* with +65.86% pulls the average close to zero.

6.2.3 Augmented Reality

Figure 6.6: ARKit (orange) signal in 2018-07-24 11:31:16.json (left) and 2018-11-15 11:49:46.json (right)

The data provided by ARKit is fundamentally different to the data provided by GPS or Beacons. GPS and Beacon provide data in a global reference frame, whereas AR data is in a local reference system. As described in detail in section 3.1 that means all data is only displacement values from the initial point (origin of local coordinate coordinate system). And the direction of movement is determined by the first value of the compass. For displaying purposes the global position of the local coordinate system was associated with the first provided global position (GPS or Beacons). This can end up in a bad alignment of the AR signal and path if the first location is inaccurate, as can be seen on the left side of Figure 6.6. In the same figure on the right side, it can be seen that sometimes aligns pretty well, if by coincidence the first values of compass and GPS are accurate. Because AR data is only given in a local reference frame, it does not make any sense to evaluate based on Testing Method 1 or Testing Method 2. This is also mirrored in the data. In both Table 6.1 and Table 6.2 the average mean, median and MSE is worse than for GPS and every Kalman Filter. Interestingly the standard deviation (σ) with $1.29m$ and respectively $1.33m$ are significantly better than the values for GPS and better or roughly the same as the Kalman Filter. This means the error is rather consistent, which speaks for smooth signals without outliers. This observation is also consistent with the visualizations (Figure 6.6)

Evaluating with the help of the length (subsection 6.1.6) and the distance start to end method (subsection 6.1.5) to assess the accuracy of ARKit makes more sense. Table 6.8 shows an average deviation of recorded length of $+2.52\%$ measured on both data sets. By looking at the particular recorded walks, deviations of up to $+15.55\%$ and -9.88% as seen. This explains the rather high standard deviation of 7.01%.

The start end distance is only well defined in Data Set 1, as explained in subsection 6.1.4. Table 6.9 shows an average of only 0.83% distance error in relation to the path length. The standard deviation is also only 0.93%.

In conclusion it can be said that ARKit handles rotations in movement better than actual distances of movement itself. The high precision regarding the start to end distance shows a very precise measuring of the turns in each corner of the basketball court. In fact, as an example where this did not work very well, it can be looked at the left image in Figure 6.6. It can be seen that the long parts of the orange line (corresponding to the east and west side lines) are not perfect parallel. This small deviation resulted in a distance of $1.34m$ between the start and endpoint, which is the second worst value of Table 6.9.

Note: All data was recorded in a fashion described in section 6.1. Different ways of holding the smartphone might result in different results. More in section 7.3

6.3 Fusion GPS and Augmented Reality

The Evaluation of the Fusion of GPS and Augmented focuses on Data Set 1. The results of the first Testing Method can be found in Table 6.1. It can be seen immediately that

	path length	recorded length	deviation (%)
2018-07-24 11:31:16.json	80.63	73.38	−9.88
2018-07-24 11:32:52.json	80.63	76.29	−5.69
2018-07-24 11:34:23.json	80.63	75.47	−6.84
2018-07-24 11:35:52.json	80.63	75.15	−7.30
2018-11-15 11:43:21.json	80.63	80.33	−0.38
2018-11-15 11:44:49.json	80.63	76.12	−5.93
2018-11-15 11:46:08.json	80.63	75.47	−6.85
2018-11-15 11:48:33.json	80.63	77.15	−4.52
2018-11-15 11:49:46.json	80.63	75.19	−7.24
2018-11-15 11:54:54.json	80.63	76.87	−4.90
2018-11-15 11:56:08.json	80.63	76.29	−5.69
2018-11-15 14:35:08.json	25.90	27.76	+6.70
2018-11-15 14:41:31.json	25.90	24.22	−6.96
2018-11-15 14:50:04.json	25.90	30.67	+15.55
2018-11-15 14:52:00.json	25.90	24.86	−4.20
2018-11-15 14:52:53.json	25.90	25.48	−1.65
2018-11-15 14:53:46.json	25.90	25.15	−2.98
2018-11-15 14:54:34.json	25.90	30.52	+15.14
2018-11-15 14:55:25.json	25.90	24.85	−4.22
average	57.59	55.33	−2.52
σ	27.02	24.51	7.01

Table 6.8: Evaluation (Data Set 1 & 2); AR; Length related data

all four Kalman Filter evaluate at about half the MSE of GPS or Augmented Reality. KF (mean: 1.23m, σ 1.01m, MSE 3.12m) stands out with the best values. An MSE of 3.12m in comparison to the GPS MSE of 7.2m is an improvement of 57%

Looking at the outcome of the second Testing Method in Table 6.2 similar results can be seen. All four Kalman Filter are better than GPS and AR alone. This time the KF Vel CT scores best with a mean error of 3.42m and a MSE of 17.25m, which is an improvement of roughly 24%. With the first measure, KF had the best score, this time it has the worst result of the four.

The outcome of the third Testing Method that has length measurement results, can be found in Table 6.10. As seen earlier (subsection 6.2.3), AR is very precise in length measurements, but measures on average −5.93% to little. Two filters manage to yield slightly better results in this category (KF CT: −4.87%; KF Vel CT: +5.59%). The bad performance of the other two filters might be traced back to jitter, as explained in subsection 6.1.6.

Looking at one detailed example of Data Set 1 in Figure 6.7, a lot of findings from the data are mirrored in the images. The GPS signal drawn in blue is somewhat distant

	path length	distance start end	%
2018-07-24 11:31:16.json	80.63	1.34	+1.66
2018-07-24 11:32:52.json	80.63	0.52	+0.65
2018-07-24 11:34:23.json	80.63	0.17	+0.21
2018-07-24 11:35:52.json	80.63	0.13	+0.16
2018-11-15 11:43:21.json	80.63	2.74	+3.39
2018-11-15 11:44:49.json	80.63	0.13	+0.16
2018-11-15 11:46:08.json	80.63	0.29	+0.35
2018-11-15 11:48:33.json	80.63	0.05	+0.06
2018-11-15 11:49:46.json	80.63	0.67	+0.84
2018-11-15 11:54:54.json	80.63	0.46	+0.57
2018-11-15 11:56:08.json	80.63	0.85	+1.06
average	80.63	0.67	+0.83
σ	0.00	0.75	0.93

Table 6.9: Evaluation (Data Set 1); AR; Distance between start and end

	path length	recorded length	deviation (%)
avg GPS	80.63	99.93	+14.01
avg AR	80.63	76.16	−5.93
avg KF	80.63	89.06	+9.18
avg KF Vel	80.63	93.04	+12.71
avg KF CT	80.63	77.14	−4.87
avg KF Vel CT	80.63	86.09	+5.59

Table 6.10: Evaluation (Data Set 1); GPS & AR; Length related data

from the Ground Truth and is irregular in a way that it does not assemble the straight lines which where actually walked. The AR signal in orange mirrors the walking of straight lines and sharp turns pretty good, but is shifted from the correct location. The shift is explained in subsection 6.2.3. The worse performance of GPS and AR in Testing Method 1 and Testing Method 2 can be easily seen in the images. The Deviation to the Ground Truth of GPS and AR is clearly worse than all the Kalman Filters. When looked very close at KF and KF Vel, the jitter in the signal can be seen, that is resulting in bad length recordings.

Neither visually nor derived from the data its immediately clear which of the Kalman Filters yields the best result, more on that in section 6.6. But it is clear that all four filters result in better accuracy than GPS alone. If looked at the MSE of the first Testing Method, even more than 50% improvement can be concluded.

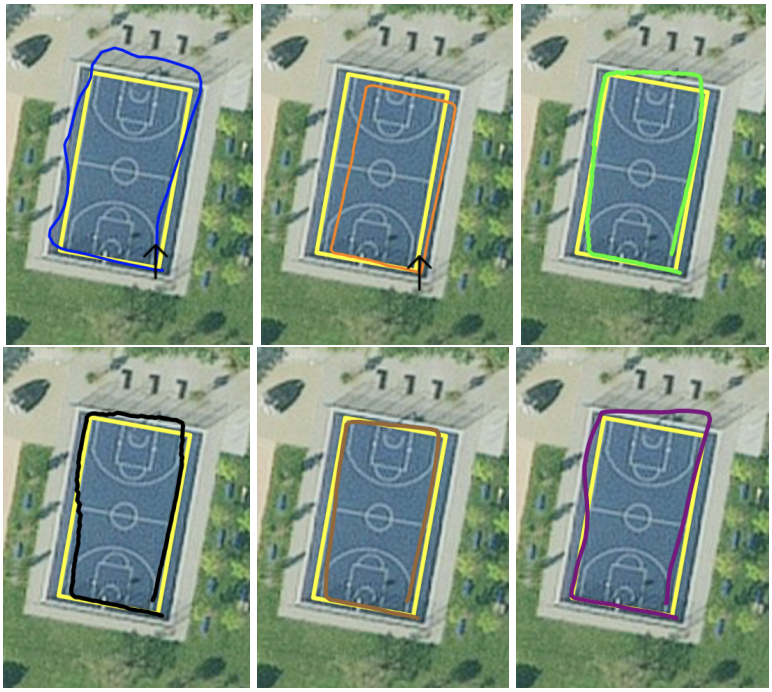


Figure 6.7: 2018-11-15 11:48:33.json; GPS (blue), AR (orange), KF (green), KF Vel (black), KF CT (brown), KF Vel CT (purple)

6.4 Fusion Beacons and Augmented Reality

	mean	**median**	**max**	**min**	σ	**MSE**
avg Beacon	1.42	1.22	4.33	0.16	1.11	4.46
avg AR	3.12	2.65	6.64	0.59	1.65	33.61
avg KF	1.26	0.79	3.94	0.13	1.07	5.15
avg KF Vel	1.37	1.08	4.28	0.01	1.21	7.66
avg KF CT	0.92	0.62	3.02	0.23	0.79	2.47
avg KF Vel CT	1.11	0.95	2.93	0.18	0.78	2.22

Table 6.11: Evaluation (Data Set 2); Beacon & AR; Distance to Ground Truth

When talking about beacons, Data Set 2 has to be pulled up. First up the results of Testing Method 1 in Table 6.11. As already mentioned in subsection 6.2.2, beacons perform very well with a mean of 1.42m and an MSE of 4.46m average over Data Set 2. The data for AR in this table is not really comparable, explained in subsection 6.2.3. Looking at the four Kalman Filters, their means are all lower than the beacon value of 1.42m (KF: 1.26m, KF Vel: 1.37m, KF CT: 0.92m, KF Vel CT: 1.11m). The first two filters have the worst mean, and their MSE is even worse than the beacon MSE (KF: 5.15m,

KF Vel: 7.66m). KF CT has the by lowest mean (35% better than beacons), but only the second lowest MSE with 2.47m (45% better than beacons). KF Vel CT scores a slightly higher mean (22% better than beacons), but has the best MSE with 2.22m (50% better than beacons).

	mean	**median**	**max**	**min**	σ	**MSE**
avg Beacon	3.72	3.27	7.32	1.13	1.89	19.10
avg AR	4.21	4.10	7.28	2.54	1.25	22.09
avg KF	3.72	3.75	6.06	1.65	1.44	17.64
avg KF Vel	3.66	3.73	6.20	1.48	1.53	17.49
avg KF CT	3.54	3.31	5.82	1.84	1.32	16.33
avg KF Vel CT	3.46	3.34	6.44	1.12	1.66	16.68

Table 6.12: Evaluation (Data Set 2); Beacon & AR; Distance to correct point on Ground Truth

The results of the second testing method (Distance to correct point on Ground Truth) can be found in Table 6.12. With this method the Kalman Filters only score slightly better than the raw Beacon signal, which might have to do with the problems of this method (subsection 6.1.4) and also the general raw beacon performance (subsection 6.2.2). It can be noted that again KF CT and KF Vel CT perform a bit better than the other two filters, based on mean and MSE. The highest improvement in MSE is 15% for KF CT

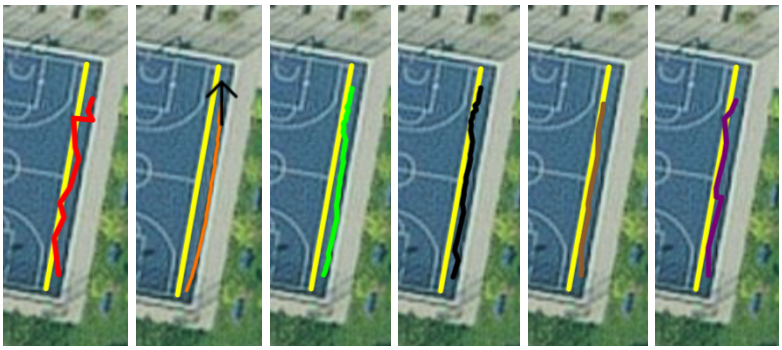


Figure 6.8: 2018-11-15 14:50:04.json; Beacon (red), AR (orange), KF (green), KF Vel (black), KF CT (brown), KF Vel CT (purple)

Figure 6.8 shows a visual representation of the walk *2018-11-15 14:50:04.json*. The red line shows the raw beacon signal that is comparatively jumpy, starts too late and ends too early. It can be seen that the Kalman Filter also start too late, because they only start working with the first beacon signal. Visually its immediately evident that the Kalman Filter perform better as the raw beacon signal and the AR signal. In this example KF CT seems very good, with low jitter aka a smooth line. When looked at the data, KF Vel CT also performed very well. Visually it might be the one with the

most jitter besides the raw beacon signal.

In conclusion it can be said that KF CT and KF Vel CT score best regarding fusion of beacons and Augmented Reality when looked at the data. KF CT visually seems smoother and has less jitter than KF Vel CT, but no clear winner can be called.

6.5 Fusion GPS, Beacons and Augmented Reality

	mean	**median**	**max**	**min**	σ	**MSE**
avg GPS	3.07	1.83	9.97	0.28	2.78	24.43
avg Beacon	1.42	1.22	4.33	0.16	1.11	4.46
avg AR	3.50	3.15	8.58	1.18	2.08	34.07
avg KF	1.70	1.16	5.38	0.16	1.39	8.00
avg KF Vel	1.89	1.12	6.54	0.07	1.66	11.32
avg KF CT	2.64	1.92	6.73	0.48	1.80	22.43
avg KF Vel CT	1.17	0.70	5.13	0.02	1.25	3.92

Table 6.13: Evaluation (Data Set 2); GPS & Beacon & AR; Distance to Ground Truth

To assess the fusion of all three input signals, again Data Set 2 has to be consolidated, because there is no beacon data in Data Set 1. The results of the first method can be found in Table 6.13. As earlier, the good performance of the beacon system can be seen. Unfortunately the values of all Kalman Filters got worse than the values for Fusion of just beacons and AR. It seems like like the bad values of GPS is pulling the Kalman Filter away from the good beacon values. In comparison to GPS the Filters perform still better, with KF Vel CT producing a 84% better result than GPS alone. But in comparison to the beacon system, KF Vel CT is the only Filter which has still better values than beacons alone (3.92m vs 4.46m), which is 12%, but this went up from 2.22m without GPS.

	mean	**median**	**max**	**min**	σ	**MSE**
avg GPS	3.52	3.10	7.13	0.92	1.96	19.89
avg Beacon	3.72	3.27	7.32	1.13	1.89	19.10
avg AR	4.33	4.24	6.90	1.79	1.45	24.65
avg KF	2.93	2.61	5.83	0.44	1.70	14.08
avg KF Vel	2.96	2.71	5.99	0.57	1.70	14.29
avg KF CT	2.80	2.51	5.48	0.72	1.52	13.00
avg KF Vel CT	2.71	2.22	6.26	0.36	1.77	13.76

Table 6.14: Evaluation (Data Set 2); GPS & Beacon & AR; Distance to correct point on Ground Truth

The results of the second method, Distance to correct point on Ground Truth, are found in Table 6.14 is painting a different picture. This time the addition of GPS increases the performance. In the last section using KF CT resulted in a 15% improvement over just using just Beacons. Here KF CT has again the lowest MSE (13m) which is an improvement of 32% over beacons and 35% over GPS. Even KF Vel, which the worst filter in this scenario still improves 25% with GPS instead of just 8% without GPS.

	path length	**recorded length**	**deviation (%)**
avg GPS	25.90	28.80	+9.48
avg Beacon	25.90	30.98	+3.00
avg AR	25.90	26.69	+2.17
avg KF	25.90	36.34	+28.54
avg KF Vel	25.90	38.67	+32.48
avg KF CT	25.90	25.99	+0.01
avg KF Vel CT	25.90	38.97	+31.46

Table 6.15: Evaluation (Data Set 2); GPS & Beacon & AR; Length related data

When looked at the length testing method (Table 6.15) bad performance of 3 out of the 4 filters can be found. KF, KF Vel and KF Vel CT recorded all about 30% more length than the path is actually long. This suggests high jitter in the data. The only Filter that has a good length measure is KF CT, actually only deviating 0.01% (9cm) of the correct length.

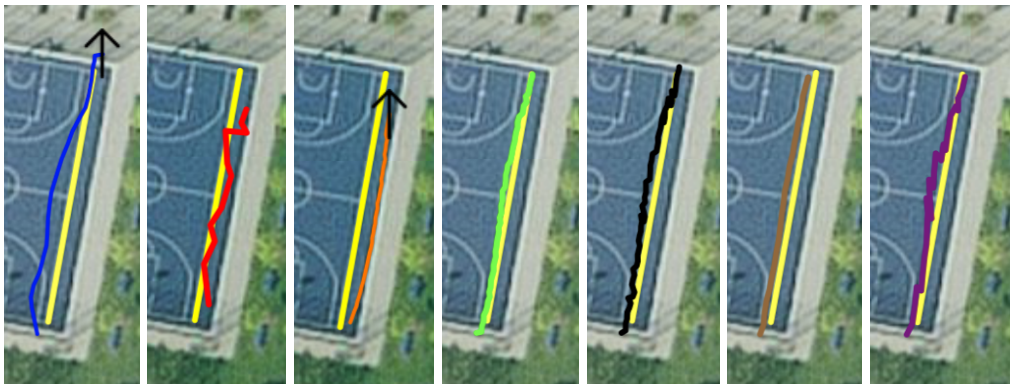


Figure 6.9: 2018-11-15 14:50:04.json; GPS (blue), Beacon (red), AR (orange), KF (green), KF Vel (black), KF CT (brown), KF Vel CT (purple)

A visual representation of the Fusion of GPS, Beacons and AR can be found in Figure 6.9. Again the figure shows walk *2018-11-15 14:50:04.json* as already in the last section. Visually it can be seen again that all four filter perform better than the raw data of GPS, Beacons and AR. The assumption that only KF CT has a smooth signal and all other filters are jittering is definitely confirmed in the images.

6.6 Conclusion

The first conclusion which can be drawn is that the proposed Beacon Trilateration system yields a better performance than GPS. The amount of improvement over GPS depends on the method that is used to test it. The more complicated and conservative Method 2 (subsection 6.1.4) returns only a 4% improvement in MSE. However the simpler Method 1 (subsection 6.1.3) which is less precise, but also less prone to human recording errors, returns an 82% better MSE score for the Beacon system. The data suggests the real improvement lays somewhere in the middle and should be further investigated. With more extensive evaluation, mainly by increasing the walking distance and deploying more beacons, the two methods should converge. More in section 7.3.

6.6.1 Improvement over Raw Data

	GPS & AR	**Beacon & AR**	**GPS & Beacon & AR**
Method 1	+57%; KF	+50%; KF Vel CT	+12% to Beacons; KF Vel CT
Method 2	+24% ; KF Vel CT	+15%; KF CT	+32% to Beacons; KF Vel CT

Table 6.16: MSE Improvement of the best Kalman Filters

After evaluating the data it can be concluded that all four Kalman Filters improve the accuracy in user position over just using GPS or the proposed Beacon Trilateration system. Table 6.16 summarizes the improvement of the best Kalman Filter for each fusion category, split up in Method 1 and Method 2. The results from the simpler Method 1 suggests not to take GPS signals into account when a Beacon system is available. When fusing GPS & Beacons & AR, the improvement over just using Beacons shrinks to 12%, whereas when only using Beacons and AR, an improvement of +50% can be reached. Method 2 is more complicated because it takes the precise user location into account. That should theoretically result in a more accurate error measurement, but practically adds additional error caused by human imperfection. The data collected by Method 2 suggest to always use all available signals. In this case an improvement of 32% to the beacon system is achieved.

6.6.2 Choice of Kalman Filter

	GPS & AR	**Beacon & AR**	**GPS & Beacon & AR**
Method 1	+51%	+50%	+12% to Beacons
Method 2	+24%	+13%	+32% to Beacons

Table 6.17: MSE Improvement of KF Vel CT

When averaging over all testing methods and all fusion concepts, the proposed Kalman Filter *KF Vel CT* reaches the best average scores. Table 6.17 shows the average improvement in MSE of KF Vel CT over the best available raw input source. This table shows an improvement of 50% when Method 1 is used and 32% when Method 2 is used.

As one final measure, the error CDF of Testing Method 1 can be looked at. In Figure 6.10 the GPS error CDF is displayed on the left and the corresponding KF Vel CT CDF is displayed on the right (Fusion of GPS & AR). The x-Axis shows the errors in meter and the y-Axis shows what percentage is below a certain error. So the value of 0.5 on the y-Axis corresponds to the 50 percentile of errors. It can be seen that the KF has generally lower errors throughout and lower maximal errors.

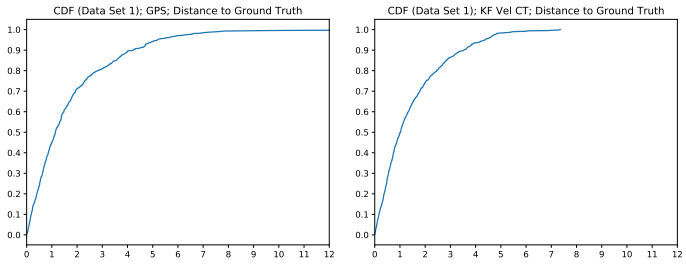


Figure 6.10: CDF of Data Set 1 (Method 1); GPS left; KF Vel CT right

When looking at the CDFs for Fusion of all three inputs on Data Set 2 in Figure 6.11, it can be clearly seen how the KF outperforms GPS and Beacons alone. For the GPS signal (left) 50% of all errors are lower than 1.88m, for Beacons its 1.22m and for KF Vel CT 0.7m

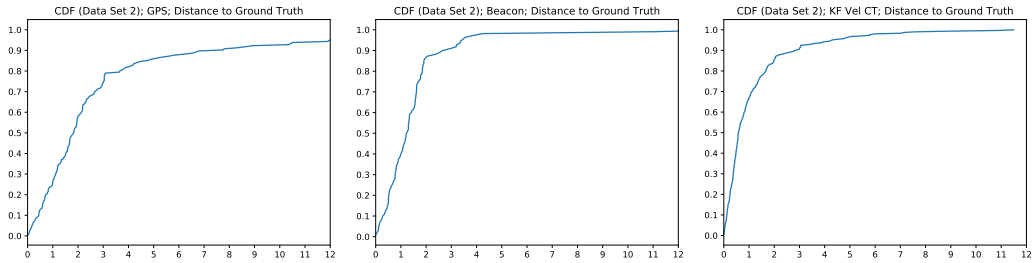


Figure 6.11: CDF of Data Set 2 (Method 1); GPS left; Beacons center; KF Vel CT right

7 Limitations and Future Work

This chapter points out several limitations of this work, as well as potential opportunities on how they could be approached in future work. This chapter is structured in several independent sections that each point out limitations and future work.

7.1 Kalman Filter

This work only considers Kalman Filters as a data fusion method, which limits the system internally to linear models. This results in an higher accuracy positioning data via complex filters (subsection 5.2.11) and a separate simple filter (subsection 5.2.12) that handles orientation based on the magnetometer.

Future projects should look at other algorithms of data fusion, like the Extended Kalman Filter (EKS), the Unscented Kalman Filter (UKS) and the Particle Filter (PF). Those are capable of handling non linear models, which could increase positioning accuracy. Furthermore it makes incorporation of the orientation into the main filter possible. This would open up the option to gain orientation data from motion and perhaps increase orientation estimation overall.

7.2 Bluetooth Beacon Trilateration

The Trilateration algorithm that was chosen in this work is fairly simple but computation efficient (details in subsection 5.2.9). For example it takes only the three closest beacons into account, even when more beacons are sensed.

In further work, a Multilateration system could be used instead, which perhaps increases accuracy.

7.3 Evaluation

The evaluation of this work is fairly extensive and includes four testing methods and two data sets. There are some properties of the data sets which naturally limits the validity of the evaluation only to use cases that have the same properties. Both data sets are recorded outside which results in full GPS coverage. The second data set that has deployed beacons, is comparably small and does not have turns in its path. Furthermore the recording was done in a fairly rigid pose of holding the smartphone.

In future work the evaluation should be extended by looking at more cases like partial GPS coverage, indoor Environments with deployed beacons, longer walking paths, more independent smartphone motion including fast movements and different walking speeds. Additionally a suspicion arose in subsection 6.2.2 that an inherent delay exists in the iPhone Bluetooth framework. In future work this should be further investigated.

7.4 Navigation

The navigational information about route that should be taken is solely based on the Apple MapKit Framework as a routing service and inherits all limitations that come with it. For example, even though the MapKit routes for pedestrians are used, MapKit does not have data about sidewalks. So the coordinates of the routes are all centered in the middle of the street. Furthermore MapKit does not have indoor maps yet.

In future work maps that have more detailed maps could be used, maybe even maps with integrated three dimensional information that helps the positioning problem.

7.5 Augmented Reality Visualization

The way of visualizing navigational information in the proposed work is fairly simple. The path is displayed on the ground as a blue band from one route coordinate to the next. This is the only option provided and besides accurate positioning no additional information about the environment is gathered. Occlusions of objects like tree stems, traffic light poles or other pedestrians are therefore not handled at all.

Alternative options regarding navigational clues could be used in the future, like floating arrows or checkpoints that need to be traversed. Gathering more visual information about the environment is key when occlusions want to be properly handled.

8 Conclusion and Outlook

8.1 Conclusion

This work introduced the idea of AR systems which know their global position and the possibilities this would open up for different location based services. One example is navigation via AR based on precise global position and map based routing data. Chapter 2 explored related work regarding AR navigation, indoor localization and fusion of pedestrian related sensor data. The details of used technologies and algorithms were explained in chapter 3. The AR navigation is approached with four different Kalman filter based algorithms that fuse together information from ARKit, GPS and a custom Bluetooth based positioning system. The later is needed so indoor environments with limited GPS availability are supported as well. Chapter 5 gives an overview of the architecture design and a more detailed look into some implementation details. An extensive evaluation was done in chapter 6 with two main goals. The first was to determine which of the four Kalman filters performs best. This turned out to be the most complicated of the four, called *KF Vel CT*. The filter uses an internal state based on location and velocity and takes the ARKit data to control the inaccurate GPS and Beacon locations. The second goal of the evaluation was to find out how much the accuracy can be improved when using data fusion. This turned out to be an interesting problem. It is not trivial to test how much more accurate than GPS a system is, when GPS is the only available tool to determine global position. To address this, two main testing methods were proposed in combination with a satellite image based ground truth that was carefully constructed. An improvement in accuracy of 32% to 50% over raw signal data was shown. When AR, GPS and the Beacon system are fused together, a median error of only 70cm is reached. Even though this is sufficient for the chosen AR navigation application, there are still some limitations to the work and several things that can potentially be done to increase accuracy even more.

8.2 Outlook

As stated in chapter 7 there are a couple of options to extend and improve the current work. Higher accuracy in this context is always preferable and enables more applications. Nevertheless the user experience of holding a phone in front of you while walking feels somewhat unnatural. This won't change until see-through HMDs become viable.

With the progression in technology and the advent of affordable HMDs, the possibilities seem endless and might have the potential to change the world as for example the smartphone did. With proper AR glasses all traditional screens become unnecessary. A TV can be replaced by a virtual screen displayed in the glasses and artificially pinned onto a real world surface of the users liking. All items in a home that just serve a visual purpose, like photos or posters on the wall can be replaced digital content, visible to everyone wearing the glasses.

This has to be separated from a Virtual Reality (VR) experience. although the fields of AR and VR are related, there is a big difference in user experience. VR places the user into a virtual environment that is generally detached from the real world. That results in setting that is usually sitting in one place for the duration of the experience. AR however can be integrated into a users daily live and enhance the the real-environment without a limitation on time or space.

When global registration for AR experience is thrown in the mix, all kinds of location based services become interesting that could increase the quality of life. One example would be friends sharing their location with each other while being in a crowded situation like a festival. Restaurants could virtually display their menu on the wall of the building, to see for people who walk by. In combination with IOT the scenarios like the following are imaginable. A hospital could track equipment and patients and potentially improve efficiency and therefore save lives.

The prospects of technology like that seem endless, but there are certain risks that also need to be considered. AR systems that are this powerful have the potential to be a major thread to privacy. More accurate localization can also be abused to extend tracking of users for malicious purposes. Because cameras are essential, not just the users become possible targets, but also bystanders that are in the field of view of the camera on coincidence.

List of Figures

List of Tables

Bibliography

[1] P.-H. Diao and N.-J. Shih. "MARINS: A Mobile Smartphone AR System for Pathfinding in a Dark Environment." In: *Sensors* 18.10 (2018), p. 3442.

[2] R. Azuma. "A survey of augmented reality." In: *Presence: Teleoperators and Virtual Environments* 6 (1997), pp. 355–385. ISSN: 10547460. DOI: 10.1.1.30.4999.

[3] T. Höllerer, S. Feiner, T. Terauchi, G. Rashid, and D. Hallaway. "Exploring MARS: developing indoor and outdoor user interfaces to a mobile augmented reality system." In: *Computers & Graphics* 23.6 (1999), pp. 779–785. ISSN: 0097-8493.

[4] W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, M. Reiner, J. Wieghardt, H. Horst, and C. Lindinger. "Pervasive information acquisition for mobile AR-navigation systems." In: *IEEE Mobile Computing Systems and Applications*. IEEE, 2003, p. 13. ISBN: 0769519954.

[5] W. Narzt, G. Pomberger, A. Ferscha, D. Kolb, R. Müller, J. Wieghardt, H. Hörtner, and C. Lindinger. "Augmented reality navigation systems." In: *Universal Access in the Information Society* 4.3 (2006), pp. 177–187. ISSN: 1615-5289.

[6] C. G. Low and Y. Lee. "Interactive virtual indoor navigation system using visual recognition and pedestrian dead reckoning techniques." In: *International Journal of Software Engineering and its Applications* 9.8 (2015), pp. 15–24. ISSN: 1738-9984.

[7] M. S. Grewal, L. R. Weill, and A. P. Andrews. *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons, 2007. ISBN: 0470099712.

[8] P. D. Groves. "Principles of GNSS." In: *Inertial, and Multisensor Integrated Navigation Systems* 521 (2008).

[9] R. E. Kalman. "A new approach to linear filtering and prediction problems." In: *Journal of basic Engineering* 82.1 (1960), pp. 35–45. ISSN: 0021-9223.

[10] J. Bird and D. Arden. "Indoor navigation with foot-mounted strapdown inertial navigation and magnetic sensors [emerging opportunities for localization and tracking]." In: *IEEE Wireless Communications* 18.2 (2011), pp. 28–35. ISSN: 1536-1284.

[11] E. Foxlin. "Pedestrian tracking with shoe-mounted inertial sensors." In: *IEEE Computer graphics and applications* 25.6 (2005), pp. 38–46. ISSN: 0272-1716.

[12] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara. "A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU." In: *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*. IEEE, 2009, pp. 37–42. ISBN: 1424450578.

[13] A. R. Jiménez, F. Seco, J. C. Prieto, and J. Guevara. "Indoor pedestrian navigation using an INS/EKF framework for yaw drift reduction and a foot-mounted IMU." In: *Positioning Navigation and Communication (WPNC), 2010 7th Workshop on*. IEEE, 2010, pp. 135–143. ISBN: 1424471583.

[14] C. Hide, T. Botterill, and M. Andreotti. "Low cost vision-aided IMU for pedestrian navigation." In: *Ubiquitous Positioning Indoor Navigation and Location Based Service (UPINLBS), 2010*. IEEE, 2010, pp. 1–7. ISBN: 1424478790.

[15] P. Ivanov, M. Raitoharju, and R. Piché. "Kalman-type filters and smoothers for pedestrian dead reckoning." In: *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2018, pp. 206–212. ISBN: 1538656353.

[16] P. Bahl and V. N. Padmanabhan. "RADAR: An in-building RF-based user location and tracking system." In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 2. Ieee, 2000, pp. 775–784. ISBN: 0780358805.

[17] A. Boukerche, H. A. B. F. Oliveira, E. F. Nakamura, and A. A. F. Loureiro. "Localization systems for wireless sensor networks." In: *IEEE wireless Communications* 14.6 (2007). ISSN: 1536-1284.

[18] F. Subhan, H. Hasbullah, A. Rozyyev, and S. T. Bakhsh. "Indoor positioning in bluetooth networks using fingerprinting and lateration approach." In: *Information Science and Applications (ICISA), 2011 International Conference on*. IEEE, 2011, pp. 1–9. ISBN: 1424492246.

[19] J. Röbesaat, P. Zhang, M. Abdelaal, and O. Theel. "An improved BLE indoor localization with Kalman-Based fusion: an experimental study." In: *Sensors* 17.5 (2017), p. 951.

[20] *ARKit | Apple Developer Documentation*. `https://developer.apple.com/documentation/arkit`. Accessed: 2018-12-03.

[21] *Understanding World Tracking in ARKit | Apple Developer Documentation*. `https://developer.apple.com/documentation/arkit/understanding_world_tracking_in_arkit`. Accessed: 2018-12-03.

[22] *Short Range iBeacon - Avvel International*. `https://www.avvel.co.uk/shop/short-range-ibeacon-1`. Accessed: 2018-12-03.

[23] *Map and Tile Coordinates | Maps JavaScript API | Google Developers*. `https://developers.google.com/maps/documentation/javascript/coordinates`. Accessed: 2018-12-11.

[24] C. C. Robusto. "The cosine-haversine formula." In: *The American Mathematical Monthly* 64.1 (1957), pp. 38–40. ISSN: 0002-9890.

[25] T. Vincenty. "Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations." In: *Survey review* 23.176 (1975), pp. 88–93. ISSN: 0039-6265.