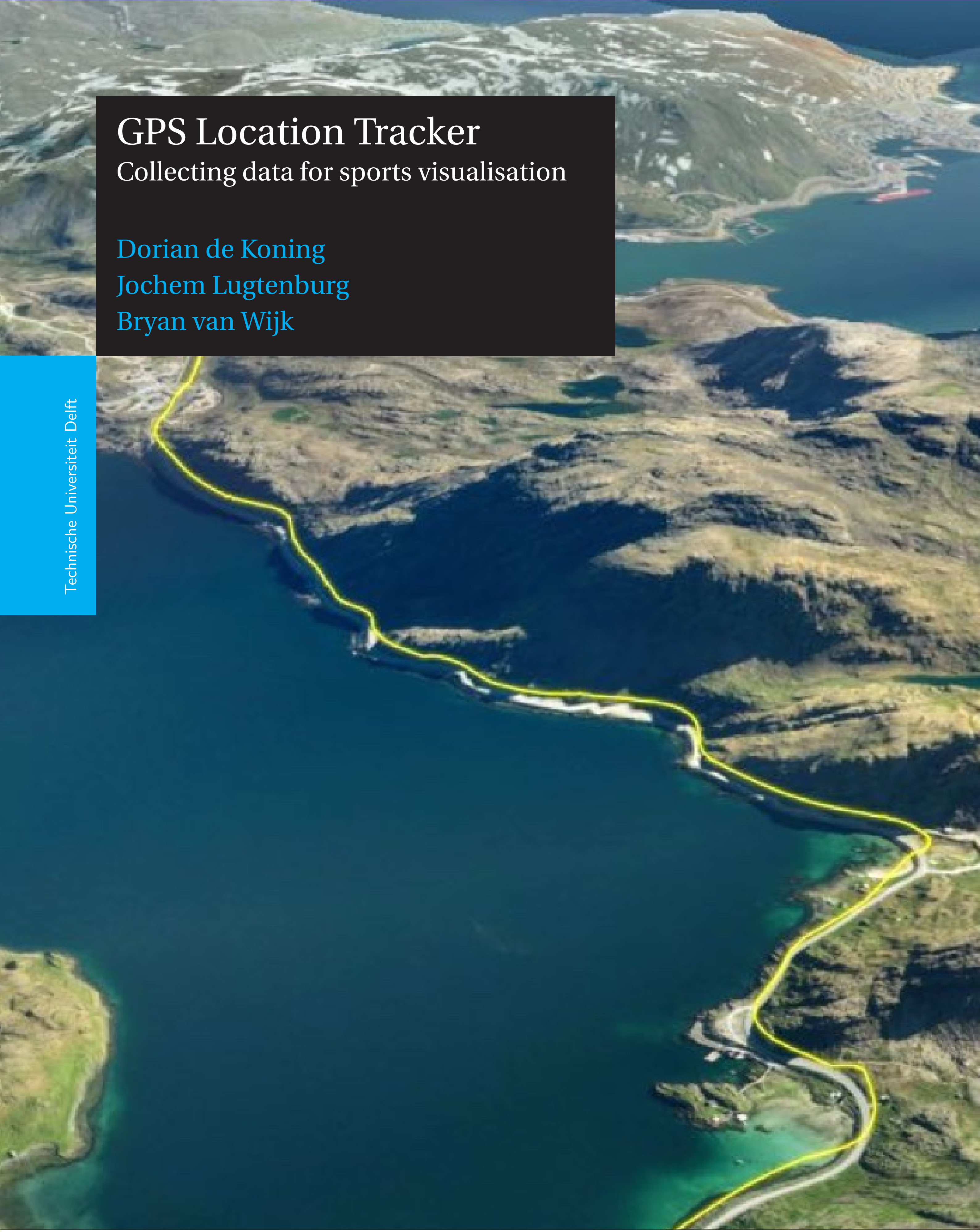


GPS Location Tracker

Collecting data for sports visualisation

Dorian de Koning
Jochem Lugtenburg
Bryan van Wijk

GPS Location Tracker

Collecting data for sports visualisation

by

Dorian de Koning
Jochem Lugtenburg
Bryan van Wijk

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science

Supervisor:	Dr. A. M. Zúñiga Zamalloa	
Thesis committee:	Dr. A. M. Zúñiga Zamalloa	TU Delft
	Dr. H. Wang	TU Delft
	Ir. R. Steen	Relive B.V.

PREFACE

As a team, we have spent the final ten weeks of the TU Delft bachelor at Relive B.V. The project proposed by Relive was a challenge in which we were able to show the skills and methods learned in the various courses offered over a period of three years.

It was exciting to integrate our implementation with the implementation already present at Relive. All employees of Relive provided great support, ranging from graphic design to technical issues. This resulted in the data obtained by our implementation to quickly and reliably produce high-quality videos.

There are a number of people we would like to thank. First, the person representing the interests of Relive, Ronald Steen. Ronald helped us connect our application to the database and render engine of Relive. We are also expressing our gratitude to the other Relive team members Yousef, Ralf, Lex, and Joris. Second, we would like to thank Marco Zúñiga for guiding us trough the project. Having a busy schedule, we greatly appreciate the time he reserved for us. Finally, we were very pleased with all the people willing to test our application and providing us with the large amounts of data we needed.

We hope that you, the reader, will be just as excited by the project as we are.

Dorian de Koning
Jochem Lugtenburg
Bryan van Wijk
Rotterdam, July 2017

SUMMARY

The start-up Relive creates videos which users can watch to experience their running or cycling activity all over again. Currently, the company depends on external data sources to generate a video. To be less dependent on these sources Relive wants to create their own tracking solution. This solution has to fit in their existing smartphone application available for iOS and Android. The company wants to remain flexible, therefore the tracking application has to be developed in such a way that it can also be used in other products the company might develop in the future. As a goal, the data has to result in visually pleasing videos for a large user base.

Based on an experimental app developed during the research phase, raw smartphone GPS data was found to be unsuitable for video rendering. To improve this data, a Kalman Filter is used, in combination with a smoothing algorithm. The system has been designed to allow code sharing between iOS and Android where possible. The system has been implemented in Objective-C, Java, and TypeScript. Separating the system in three blocks enables code reuse which improves maintainability of the system. The filter has been integrated as shared code in the TypeScript implementation, which allows filtering to happen on the device. The user of the React Native Module developed has freedom to retrieve the unprocessed and processed data.

The system has been tested by means of unit tests in all three programming languages used. Tests have been executed using a continuous integration server, testing each pull request against the current code base to ensure quality. Part of the testing phase includes the React Native Module to be implemented in the client's smartphone application to demonstrate its use. The application has been sent to a number of test participants to collect data from different routes and activities. The project can be seen as a success since all important requirements have been successfully implemented.

Contents

1 Introduction

During the final quarter of the Computer Science bachelor, a team of three students has worked on the development of a sports activity tracking solution for the Dutch start-up Relive. Relive's main goal is allowing a person to re-experience their activity by means of a 3D video. Currently, Relive depends on external sources to gather location data. Developing a reliable tracking solution, compatible with an existing smartphone application is an exciting challenge. In this document, the design, implementation, and results of the TI3806 Bachelor Project course given at the TU Delft faculty of Electrical Engineering, Mathematics and Computer Science are described.

First, Chapter two gives an overview of the research phase of the project. The research phase covers the first two weeks of the project. It contains the project description as discussed with the client. Existing solutions and alternatives are discussed. Finally, an overview of technologies to be used is given.

Second, Chapter three covers the system design. It contains an overview of requirements derived from the research phase and discussions with the client. Also, an overview of the overall system architecture is given. An important part of the system, data processing and filtering, is described.

Third, Chapter four contains implementation details of the developed solution. A global overview of the different implementations in three different programming languages is given. Also, details of important components and features are discussed. The chapter concludes with a use case in which the developed solution is applied.

Fourth, Chapter five contains an analysis of the quality of the system. Both test methodologies and results are described. Also, the evaluation retrieved from the Software Improvement Group (SIG), is briefly discussed.

Fifth, Chapter six describes the process that has been followed during the project. It covers software engineering methodologies such as proper usage of version control, scrum and organizational details of the project.

The report ends with a conclusion, a reflection on the course of the process and recommendations to the client.

2 RESEARCH

2.1. INTRODUCTION

This chapter contains the results of the initial research phase of the project. During the project, the team will develop a sports location tracking feature. First, an introduction of the project will be given, introducing the reader to the client and their problem. Second, existing solutions are listed. Third, a field test was conducted to determine mobile phone location accuracy. Whereafter, possible solutions to accuracy problems discovered during the field test are discussed. The report concludes with an overview of technologies to be used during the implementation phase of the project.

2.2. PROJECT DESCRIPTION

The client has provided the team with a problem definition during meetings in the weeks before the start of the project. This section summarizes this problem definition.

2.2.1. CLIENT DESCRIPTION

To make the reader familiar with the company, a short description of Relive has been provided by the client.

> Relive turns outdoor adventures into short video animations that people love to share with family and friends. Relive is all about the total experience of your trip. No focus on performance metrics and competition. Let's leave that to traditional sports trackers.
>
> Since April Relive has been growing extremely fast. At the moment, we're over 700,000 users, including Laurens ten Dam, Lance Armstrong, and Instagram founder Kevin Systrom; great people. Each month we're processing more than 1 million videos.
>
> Right now, exciting new steps in our product development are being made. The ideas for future releases are endless. It's great. This is where you come in.

2.2.2. PROJECT DEFINITION

Relive collects GPS data from third party trackers such as Strava[1] and Garmin Connect[2]. Recently Relive has released a mobile application, providing easy access to previously generated video animations. It would be convenient for Relive to have a tracking feature within the Relive app making the company less dependent on other companies for data collection. This tracking feature should be reliable, in the sense that it should be accurate and be able to deal with different forms of failure such as device shutdown, app crash, and an unreliable GPS signal. It should also be flexible, in the sense that it works under different conditions and sports activities. Also important are aspects such as battery usage, sensing interesting moments during an activity and the overall user experience. The team should investigate existing solutions and decide if existing components can be used, or new smarter solutions have to be developed. On a regular basis, the team should discuss intermediate findings with Relive and come up with a technical design of the product that is realistic to develop during the project.

[1] Strava Inc, `http://www.strava.com`
[2] Garmin Ltd, `https://connect.garmin.com`

2.2.3. Project Goal
The development of a mobile GPS tracking feature compatible with the Relive app on iOS and Android.

2.2.4. Involved Stakeholders
Relive
Ronald Steen will represent the interests of Relive during the project. He will represent the client within the Bachelor Committee. The main interest of Relive is having a working GPS tracking solution that can be deployed within the existing Relive Application. This solution has to be proven to be robust and reliable to be used in a production environment.

Delft University of Technology
Marco Zúñiga represents the educational interests of the TU Delft. Representing TU Delft within the Bachelor Committee in the role of TU Coach he will act as a guardian of the educational interests of the university. He acts as a counterweight to the interests of the client. He is also responsible for giving feedback to the students on the research report, final report and final presentations.

Students
Dorian de Koning, Jochem Lugtenburg, and Bryan van Wijk are the team executing the project. Their interest is successful completion of the project. To achieve this, the interests of both Relive and TU Delft have to be met. For the team, this is a fun and interesting way to apply knowledge gathered during the Bachelor program.

2.3. Existing Solutions
In the current situation Relive depends on the collection of GPS data by different third parties. The downside of using these third parties to get the GPS data is the possible chance of a third party changing the implementation. When this happens, Relive should adapt to these changes in implementation. Using an own app to collect the GPS data will give much more flexibility. However, looking at other apps could be a useful inspiration about what is possible. Several open source GPS sports trackers exist. However, most of these connect to dedicated hardware GPS trackers which are not suitable for this project. No viable open source solutions where found that use the GPS in mobile devices for the tracking.

2.3.1. Smartphone applications
Currently, almost all smartphones have a GPS module inside and have the possibility to determine the location of the user reporting longitude, latitude and in some cases even altitude. This feature can be used to track sports activities such as going for a run or cycling. A lot of different GPS tracking apps already exist which are all able to do this. Most of them have the possibility to select the sport the user is going to perform and then start the measurement with single button press. After finishing your workout, the user has to stop the workout in the app to save the route. The most important difference between tracking of different sports is the time between two saved locations. For example, during a cycling activity, the movement is much faster than while running and it is required to have a location update more often to accurately follow the route that has been traveled. A useful feature some of the tracking apps have is automatically pausing an activity when taking a break or automatically detect that an activity has been started.

2.3.2. Dedicated hardware GPS trackers
Instead of using a smartphone for collecting location data it is also possible to use a special hardware device. The downside of this is that the user needs to buy a new expensive device to track his sports activities. Another downside is that you need to carry another device with you even when you already have a smartphone with you.

2.3.3. Existing solutions comparison
The already existing solutions for GPS tracking are not all the same. To be able to compare what these solutions track and how well they approximate the real traveled path a field test is executed. In this test, the same path is traveled multiple times with different apps on a bike. The used apps are Endomondo[3], Fitbit[4],

[3]Endomondo, http://www.endomondo.com
[4]Fitbit, http://www.fitbit.com

MapMyRun[5], Runkeeper[6], Runtastic[7] and Strava[8]. First, all these apps are used by selecting the option to track a running activity. The speed during the test was measured with a simple bike computer keeping the velocity between 15-16 km/h. For all these tests the same Samsung Galaxy S8+ with Android 7.0 is used. The smartphone is mounted on the bike to have the best range. The results are visualized in figure D.1. In table 2.1 multiple measurements based on the GPS locations collected with the different applications are shown. The distance is calculated by summing up the individual great circle distances between every consecutive point. The average distance is the average great-circle distance, the shortest distance between two points on the surface of a sphere, between two consecutive points. Decimal places refer to the number of digits the latitude and longitude coordinates have after the decimal point.

Application	**Average distance (m)**	**Sample interval (s)**	**Distance (m)**	**Samples**	**Decimal places**
Endomondo	20.39	5.08	713.7	36	6
Fitbit	3.852	1.03	693.4	181	6
MapMyRun	12.44	2.90	759.4	62	10
Runkeeper	11.56	2.79	751.6	66	6
Runtastic	15.40	3.71	739.1	49	16
Strava	4.183	1.02	744.5	179	6

Table 2.1: Overview existing apps with measurements of a running activity.

The second test is executed in the same way as the previous test but this time tracking a cycling activity. In this test, the speed is kept between 26-27 km/h. In figure D.2 the results are shown on a map. In table 2.2 the measurements for this test are displayed.

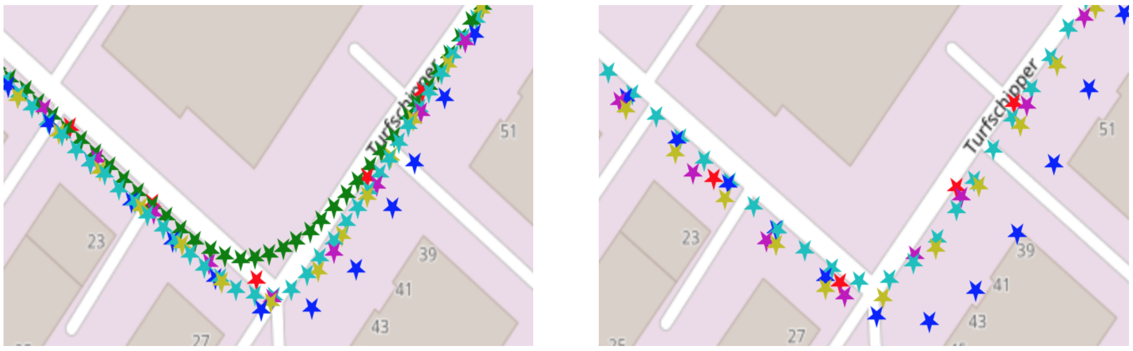
Application	**Average distance (m)**	**Sample interval (s)**	**Distance (m)**	**Samples**	**Decimal places**
Endomondo	28.24	4.19	705.9	26	6
MapMyRun	15.05	2.02	782.6	53	10
Runkeeper	14.34	1.98	745.9	53	6
Runtastic	19.59	2.77	744.2	39	16
Strava	7.297	1.02	737.0	102	7

Table 2.2: Overview existing apps with measurements of a cycling activity.

From this test, we can conclude the following: although the same smartphone is used, different apps result in different tracked paths. The most noticeable difference is the sample rate. In the running test, the highest sample interval is 5.08 seconds between the measurements while at the lowest sample interval this is 1.02 seconds. With a lower sample interval, it is possible to approximate the original route more precisely. Another important difference is the traveled path as shown in figure 2.1. The path marked in green and blue are noticeably different than the other paths. This was not an incidental deviation, but when repeated the same route with these apps the same deviation was visible. The green marked route is not visible for cycling because this app has no cycle tracking feature. The difference between tracking a running or cycling activity is for almost all apps visible in the sample rate. As expected almost all applications increased the sample rate during a cycling activity. The maximum sample interval is for both activities not lower than 1 second between two samples. The number of decimal places of the longitude and latitude returned by the different apps is also different. The highest precision is 16 decimal places and the lowest is 6 decimal places. 6 decimal places correspond already to a precision of approximately 10 centimeters.

Although there are a lot of solutions available to collect location data, developing a feature to be used in the currently available app will have a lot of benefits. The most important one is the flexibility, with an own app you can easily change measures collected or make sure there is a more efficient way of collecting data. It also opens up possibilities to introduce new features specifically useful for creating the videos. New features

[5]MapMyRun, http://www.mapmyrun.com
[6]Runkeeper, http://www.runkeeper.com
[7]Runtastic, http://www.runtastic.com
[8]Strava Inc, http://www.strava.com



(a) Running zoomed

(b) Cycling zoomed

Figure 2.1: Same travelled path with approximately 15-16 km/h (running) and 26-27 km/h (cycling) using different apps. All paths were traversed on a bike to make it more easy to keep the speed constant. ● Endomondo, ● Fitbit, ● MapMyRun, ● Runkeeper, ● Runtastic, ● Strava

could be taking pictures at automatically detected exciting moments or creating a video when you are going down a hill with a high speed. It will also be possible to easily extend the feature with other activities such as paragliding, riding a motorcycle or sailing.

2.4. GPS ACCURACY

To create aesthetically pleasing videos the location of the user should be predicted as accurate as possible while having as little impact on battery life as possible.

When starting the research phase the team had little knowledge about the challenges that could arise when using mobile devices for GPS tracking. Before creating a tracking application, it is important to know the limits and possibilities of the devices used for tracking. Today's phones contain a wide range of available sensors [1]. For sports tracking, sensors such as GPS, accelerometer, compass, and gyroscope can be of great value. Since GPS is the main contributor to location tracking, measuring its accuracy greatly determines the actual implementation of a tracking feature. With a simple test, the team wants to gain insight into the accuracy of the GPS functionality of mobile devices and the differences in accuracy between different mobile devices.

2.4.1. RESEARCH APPROACH

The general idea of the test is to choose a set of easily recognizable objects on Google Maps[9] for which the actual location can be determined with relatively large accuracy. Different phones will be held in these locations where 10 GPS data samples will be requested and stored in the memory of the phone. This data can then be used to compare accuracy in different locations, across different phones and different operating systems. For the test 4 locations where selected. Two state of the art Android and iOS phones and an older Android and iOS phone will be used.

TEST APPLICATION

To collect the GPS data a simple app will be developed, this app requests the location of the device at a regular interval. The interval was set to 2 seconds for this test. The app requests the current longitude, latitude, altitude, time and accuracy from the operating system. The app contains a start stop button to begin and end the sampling of data. When the stop button is pressed the data is stored in JSON format in the internal memory of the phone. Developing this application will also provide the team with more insight into the technological challenges that could arise during the project.

[9]Google Maps,http://maps.google.com

2.4.2. Research results

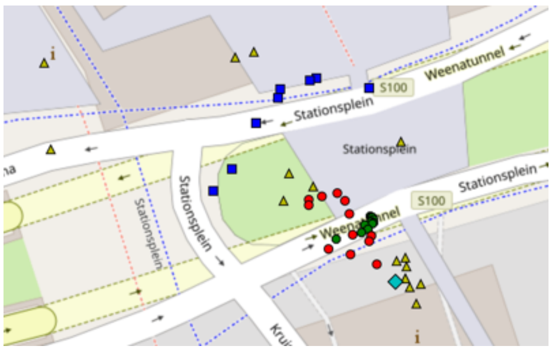
The results of the test can be seen drawn on a map in 2.2. For all the collected samples the distance to the actual location was calculated and these distances are plotted in a boxplot in figure D.3a. In table 2.3 the values used to create the boxplots are shown.

2.4.3. Discussion

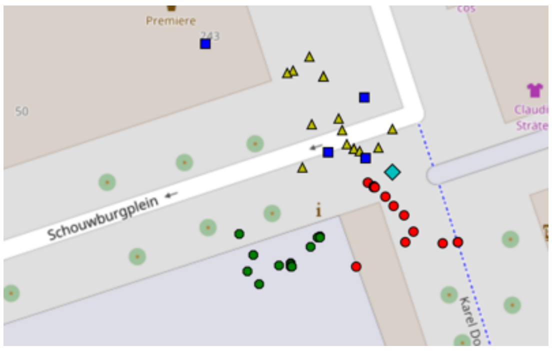
One of the main points that became clear during the development of the test app are the differences between the location services on Android and iOS. On Android, the location is purely based on GPS and can be requested whereas on iOS the location is also based on other factors such as Wi-Fi and mobile networks. Some of the users will be in remote areas without access to these services, a location like this was not available during this study. Therefore further research has to be conducted on the accuracy of the GPS location provided by iOS in remote areas. Another uncertainty is the fact whether the Android and iOS operating systems already process the GPS data in some way.
This study looked into the accuracy of stationary devices instead of moving devices which will be used in practice. The reason for this is the fact that it is a lot easier to compare data from stationary devices than from moving devices. The accuracy of stationary versus moving devices should not matter when purely using GPS data. GPS satellites are at about 20.000 km above the surface of the earth and travel twice around the earth per day [2]. This means they at a speed of roughly 3.500 km/h relative to the surface of the earth, therefore, speeds below 100 km/h for our application will be negligible.

2.4.4. Conclusion

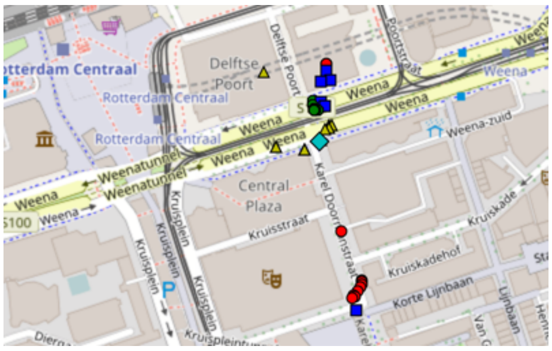
From the test results, we can conclude multiple things. First, from the GPS samples drawn on the map we can see that for the devices where the points lie relatively close to each other there seems to be a systematic error. This is especially the case for the XT1092 and S8+. The points that are more scattered seem to be scattered



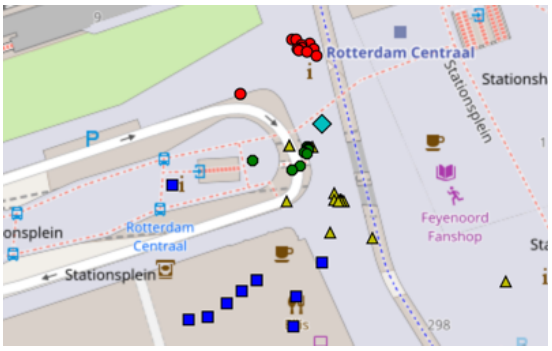
(a) Location 1: lat. 51.922787, lon. 4.470553



(b) Location 2: lat. 51.921666, lon. 4.474038



(c) Location 3: lat. 51.923327, lon. 4.473385



(d) Location 4: lat. 51.923998, lon. 4.469033

Figure 2.2: ● Reference position, ● Samsung Galaxy S8+, ● Motorola XT1092, ● Apple iPhone 5, ● Apple iPhone 6

	name	min	lower quartile	mean	upper quartile	max	standard deviation
Position 1	S8+	4.27	8.65	14.24	18.94	20.03	5.43
	XT1092	9.96	10.04	10.71	11.60	12.31	0.91
	iphone 5	32.92	32.92	35.52	36.62	37.75	1.79
	iphone 6	3.27	4.34	14.10	25.56	46.31	16.21
Position 2	S8+	4.11	7.05	12.53	15.15	15.91	4.22
	XT1092	17.43	21.74	22.52	26.00	27.62	3.04
	iphone 5	4.73	4.73	10.67	12.51	296.63	86.13
	iphone 6	7.14	7.14	18.56	22.42	22.81	6.22
Position 3	S8+	83.62	83.62	162.72	171.28	173.69	39.88
	XT1092	33.55	33.66	34.33	40.34	44.49	4.20
	iphone 5	38.44	38.44	41.39	66.74	187.03	43.35
	iphone 6	2.14	19.33	19.33	47.44	95.65	26.54
Position 4	S8+	20.20	21.24	22.63	23.60	23.87	1.28
	XT1092	7.46	8.07	11.12	15.11	21.27	5.04
	iphone 5	37.55	47.50	58.19	64.48	301.81	99.75
	iphone 6	6.87	10.77	21.33	29.60	65.62	16.41

Table 2.3: Test results

one direction away from the actual point instead of randomly distributed around the actual point. Finally, there are some erroneous samples that are more than 100 meters away from the actual location, however, these will be easily detected and removed in practice.
The boxplots show that an error larger than 10 meters is not uncommon. Since errors like this will be visible in the created videos, for example when the user is cycling over a narrow road. Therefore measures should be taken to increase the accuracy of determining the location of the user.

2.5. GPS Accuracy Solutions

During the field test, we've discovered accuracy is a large issue while developing a tracking application. In this section, we discuss some possible solutions to solve this problem. It is important that the team overcomes low accuracy in order to build a robust tracking feature.

2.5.1. Filters

Due to the observed variation in measured position per phone, a method to filter out this variation is required. Kalman filters[3] can be used to give an estimation of the correct position. Kalman filters exist in various forms such as the extended, adaptive and unscented Kalman filters [4, 5]. During the development of a feature tracking application, the Kalman filter can be used to give a more accurate prediction of the user's actual position. If this estimation is accurate enough, fewer GPS locations are required which has a positive influence on battery preservation.

2.5.2. Map data

Map data available from sources such as Open Streetmap[10] can be used to map a measured trajectory to a road grid [6, 7]. Methods described in [6] use cellular data combined with smartphone sensor data to map a path followed to a road grid making use of a method using Hidden Markov Models to identify road segments which have been followed with a high likelihood [8]. A three step map-matching method using greedy algorithms is described in [7]. Mapping a GPS trajectory to a road grid increases the accuracy of the measurements for cases in which the user has been moving over a road. Since users might be tracking their locations in off-road areas such as forests, using map data will not always increase accuracy.

2.5.3. Image recognition

If the user is not close to roads for instance when cycling trough a meadow, image recognition may be an option to determine if there is a path. Using image segmentation techniques on satellite imaging, paths and roads can be extracted [9, 10] and compared to the GPS data. The team expects that image recognition will

[10]Open Streetmap, `http://planet.openstreetmap.org`

not be required since a high amount of map data is already available, some of which is already based on image recognition. Image recognition might be a solution for extreme corner cases in which no data is available.

2.5.4. CROWDSOURCING

Crowdsourcing is the concept of using a random large group of people to collect data which can be used to solve a problem[11]. Is mainly driven by the increasing quality and speed of the Internet which makes it possible to collect and share really fast large amounts of data. A device that can be used for this is the smartphone. The smartphone connected to the Internet and has a high number of sensors already in it which makes it easy to collect large amounts of data[12]. Advantages of crowdsourcing are the low costs, high scalability and great flexibility in the sort of data that must be collected.

Crowdsourcing makes a large data set containing all combined GPS tracks available which could be very useful. Algorithms as described in [13] [14] can be used to interfere travel paths from this data. This gives the possibility to find the correct paths traveled with a high likelihood. It can also be used to improve the path estimation of further tracking uses. For example, consider someone is tracking his sports activity but the route that best estimates all the routes is a few meters to the left. There is a high chance the path currently tracked is also on that same average route, this way it is possible even with a bad GPS signal to estimate the correct path. One of the problems with the use of these algorithms is they do not take into account time. This means when you cross an already visited path this will influence the result. A benefit of these algorithms is the decreased complexity and no influence on the result of different movement speeds.

The test application as described earlier is used to track a simple path multiple times. This data together can be used to estimate the real path traveled better than a single track. Even with a rather simple algorithm, a pretty good result can be achieved. First, the data of the different tracks is linearly interpolated over time to have the same amount of location points on every track. Finally, for every time stamp, the average of the corresponding location is taken to get the average path estimation. This method will only work when the movement speed during every track is the same, more research is necessary to solve this problem. When this is not the case the averaged positions are not approximately the same. The result can be found in figure D.4. This result will be improved when there is more data available.

2.6. CHOICE OF TECHNOLOGIES

This section contains an overview of the technologies that will be used during the execution of the project. Most of these technologies have been evaluated during the creation of a small application which gathered test data for the comparison. This evaluation makes sure all team members are familiar with the used technologies, preventing problems in the next phase of the project. The tracking feature should be compatible with the existing app of the client, but should not necessarily be a separate application. The existing application of the client is available for both the iOS and Android ecosystems. Hence, the technologies implementing the tracking feature should take both ecosystems into account.

2.6.1. APPLICATION DEVELOPMENT

Since the tracking feature requires interaction with both iOS and Android ecosystem, the choice of frameworks used highly impacts the maintainability of the final product. Ideally, the code can be shared between iOS and Android which reduces the workload in both maintaining and extending the implementation.

REACT NATIVE

Currently, the Relive application has been built using the React Native[11] framework which is an open source framework actively maintained and used by Facebook. The React Native framework provides a shared JavaScript layer between the iOS and Android ecosystems, promoting code reuse and thus maintainability. React Native was used as the basis for the test data gathering application used in the research. GPS data collection using React Native appeared to be challenging because the JavaScript geolocation implementation of React Native is fairly limited. This requires the team to develop a native implementation for geolocation for both iOS and Android.

[11] React Native, https://github.com/facebook/react-native

iOS Development
Since native implementations for iOS will be required, developer tools supplied by Apple will be used. Apple provides a framework dealing with user's location. Core Location[12] provides a large set of methods and classes providing information about the location of the mobile device.

Android Development
Native implementations in Android will be performed using the Android SDK. Both the Android SDK and Google provide API's dealing with location. The Android SDK provides android.location[13] which provides access to location services on a lower level. The Google API[14] provides more high level (simpler) access to location services.

2.6.2. Test Frameworks
Jest
In order to test a React Native implementation the Jest[15] testing framework can be used. Unit tests are written in JavaScript and can be executed automatically. During the creation of the test application the team discovered that if React Native depends on the non-javascript native code, mocking is required. Jest provides a powerful mocking library to deal with these cases. Code coverage reports are automatically generated, providing the team with valuable insights.

MochaJS
An alternative to Jest is MochaJS[16]. While MochaJS is more mature, it does not natively support development using React Native and requires more configuration. Because Jest natively supports React Native, the team prefers Jest over MochaJS.

JUnit
Unit testing in Java will be performed using the JUnit[17] framework. The team is already familiar with this framework, making it a suitable option for testing.

XCTest
Tests for iOS will be conducted using the XCTest framework provided by Apple [18].

2.6.3. Languages
TypeScript
Due to the lack of types in JavaScript, type related errors are highly likely to occur. TypeScript[19] prevents these errors by providing a typed superset which compiles to JavaScript. TypeScript will generate errors if incorrect types preventing a large range of issues. TypeScript is open source and is actively maintained by Microsoft.

Java
The official language for programming Android applications is Java. The team will use Java to develop native Android implementations.

Swift and Objective-C
Native development for iOS can be done in various languages, with Swift and Objective-C being the languages used by the Apple iOS Development APIs. Swift is less complex, open source and more easy to read. This means that Swift is also more easy to maintain. Swift and Objective-C can be used interchangeably using header files.

[12]Core Location, https://developer.apple.com/reference/corelocation
[13]android.location, https://developer.android.com/reference/android/location/package-summary.html
[14]com.google.android.gms.location, https://developers.google.com/android/reference/com/google/android/gms/location/package-summary
[15]Jest, https://github.com/facebook/jest
[16]MochaJS, https://mochajs.org
[17]JUnit, http://junit.org
[18]XCTest, https://developer.apple.com/reference/xctest
[19]TypeScript, https://www.typescriptlang.org/

2.6.4. SOFTWARE ENGINEERING ASSISTANCE

GITHUB

Good software engineering habits dictate the use of version control systems. The team will use GitHub[20] to keep track of different versions of the implementation.

TRAVIS CI

To stimulate well tested and clearly written code, continuous integration using Travis CI[21] can be used to test each pull request against the master branch of the repository. Travis will check if the code written compiles correctly, passes all test cases and has a correct coding style enforced by static analysis. A merge into master will not be allowed if the continuous integration server fails.

TSLINT

In order to make sure that the TypeScript code readable, maintainable and functions as intended, static analysis is performed using TSLint[22]. TSLint produces errors if the supplied rules are not followed causing a build error on the continuous integration server thus enforcing good coding habits.

2.6.5. TECHNOLOGY OVERVIEW

Concluding this section, we will provide an overview of technologies to be used. Since the clients existing Application uses React Native and the tracking feature should be integrated into this application, it makes sense to use React Native as the main framework to share code between platforms. Since React Native has limitations, native iOS and Android extensions have to be developed. The team will use Jest, JUnit, and XCTest to verify the reliability of the implementation. Java will be used for Android development. For iOS development, the team prefers usage of Swift, as it is more easy to maintain. If parts of the code cannot be implemented using Swift, the team will use Objective-C for these cases. Table z2.4 gives an overview of the technology choices to be taken into account in the next phase of the project and lists the location where documentation can be found.

Application Development	**Documentation**
React Native	http://facebook.github.io/react-native
iOS	https://developer.apple.com/reference
Android	https://developer.android.com/reference
Test Frameworks	
Jest	http://facebook.github.io/jest
JUnit	http://junit.org
XCTest	https://developer.apple.com/reference/xctest
Languages	
TypeScript	https://www.typescriptlang.org
Java	https://docs.oracle.com/en/java/
Swift/Objective-C	https://swift.org/documentation
Software Engineering	
Travis CI	https://docs.travis-ci.com
TSLint	https://palantir.github.io/tslint
GitHub	https://help.github.com

Table 2.4: Overview of technologies to be used

[20]GitHub, http://www.github.com
[21]Travis CI, https://travis-ci.com
[22]TSLint, https://palantir.github.io/tslint/

3 PRODUCT DESIGN

This chapter contains the overall system design. It starts with requirements derived from the research phase and discussions with the client. Then the overall architecture of the system is described. Finally an important part of the system, the Kalman Filter algorithm is explained.

3.1. REQUIREMENTS

The final product has to fulfill a set of different requirements. To prioritize these requirements, the MoSCoW method was used. This method divides the list of requirements in must, should, could and won't haves. This division helps the team focus on what is important at a given moment throughout the whole project. The MoSCoW method was chosen because the team is familiar with this method since it was used in multiple projects throughout the bachelor.

3.1.1. MUST HAVES

The requirements listed in Table 3.1 are essential for the success of the project.

Requirement
The implementation must be usable on Android SDK version 19 and iOS version 9.
The data gathered by the implementation must be compatible with the current video rendering software of the client.
The tracking feature must work when the app is not running in the foreground.
Data should not be lost after the app is closed or crashed specifically during recording.
The user must be able to start and stop tracking.
The data must automatically be sent to a server of the client.
The tracking functionality must gather longitude, latitude, altitude and time data.
The applications should be able to track for at least 4 hours.

Figure 3.1: Must haves

3.1.2. SHOULD HAVES

The requirements listed in Table 3.2 are not essential for the success of the project, but would greatly improve the overall quality and user experience.

Requirement
The user should be able to pause the data gathering.
The user should be warned when the application is started while location services are not available.
The data should be stored in a storage efficient data format.
The user should be able to select the sport he/she wants to track.
The data gathering should automatically be paused when the user does not move.
The data should be processed to improve the video rendering quality.
The application should be able to track for at least 5 hours.
The application should be able to continue a tracking activity after the application has been terminated.

Figure 3.2: Should haves

3.1.3. COULD HAVES

The requirements listed in Table 3.3 are not essential for the success of the project but would provide the user with additional features, improving the overall quality and user experience.

Requirement
The implementation could provide activity statistics.
The implementation could automatically detect when the user is performing an activity.
The user could see his current speed in real time.
The user could see the currently traveled distance in real time.
The rate at which location data is sampled could be adjusted dynamically.
The implementation could support additional data such as heart rate, step counter, cadence, and calories.
The applications could be able to track for at least 6 hours.

Figure 3.3: Could haves

3.1.4. WON'T HAVES

The requirements listed in Table 3.4 will not be included in the final product.

Requirement
The implementation won't be compatible with the Windows Phone platform
The implementation won't have to work with devices which have no GPS hardware.

Figure 3.4: Won't haves

3.2. ARCHITECTURE

The app currently used by the client is implemented in TypeScript in combination with the React Native framework. The new tracking feature created during the project is implemented as a React Native module implemented in TypeScript in combination with native modules. This ensures the tracking feature can be implemented completely independent of the current app and could easily be used in every React Native application. The responsibilities for the tracking module are the retrieval of location data and ensuring a persistent data storage.

3.2.1. OVERVIEW

The overview of the complete tracking module including the use case with the current mobile app is shown in Figure 3.5. The mobile app retrieves the location data from the tracker module and sends it to a server to create videos from it. The TypeScript layer of the tracker React Native module will provide a consistent interface to use the two different implementations of the native iOS and Android layer. Furthermore, the

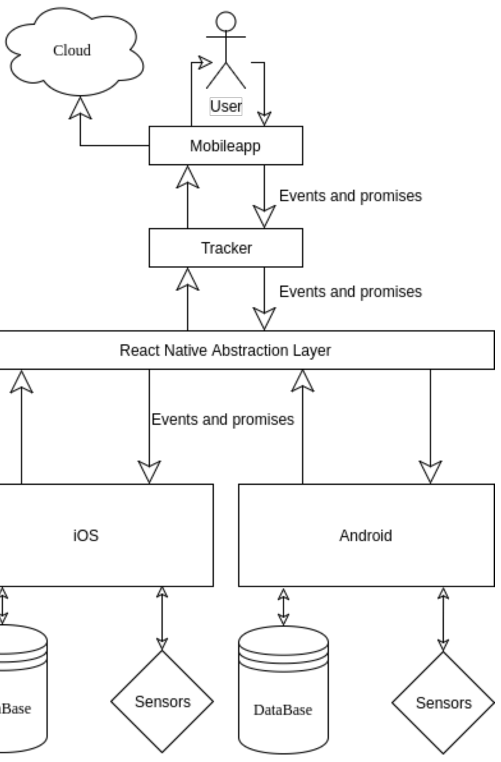


Figure 3.5: Global overview of the system

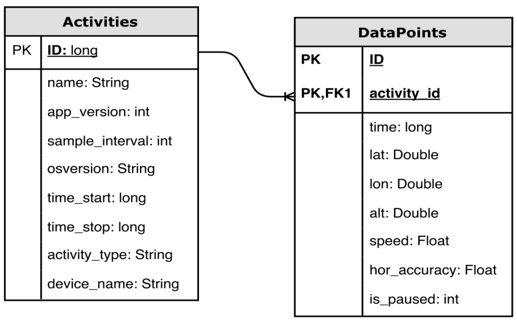
TypeScript layer keeps track of the tracking state and all functions which are the same for both iOS and Android. The communication to the outside from within the module is done using promises, function calls and events. Events are for example used to provide updates about the location and updates about the current state of the tracker. Within the module, there should be a communication link between the tracker module and native Android and iOS code. This communication can be established using the React Native abstract layer. The native code can send events to the React Native module using events, for example about location updates. The React Native module uses JavaScript promises to send information to the native modules. The two different implementations for the native iOS and Android code both have the responsibility to provide a persistent data storage and retrieving sensor data.

3.2.2. DATA GATHERING

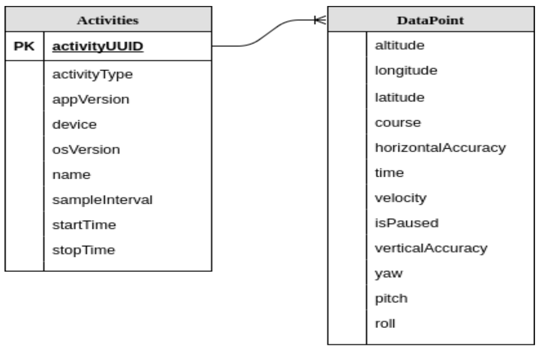
Collecting location data is the responsibility of the native code. React Native does not work in the background which makes this not a feasible solution for a tracking application. In the native iOS and Android code, it is more convenient to enable a background service to keep collecting location data. Sending the gathered data to a server is not the responsibility of the tracker module, but the responsibility of the users of the module.

3.2.3. DATA STORAGE

From the requirements, it follows that data tracked by the user should be persistent. No data should be lost when the user restarts the app or the app crashes. A database is chosen for the storage of this persistent data. The other options are a key-value pair storage which should primarily be used to store simple key-value pairs but not larger sets of data like the array of GPS locations. Another option is the use of the file system to store data in raw data files. This has a disadvantage, however, when using a simple implementation this requires reading and writing the file every time a change is to be made therefore this way of storing data would be very costly. Therefore a database system to store the data was chosen. React Native cannot trivially be run in the background which is required to ensure persistence of data when the app is running in the background. On both Android and iOS there is an easy way to implement code execution in the background. Therefore the database implementation will be made in the native Android and iOS code. On both platforms, a system like this is well supported and relatively easy to use. However, there are some slight differences between both implementations, on Android it is required to explicitly declare a Foreign Key relationship which is not needed on iOS. On both platforms, the database consists of two tables: an activities table where each row represents an activity with for example its type, sample rate and start time. The other table represents each location provided by the location services of the device. Example values of a location are the longitude, velocity and a flag isPaused indicating if the auto pausing feature detected that the user paused on this location. A simple test revealed that the database has a size of 1.41 MegaBytes for a single 6-hour activity (21600 location samples). Since modern phones have a storage capacity of at least 16 Giga Bytes or more and only a couple of activities will be stored on the phone this file size is small enough. Thus fulfilling the efficient data storage requirement. The database model can be found in Figure 3.6.



(a) Database model on Android



(b) Database model on iOS

Figure 3.6: Database model on both platforms

3.3. Kalman Filter Design

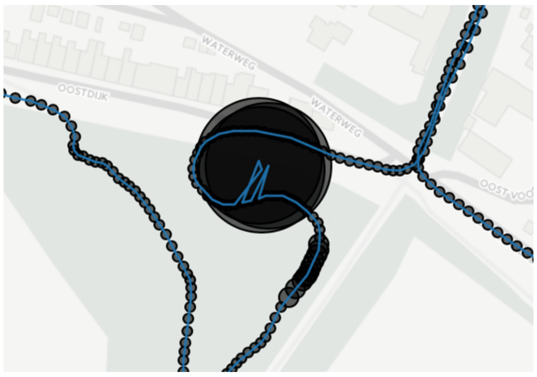
After obtaining methods to collect and store data from both Android and iOS devices, it became clear that some form of filtering was required to make the data visually appealing in video's generated by the client. Based on the solutions listed in the Research section, the decision was made to use a Kalman Filter to make the results returned from the device location services suitable for video rendering. In this section, the design choices made during the design of the filter will be explained.

3.3.1. Collected Data Accuracy

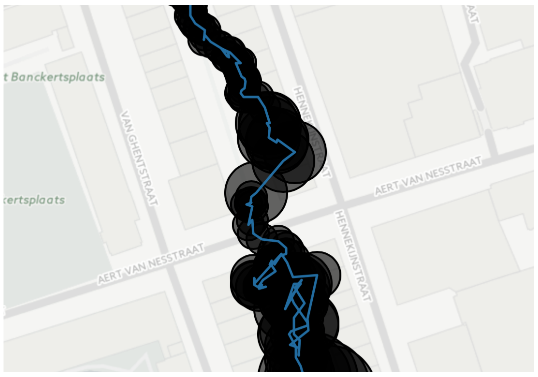
The raw, unprocessed data results in unsightly results for users tracking their activity. Since data tracked by the user is stored on an external server a high number of data samples is available from which this can be visualized. Figure D.6 contains a selection of received data and the videos generated by the client. From data sample 1, a straight line is expected since cyclists do not show unstable behavior while performing their ride. Data sample 2 shows uncertainty around the starting location. This uncertainty leads to a pointy object being rendered. Data sample 3 shows that some internal processing is done on the side of the client. While the data has inaccuracies at a bridge, they are not visible in the rendered video. Data sample 4 shows an extreme case of inaccuracy. The locations should be on a straight line, following the train track. To determine the parameters for the filter to be described in the next subsection, the horizontal accuracy which is reported by the phone for each location is used. The horizontal accuracy is defined as the radius of a circle in meters. The true location lies within this circle. Figure 3.7 plots these accuracies for two routes, visualizing the uncertainty of location when using GPS to determine location.

3.3.2. Kalman Filter

The Kalman filter is a Bayesian filter in which the assumption is made that the data is normally distributed. The measurement is taken as the mean and the uncertainty is taken as the variance. The state represents the values at a particular point in time, and the belief represents the probability that the system is in this state. The filtering algorithm consists of two steps. In the first step, a prediction is made to predict the next state. The second step is an update step in which the predicted state and belief are updated based on a weighted



(a) Losing the GPS signal for a short moment while cycling.



(b) Walking in a straight line in a city.

Figure 3.7: Overview of horizontal accuracy reported by iOS and Android.

average of the predicted and measured state [15]. Since each step contains information from the previous step, each state estimate contains information about all previous measurements. The Kalman Filter used for filtering the GPS data will track three dimensions of motion in three different dimensions: position, velocity, and acceleration in the x, y and z directions of a local coordinate plane. The local coordinate plane was chosen to make the output of the filter more readable by converting latitude and longitude in degrees to x and y in meters. After this conversion position, velocity and acceleration have the same unit. It also makes an extension of the filter more easy in the future. A way in which the filter can be extended is by including wheel sensor measurements from a Bluetooth connected device, which records the number of meters traveled for each rotation of the wheel. Next, an overview of the matrices that have to be designed in order for the filter to work correctly will be given.

Initialization Step

In the initialization step, the initial values of state **x** and state covariance (belief) **P** are designed. Newton dot notation is used to mark speed and acceleration, which are the first and second derivatives of position. State **x** is defined as a vector containing nine state variables. The initial position in x, y and z are known, as it is the first location in a time-ordered list of locations. Since velocity and acceleration are not known in the starting location, they are set to 0, assuming a stationary starting position.

$$\mathbf{x} = \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ y \\ \dot{y} \\ \ddot{y} \\ z \\ \dot{z} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} x_0 \\ 0 \\ 0 \\ y_0 \\ 0 \\ 0 \\ z_0 \\ 0 \\ 0 \end{bmatrix} \tag{3.1}$$

State covariance **P** is a matrix containing the covariances between state variables which gives a measure of variance between state variables, allowing prediction. The diagonal contains the variance σ^2 of each state variable. The other elements contain covariances $\sigma_{i,j}$ between state variables. This matrix is initialized as a matrix filled with zeros, except on the diagonal. The diagonal is initialized with an approximation of known variances in the data.

$$\mathbf{P} = \begin{bmatrix} \sigma_x^2 & \sigma_{x,\dot{x}} & \sigma_{x,\ddot{x}} & \cdots & \cdots & \cdots & \cdots & \cdots & \sigma_{x,\ddot{z}} \\ \sigma_{\dot{x},x} & \sigma_{\dot{x}}^2 & \sigma_{\dot{x},\ddot{x}} & \cdots & \cdots & \cdots & \cdots & \cdots & \sigma_{\dot{x},\ddot{z}} \\ \sigma_{\ddot{x},x} & \sigma_{\ddot{x},\dot{x}} & \sigma_{\ddot{x}}^2 & \cdots & \cdots & \cdots & \cdots & \cdots & \sigma_{\ddot{x},\ddot{z}} \\ \vdots & \vdots & \vdots & \sigma_y^2 & \cdots & \cdots & \cdots & \cdots & \sigma_{y,\ddot{z}} \\ \vdots & \vdots & \vdots & \vdots & \sigma_{\dot{y}}^2 & \cdots & \cdots & \cdots & \sigma_{\dot{y},\ddot{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \sigma_{\ddot{y}}^2 & \cdots & \cdots & \sigma_{\ddot{y},\ddot{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \sigma_z^2 & \cdots & \sigma_{z,\ddot{z}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \sigma_{\dot{z}}^2 & \sigma_{\dot{z},\ddot{z}} \\ \sigma_{\ddot{z},x} & \sigma_{\ddot{z},\dot{x}} & \sigma_{\ddot{z},\ddot{x}} & \sigma_{\ddot{z},y} & \sigma_{\ddot{z},\dot{y}} & \sigma_{\ddot{z},\ddot{y}} & \sigma_{\ddot{z},z} & \sigma_{\ddot{z},\dot{z}} & \sigma_{\ddot{z}}^2 \end{bmatrix} \tag{3.2}$$

PREDICTION STEP

In the prediction step, state and covariance are predicted using a state transition matrix **F** and process noise matrix **Q**. The state transition matrix is a model on which we base our prediction. The process noise matrix accounts for modelling errors. The outputs of the prediction step are prior state ($\bar{\mathbf{x}}$) and covariance ($\bar{\mathbf{P}}$). In this context, prior is defined as the distribution before incorporating the measurement. The prior values are calculated using the following prediction equations:

$$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x} \tag{3.3}$$

$$\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^{\mathbf{T}} + \mathbf{Q} \tag{3.4}$$

State transitions are modeled using Newton's equations of motion. The equation for motion with constant acceleration is

$$x_0 + v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2 \tag{3.5}$$

If this formula is applied to each state variable, the following state transition matrix is obtained where Δt represents the sampling rate of the collected data.

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

When tracking an activity the acceleration is never constant. To account for the constant acceleration assumption, we add process noise to the prior state covariance. This is represented by **Q**. To obtain **Q**, a piecewise noise model[15] is used in which we assume that the acceleration is constant during one sample rate interval, but is different for each interval. **Q** is computed for one dimension (i.e. x) using the following equation:

$$\mathbf{Q_x} = \Gamma\sigma^2\Gamma^T \qquad \text{where} \qquad \Gamma = \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \\ 1 \end{bmatrix} \tag{3.7}$$

$$\mathbf{Q_x} = \begin{bmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 & \Delta t^2 & \Delta t \\ \Delta t^2 & \Delta t & 1 \end{bmatrix} \tag{3.8}$$

If this matrix is extended to all three dimensions, the following matrix for Q is obtained:

$$\mathbf{Q}=\begin{bmatrix} \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2}\Delta t^2 & \Delta t^2 & \Delta t & 0 & 0 & 0 & 0 & 0 & 0 \\ \Delta t^2 & \Delta t & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 & \frac{1}{2}\Delta t^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}\Delta t^2 & \Delta t^2 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta t^2 & \Delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4}\Delta t^4 & \frac{1}{2}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}\Delta t^2 & \Delta t^2 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & \Delta t^2 & \Delta t & 1 \end{bmatrix} \tag{3.9}$$

Update Step

In the update step, the predicted state ($\bar{\mathbf{x}}$) is subtracted from the measurement $\mathbf{z}$. $\mathbf{z}$ contains the measured x, y and z position. The result of this subtraction is the residual $\mathbf{y}$. The residual is weighted using the Kalman gain $\mathbf{K}$ to obtain the prediction after incorporating he measurement, the posterior prediction. To compare the measurement to $\bar{\mathbf{x}}$, $\bar{\mathbf{x}}$ has to be mapped to the same dimension as $\mathbf{z}$ using a measurement function $\mathbf{H}$. Since phone measurements are noisy, we model this measurement noise using a covariance matrix $\mathbf{R}$. The posterior is calculated using the following equations:

$$\mathbf{y}=\mathbf{z}-\mathbf{H}\bar{\mathbf{x}} \tag{3.10}$$

$$\mathbf{K}=\bar{\mathbf{P}}\mathbf{H}^{\mathbf{T}}(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^{\mathbf{T}}+\mathbf{R})^{-1} \tag{3.11}$$

$$\mathbf{x}=\bar{\mathbf{x}}+\mathbf{K}\mathbf{y} \tag{3.12}$$

$$\mathbf{P}=(\mathbf{I}-\mathbf{K}\mathbf{H})\bar{\mathbf{P}} \tag{3.13}$$

The measurement function $\mathbf{H}$ is a three by nine matrix, mapping each state to its corresponding measurement.

$$\mathbf{H}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.14}$$

$\mathbf{R}$ is a three by three matrix containing the noise of the sensor measurements. They are set to the approximate variance σ^2 of the measurements, to account for sensor noise.

$$\mathbf{R}=\begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix} \tag{3.15}$$

To conclude this section, a visual representation of the filter is given. Figure 3.8 illustrates a single iteration of the filter.

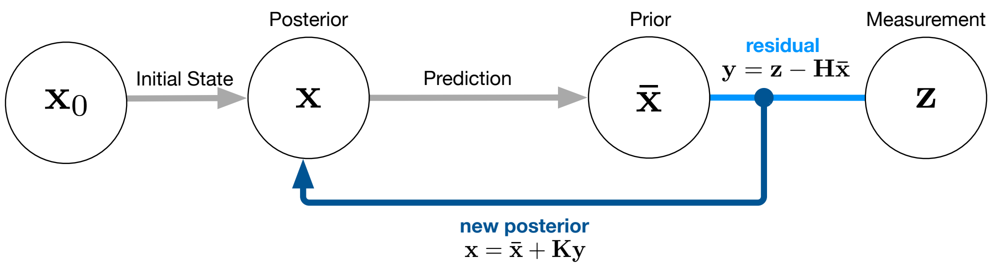


Figure 3.8: Single iteration step of the Kalman Filter.

3.3.3. RAUCH-TUNG-STRIEBEL SMOOTHING

In some cases, filtered data is still found to be unsuitable for rendering a visually pleasing video. To solve this problem, an additional step is added to smooth the output of the filter. For smoothing, the optimal Rauch Tung Striebel smoother [16] is used. The algorithm moves backward on the data, incorporating future measurements into the smoothing. As a result, the predicted state contains information about all measurements. Because this smoothing requires all data, it can only be executed after finishing an activity. The smoother contains two steps which are defined by the following equations.

PREDICTION STEP

$$\mathbf{P} = \mathbf{F}\mathbf{P}_i\mathbf{F}^{\mathsf{T}} + \mathbf{Q} \tag{3.16}$$

UPDATE STEP

$$\mathbf{K}_i = \mathbf{P}_i\mathbf{F}^{\mathsf{T}} \quad \mathbf{P}^{-1} \tag{3.17}$$

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{K}_i(\mathbf{x}_{i+1} - \mathbf{F}\mathbf{x}_i) \tag{3.18}$$

$$\mathbf{P}_i = \mathbf{P}_i + \mathbf{K}_i(\mathbf{P}_{i+1} - \mathbf{P})\mathbf{K}_i^{\mathsf{T}} \tag{3.19}$$

3.3.4. ALGORITHM DESIGN

The mathematical implementation of the filter has to be converted to an algorithm in order to make implementation possible on mobile phones. The algorithm should iterate over all locations, keeping track of posterior state **x** and state covariance **P**. During each iteration step, both the predict and the update step of the filter are executed.

```
Function predict($\mathbf{x}, \mathbf{P}, \mathbf{F}, \mathbf{Q}$):
    input  : Current state $\mathbf{x}$.
             Current state covariance $\mathbf{P}$.
             State transition matrix $\mathbf{F}$.
             Process noise matrix $\mathbf{Q}$.
    output: Prior state $\bar{\mathbf{x}}$ and state covariance $\bar{\mathbf{P}}$

    $\bar{\mathbf{x}} \longleftarrow \mathbf{F}\mathbf{x}$;
    $\bar{\mathbf{P}} \longleftarrow \mathbf{F}\mathbf{P}\mathbf{F}^{\mathsf{T}} + \mathbf{Q}$;

    return $\bar{\mathbf{x}}, \bar{\mathbf{P}}$
```

Algorithm 1: Kalman Filter Prediction Function

```
Function update($\bar{\mathbf{x}}, \bar{\mathbf{P}}, \mathbf{H}, \mathbf{R}$):
    input  : Measurement vector $\mathbf{z}$.
             Prior state $\mathbf{x}$.
             Prior state covariance $\mathbf{P}$.
             Measurement function $\mathbf{H}$.
             Noise covariance matrix $\mathbf{R}$
    output: Posterior state $\mathbf{x}$ and state covariance $\mathbf{P}$

    $\mathbf{y} \longleftarrow \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$;
    $\mathbf{K} \longleftarrow \bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}}(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^{\mathsf{T}} + \mathbf{R})^{-1}$;
    $\mathbf{x} \longleftarrow \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$;
    $\mathbf{P} \longleftarrow (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}}$;

    return $\mathbf{x}, \mathbf{P}$
```

Algorithm 2: Kalman Filter Update Function

After the Kalman Filter has been executed, the posterior state **x** and state covariance **P** are smoothed using the Rauch-Tung-Striebel method.

```
Function rtsSmooth($\bar{\mathbf{x}}, \bar{\mathbf{P}}, \mathbf{H}, \mathbf{R}$):
    input  : Prior states $\mathbf{x}_{\text{filtered}} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$.
             Prior state covariances $\mathbf{P}_{\text{filtered}} = \{\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_n\}$.
             State transition matrix $\mathbf{F}$.
             Process noise matrix $\mathbf{Q}$.
    output: Smoothed state $\mathbf{x}_{\text{filtered}}$
    $\mathbf{K} \longleftarrow \emptyset$;
    for $i \longleftarrow 1$ to $n$ do
        $\mathbf{K}_i \longleftarrow \emptyset$;
        $\mathbf{K} \longleftarrow \mathbf{K} \cup \{\mathbf{K}_i\}$;
    end
    $\mathbf{K} \longleftarrow \{\mathbf{K}_1, \mathbf{K}_2, ..., \mathbf{K}_n\}$;
    for $i \longleftarrow n-1$ to $1$ do
        $\mathbf{P} \longleftarrow \mathbf{F}\mathbf{P}_i\mathbf{F}^T + \mathbf{Q}$;
        $\mathbf{K}_i \longleftarrow \mathbf{P}_i\mathbf{F}^\mathsf{T} \ \ \mathbf{P}^{-1}$;
        $\mathbf{x}_i \longleftarrow \mathbf{x}_i + \mathbf{K}_i(\mathbf{x}_{i+1} - \mathbf{F}\mathbf{x}_i)$;
        $\mathbf{P}_i \longleftarrow \mathbf{P}_i + \mathbf{K}_i(\mathbf{P}_{i+1} - \mathbf{P})\mathbf{K}_i^\mathsf{T}$;
    end
    return $\mathbf{x}_{filtered}$
```

Algorithm 3: Rauch-Tung-Striebel Smoothing Algorithm

To aid readability, the predict, update and smoothing steps are given in three separate algorithms, Algorithm 1, Algorithm 2 and Algorithm 3. Combined they form the main data processing step of the system.

```
input  : $L = \{l_0, l_1, ..., l_t\}$ sorted list of location objects containing x, y and z.
         Sample rate of the data $t$.
output: $L' = \{l_0, l_1, ..., l_t\}$ sorted list of location objects containing x, y and z.
begin
    $\mathbf{x} \longleftarrow$ Initial state;
    $\mathbf{P} \longleftarrow$ Initial state covariance;
    $\mathbf{H} \longleftarrow$ Measurement function;
    $\mathbf{R} \longleftarrow$ Measurement noise covariance;
    $\mathbf{F} \longleftarrow$ State transition matrix;
    $\mathbf{Q} \longleftarrow$ Process noise matrix;
    $\mathbf{x}_{\text{filtered}} \longleftarrow \emptyset$;
    $\mathbf{P}_{\text{filtered}} \longleftarrow \emptyset$;
    for $l$ in $L$ do
        $\bar{\mathbf{x}}, \bar{\mathbf{P}} \longleftarrow$ predict($\mathbf{x}, \mathbf{P}, \mathbf{F}, \mathbf{Q}$);
        $\mathbf{x}, \mathbf{P} \longleftarrow$ update($\bar{\mathbf{x}}, \bar{\mathbf{P}}, \mathbf{H}, \mathbf{R}$);
        $\mathbf{x}_{\text{filtered}} \longleftarrow \mathbf{x}_{\text{filtered}} \cup \{\mathbf{x}\}$;
        $\mathbf{P}_{\text{filtered}} \longleftarrow \mathbf{P}_{\text{filtered}} \cup \{\mathbf{P}\}$;
    end
    $\mathbf{x}_{\text{filtered}} \longleftarrow$ rtsSmooth($\mathbf{x}_{filtered}, \mathbf{P}_{filtered}, \mathbf{F}, \mathbf{Q}$);
    return $\mathbf{x}_{filtered}$
end
```

Algorithm 4: Kalman Filter Algorithm

Because the size of the matrices is known before executing the algorithm, the number of matrix operations is constant for each iteration, allowing the time complexity of the algorithm to be in the order of $O(n)$, where n represents the length of the list of input locations.

3.3.5. FILTER PARAMETERS

Filter parameters are determined by examining a lot of data sets, and experimentally tweaking the noise matrices. Due to all tracked activities being stored on an external server, large amounts of data were available in the final weeks of the project. These datasets were used to determine the final parameters for the filter. The following parameters have to be determined in order for the filter to function correctly:

- Initial state covariance **P**.
- Measurement noise variance **R**.
- Process noise variance. **Q**.

By examination of the horizontal accuracy of a large number of datasets, the variance in the horizontal accuracy of the sensors can be approximated. After removing locations with unknown speed and accuracy, a total of 361801 GPS locations from 235 collected datasets were used to approximate the variances. The values were retrieved by using the describe() method of the Pandas [1] Python library to make a quick analysis of the results possible. Table 3.9 contains the results of the analysis for velocity and horizontal accuracy.

	mean	σ	σ^2	**min**	**25%**	**50%**	**75%**	**max**
horizontal accuracy (m)	10.139	15.686	246.050	2.000	4.000	6.000	12.000	400.000
velocity (m/s)	7.705	8.477	71.860	0.000	1.190	5.173	9.532	128.000

Figure 3.9: Overview of 351801 GPS locations retrieved 235 datasets collected by the application.

From the horizontal accuracy the variance in x and y direction is taken as the initial uncertainty. x and y velocity variance is set to the variance in speed measurements. All other values for which no accuracy information is present, the values are set to a relatively high number. **P** is now defined as a zero filled matrix with the following values on the diagonal:

$$\mathbf{x} = \begin{bmatrix} \sigma_x^2 \\ \sigma_{\dot{x}}^2 \\ \sigma_{\ddot{x}}^2 \\ \sigma_y^2 \\ \sigma_{\dot{y}}^2 \\ \sigma_{\ddot{y}}^2 \\ \sigma_z^2 \\ \sigma_{\dot{z}}^2 \\ \sigma_{\ddot{z}}^2 \end{bmatrix} = \begin{bmatrix} 246.1 \\ 71.9 \\ 500.0 \\ 246.1 \\ 71.9 \\ 500.0 \\ 500.0 \\ 500.0 \\ 500.0 \end{bmatrix} \tag{3.20}$$

Since the values obtained from the datasets represent real measurements from various phones, **R** can be initialized with the same values. Due to the measurement noise of the altitude measurement being unknown, the altitude variance is assumed to be equal to the horizontal accuracy.

$$\mathbf{R} = \begin{bmatrix} 246.1 & 0 & 0 \\ 0 & 246.1 & 0 \\ 0 & 0 & 246.1 \end{bmatrix} \tag{3.21}$$

The process noise variance **Q** is determined by examining filter results. For small variances, the filter will favor the prediction. For large values, the filter will favor the measurement. **Q** can be used to balance the output of the filter. To illustrate this, two activities are processed for four different variances of Q. The results are shown in D.5. The other parameters are set as discussed in this subsection. By looking at the output of the filter, it was found that a value between 0.1 and 1.0 obtains the best results. Smaller values cannot remove the variance in the signals, making the data unsuitable for video rendering. Large values create diverging patterns around sharp corners.

[1]Pandas, `http://pandas.pydata.org`

3.3.6. DATA PREPROCESSING

Before applying the filter, a number of highly unlikely locations can be removed from the raw data. Data with very large horizontal accuracies in the order of a few kilometers are removed from the data before applying the filter since it is highly unlikely the user has actually been there. For all activities, speeds above 70m/s are unlikely to be reached. These locations will also be removed. The same holds for extremely low speeds below 0.2m/s. The location tracking implementation of the phone will keep track of pauses. Locations flagged as *paused* will also be removed before applying the filter.

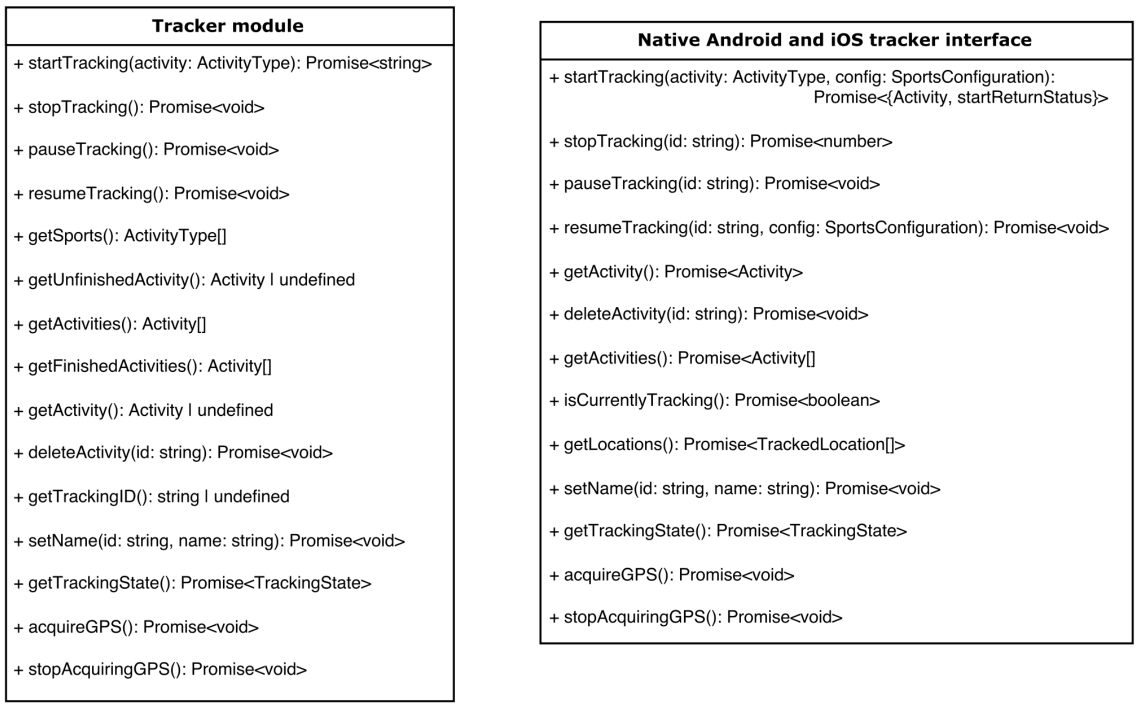
4
Product Implementation

In this chapter, the implementation of the different features implemented in the React Native tracking module are discussed. First, an overview for each part of the module is given. In the second section, the differences between the Android and iOS implementations are pointed out.

4.1. Overview Implementation

The implementation of the React Native tracking module is done in three separated parts as described earlier. In this section, the implementation of each part is described.

4.1.1. TypeScript implementation



```
Tracker module
+ startTracking(activity: ActivityType): Promise<string>
+ stopTracking(): Promise<void>
+ pauseTracking(): Promise<void>
+ resumeTracking(): Promise<void>
+ getSports(): ActivityType[]
+ getUnfinishedActivity(): Activity | undefined
+ getActivities(): Activity[]
+ getFinishedActivities(): Activity[]
+ getActivity(): Activity | undefined
+ deleteActivity(id: string): Promise<void>
+ getTrackingID(): string | undefined
+ setName(id: string, name: string): Promise<void>
+ getTrackingState(): Promise<TrackingState>
+ acquireGPS(): Promise<void>
+ stopAcquiringGPS(): Promise<void>
```

```
Native Android and iOS tracker interface
+ startTracking(activity: ActivityType, config: SportsConfiguration):
                                    Promise<{Activity, startReturnStatus}>
+ stopTracking(id: string): Promise<number>
+ pauseTracking(id: string): Promise<void>
+ resumeTracking(id: string, config: SportsConfiguration): Promise<void>
+ getActivity(): Promise<Activity>
+ deleteActivity(id: string): Promise<void>
+ getActivities(): Promise<Activity[]
+ isCurrentlyTracking(): Promise<boolean>
+ getLocations(): Promise<TrackedLocation[]>
+ setName(id: string, name: string): Promise<void>
+ getTrackingState(): Promise<TrackingState>
+ acquireGPS(): Promise<void>
+ stopAcquiringGPS(): Promise<void>
```

Figure 4.1: TypeScript interface exposed to outside the module and interface to the native Android and iOS implementation

The TypeScript part of the implementation is used to implement everything that is the same for both platforms. It connects the two different native implementations to one interface and adds some extra implementation. The two interfaces of the TypeScript and native implementation can be found in Figure 4.1 The advantage of adding functionality in this layer of the code is that it only has to be done once, which results in less code duplication. The TypeScript layer keeps track of the current tracking state and the activities which are in memory. The processing of the data using the Kalman filter is also done in the TypeScript code.

4.1.2. ANDROID

A UML diagram of the Android implementation in Java can be found in Appendix E.1. The SkyhawkTrackerModule class defines the methods which are exposed to the TypeScript layer of the module. From this class, the tracking can be controlled and already tracked activities can be retrieved. To store Locations a TrackedLocation class exists that extends the default Android Location class, for storing activities there is a TrackedActivity class.

4.1.3. IOS

The iOS implementation is done in the programming language Objective-C. In the research phase of the project, Swift was considered as a programming language for the native implementation because it is more readable. In the research phase of the project, we directly added native code to the React Native application. This allowed Swift to be used. However, our product implementation is a separate module which is imported in the client's implementation. Since a separate module is a static library, and Swift does not yet support static libraries Swift could not be used. This became an additional challenge for the team since a new language had to be learned. The UML diagram of the iOS implementation can be found in Appendix E.2. The Skyhawk-Tracker class is the communication link between the TypeScript code and the Objective-C implementation. From this class, events are sent back to the TypeScript implementation and it exposes functions that can be called from the TypeScript layer. Events to TypeScript can only be sent from this module, because of the required React dependencies. The TrackingManager and GPSStatusMonitor use a delegate method provided by the SkyhawkTracker class to send events about location updates to the TypeScript. In this way, only the necessary methods are exposed to the TrackingManager and GPSStatusMonitor.

4.2. FEATURES IMPLEMENTATION

The implementations of the different features are described in this section. For features which are not the same for Android and iOS, the differences are discussed.

4.2.1. TRACKING IMPLEMENTATION

The collection of location data is done in the native Android and iOS code as discussed before. During an activity, tracking locations needs to continue even when the app is in the background. The implementations for this differ on both platforms. Both platforms provide a different framework to collect location data and to allow running them while the application is in the background. The tracking implementation should also provide a way to dynamically change the sample interval which could change for different sport activity types.

ANDROID

On Android, the Android location API is used. With this API it is possible to choose the location provider which means the source of the location data is known. To keep tracking locations in the background a service needs to be started. This service stays active even when the React Native app is in the background. The service starts tracking locations by adding a location listener to the Android LocationManager class.

IOS

On iOS, the Core Location framework is used. This framework provides services for determining a device's geographic location, altitude, and orientation. The framework uses all available onboard hardware, including Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware to gather data. The framework even provides a method to receive location updates in the background without a lot of extra work.

4.2.2. DATABASE IMPLEMENTATION

The implementation of the database differs for Android and iOS because two different frameworks are used. In this section, the specific implementations are described.

Android

On Android, various classes for interacting with an SQLite database are implemented. One of them is a DataManager class to manage reading and writing to the database. This class is a singleton to ensure only a single connection with the database is open at a time. This class uses the SQLiteOpenHelper and DataBaseContract classes to interact with the database. On Android, the datapoint table also has a column indicating to which activity the GPS location belongs connecting the two tables.

iOS

On iOS, the Core Data framework is used. All relationships between different tables are handled by the framework. This makes it convenient to use. The framework provides for every table a class for which it guarantees changes made in a created object are reflected in the database. This makes the usage rather simple since you can even create custom methods for these classes to directly modify the data in the database.

4.2.3. Auto pausing

Location data received from the smartphone's operating system contains besides location data also information about the velocity. This information is used to detect whether a new location point is a point where the user was actually moving or was taking a break. These points are marked as paused but are not removed. This way the user of the module could decide later whether these points should be removed from the result or not. The isPaused flag is decided based on the last 5 points, to be insensitive to outliers in the velocity data. For these data points, a check is applied whether the speed of 4 out of the 5 locations are below a certain threshold. If this is the case the isPaused flag is set to true for the following points until 4 out of the 5 last points exceed this threshold again. This method proved to be a fairly robust way to detect if the user is not moving. This value can be easily configured for each type of activity via a configuration file. With some testing thresholds of 0.8m/s for cycling and 0.3m/s for running where found.

4.2.4. GPS acquiring

The first few locations gathered by almost all GPS chips are not very accurate. Possibly because the phone is still establishing a connection with more GPS satellites. To overcome this problem the module offers the possibility to start requesting location updates before the actual tracking is started. This allows listening for GPS status changes when for example the user turns off the location services or declines the permission for the location services. This information can be used by the user of the module to request permission or notify the user about a problem with the GPS status.

4.3. Kalman Filter Implementation

Coordinate Conversion

Because the filter is designed to function on a local coordinate plane in meters and the measurements are in World Geodetic System (WGS84)[1] coordinates a conversion is required. To allow this conversion two steps are required. First, the latitude and longitude coordinates have to be converted to the ECEF coordinate system. If the coordinates are in ECEF format, the final transformation to a local East-North-Up (ENU)[2] coordinate system can be applied. After the filter is applied, ENU is converted back to ECEF and ECEF is converted back to WGS84. For this conversion to work correctly, it is important to set a reference position, which is the starting location of the tracked activity. [17] lists the algorithms used for this conversion.

Python Prototype

To make development and changing parameters easier, first, a Python implementation of the filtering steps has been developed. The Python implementation reads the client's file format and loads the data in a Pandas data frame. Then the preprocessing step is executed, reducing the number of points. Finally, the filter is applied and the results are plotted on a map using the Folium[3] map plotting library.

[1]WGS84, `http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html`
[2]ECEF to ENU, ESA Navipedia, `http://www.navipedia.net/index.php/Transformations_between_ECEF_and_ENU_coordinates`
[3]Folium, `https://github.com/python-visualization/folium`

NUMERICJS

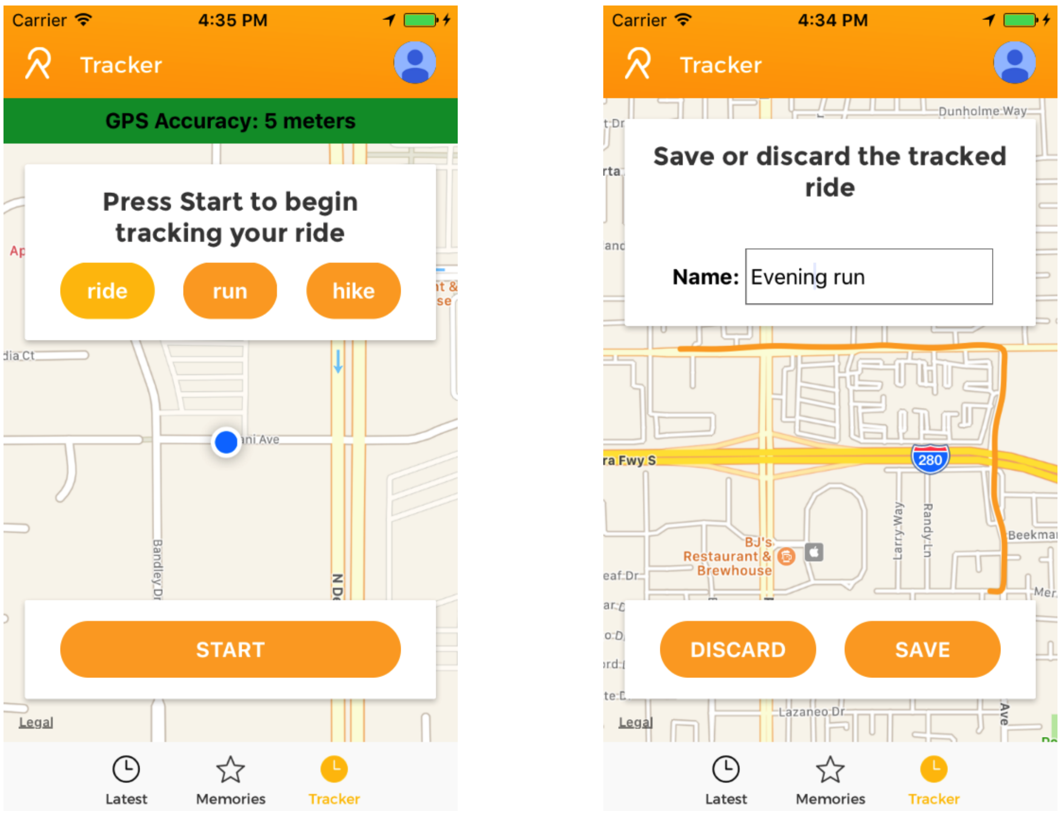
The designed filter implementation is implemented in the tracker module TypeScript code using the NumericJS library [4]. NumericJS allows matrix operations and defines matrices as a 2D array. A wrapper class was built around NumericJS allowing Matrix operations to be written in a readable format.

FILTER RESULTS

In the Product Design chapter, Figure D.6 lists inaccurate video results. These results significantly improved after applying the designed filter. Figure D.7 contains the filtered versions of the inaccurate data sets. Data sample 1 no longer varies on the road. In data sample 2, the starting point is no longer noisy, but a smooth line heading towards the road. Data sample 3 no longer has a strange edge before entering the tunnel. The biggest improvement can be seen in data sample 4. All inaccurate locations have been removed, and the train track is now followed correctly.

4.4. TRACKING LIBRARY APP USE CASE

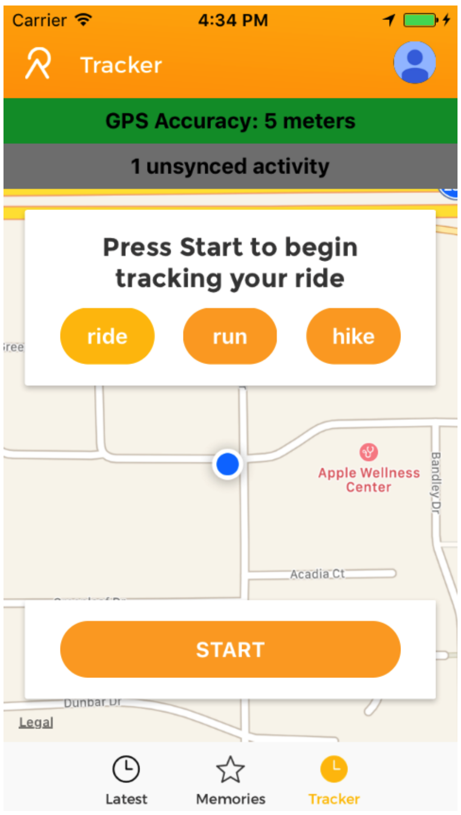
The module should be used in an application providing the user interface. To make this possible a tracking view is made inside the already existing app of the client. In this way, it is possible to directly test the functionality of the library in real world scenarios. From this application, the data is automatically sent to an Amazon S3[5] bucket, stored in the cloud. All this data can then be used for analyzing the behavior of the app and filter in a lot of different cases. Figure 4.2 and Figure 4.3 gives an overview of the implementation in the client's application.



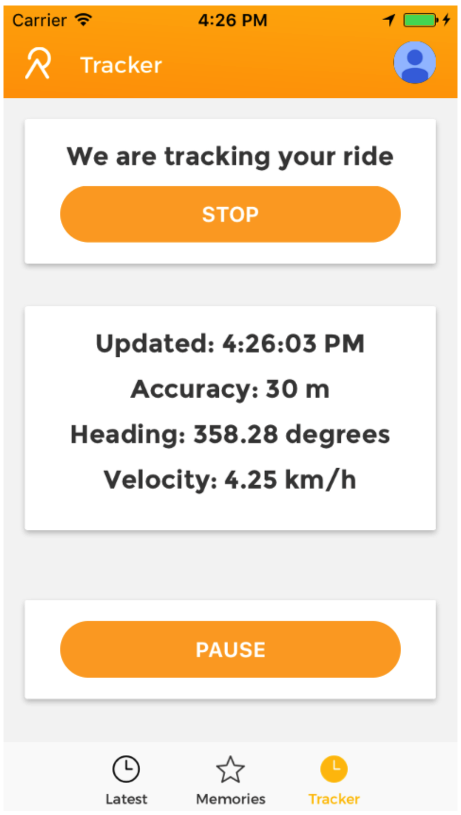
(a) Stopped Screen (b) Finished Screen

Figure 4.2: Overview of screens implemented in the client's application, the views before starting and after finishing an activity.

[4]NumericJS, http://www.numericjs.com
[5]Amazon S3, https://aws.amazon.com/s3



(a) Unsynced activity



Updated: 4:26:03 PM
Accuracy: 30 m
Heading: 358.28 degrees
Velocity: 4.25 km/h

(b) Tracking Screen

Figure 4.3: Overview of screens implemented in the client's application, showing the cases where an activity has not been synced and where the user is currently tracking.

5 Product Quality Evaluation

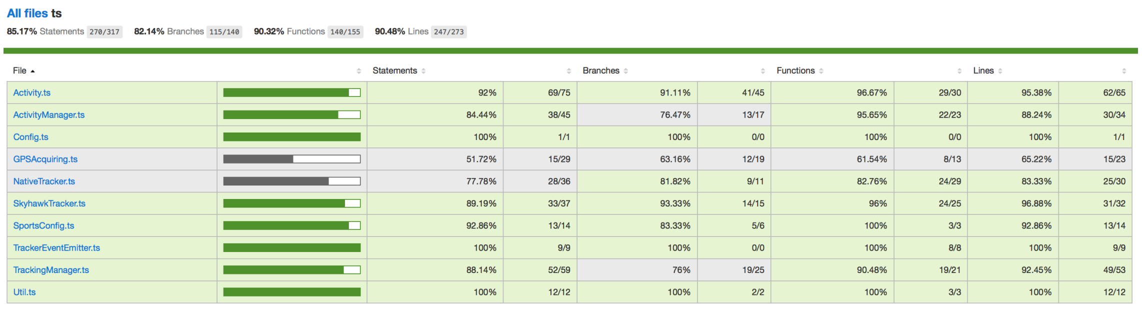
To ensure that the final product is of good quality, several measures have been taken. First, various testing techniques have been used. Second, the code has been verified by the Software Improvement Group[1]. This chapter is concluded by comparing the requirements to the actual implementation to make sure all essential requirements have been fulfilled.

5.1. Testing

Testing is essential to ensure the quality of the developed software. It can be done in various ways with different costs and needed efforts. Testing can prevent bugs, but it does not ensure the system is bug-free. In the next subsections, the two forms of testing during the project are described.

5.1.1. Unit testing

Unit tests are used to ensure the quality of the different subcomponents of the developed software. For the three different languages, we also needed to use three different kinds of unit testing frameworks. Which will be discussed in the following subsections.

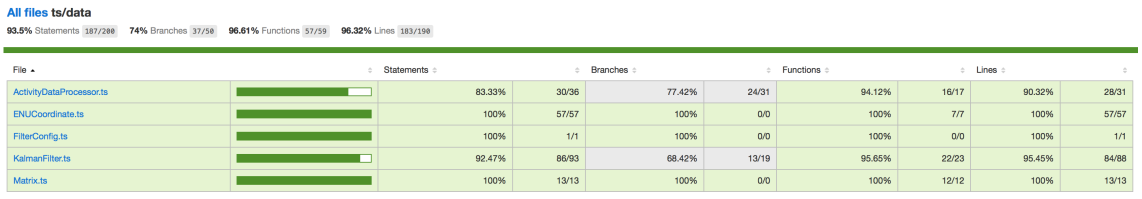


All files ts

85.17% Statements 270/317 **82.14%** Branches 115/140 **90.32%** Functions 140/155 **90.48%** Lines 247/273

File		Statements		Branches		Functions		Lines	
Activity.ts		92%	69/75	91.11%	41/45	96.67%	29/30	95.38%	62/65
ActivityManager.ts		84.44%	38/45	76.47%	13/17	95.65%	22/23	88.24%	30/34
Config.ts		100%	1/1	100%	0/0	100%	0/0	100%	1/1
GPSAcquiring.ts		51.72%	15/29	63.16%	12/19	61.54%	8/13	65.22%	15/23
NativeTracker.ts		77.78%	28/36	81.82%	9/11	82.76%	24/29	83.33%	25/30
SkyhawkTracker.ts		89.19%	33/37	93.33%	14/15	96%	24/25	96.88%	31/32
SportsConfig.ts		92.86%	13/14	83.33%	5/6	100%	3/3	92.86%	13/14
TrackerEventEmitter.ts		100%	9/9	100%	0/0	100%	8/8	100%	9/9
TrackingManager.ts		88.14%	52/59	76%	19/25	90.48%	19/21	92.45%	49/53
Util.ts		100%	12/12	100%	2/2	100%	3/3	100%	12/12

(a) Activity Management



All files ts/data

93.5% Statements 187/200 **74%** Branches 37/50 **96.61%** Functions 57/59 **96.32%** Lines 183/190

File		Statements		Branches		Functions		Lines	
ActivityDataProcessor.ts		83.33%	30/36	77.42%	24/31	94.12%	16/17	90.32%	28/31
ENUCoordinate.ts		100%	57/57	100%	0/0	100%	7/7	100%	57/57
FilterConfig.ts		100%	1/1	100%	0/0	100%	0/0	100%	1/1
KalmanFilter.ts		92.47%	86/93	68.42%	13/19	95.65%	22/23	95.45%	84/88
Matrix.ts		100%	13/13	100%	0/0	100%	12/12	100%	13/13

(b) Filter implementation

Figure 5.1: Typescript coverage generated with Istanbul

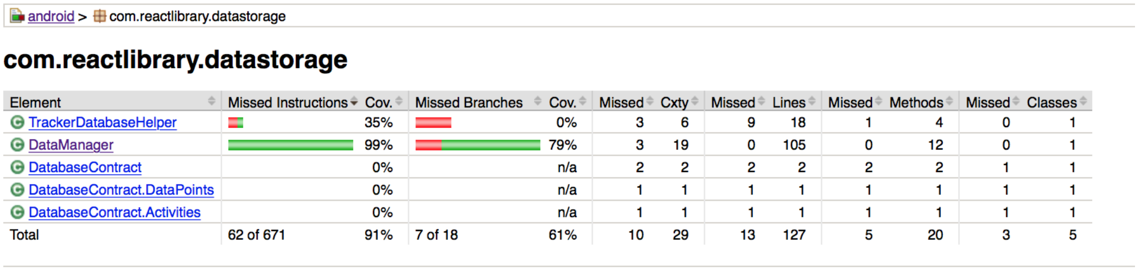
[1] Software Improvement Group https://www.sig.eu

TYPESCRIPT TESTING

Jest is used to test the typescript code which made it possible to mock dependencies of different modules. This made it possible to test modules completely independently of each other. The coverage for the Typescript code, generated using Istanbul [2] can be found in Figure 5.1.

JAVA TESTING

The Java Android code is tested in two different ways. For code that requires the Android operating system, there are instrumented tests which are executed on an Android emulator or phone. The other classes are tested with JUnit, these tests are run in the JVM. Difficulties with Java testing were the dependence on special React Native types necessary for communicating with the TypeScript layer. Testing in Android proved to be quite challenging. In unit testing, Mockito is used to mock dependencies to be able to test separate classes in isolation from others. Some classes like WritableMap from the react Native library were not available during testing, therefore PowerMock[3] had to be used. Some of the functionality of PowerMock could be considered a bad practice because it might cause testing against an implementation rather than an interface. In this case, it is only used because otherwise some classes are very difficult to test. Some classes required instrumentation and mocking for testing. No framework for mocking in instrumented tests was available, therefore some classes are not thoroughly tested. The overall test coverage for the Java implementation can be found in 5.2

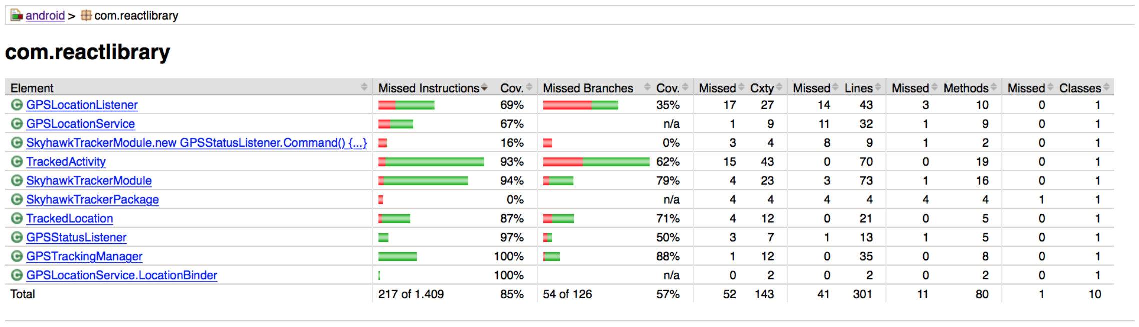


android > com.reactlibrary.datastorage

com.reactlibrary.datastorage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
TrackerDatabaseHelper		35%		0%	3	6	9	18	1	4	0	1
DataManager		99%		79%	3	19	0	105	0	12	0	1
DatabaseContract		0%		n/a	2	2	2	2	2	2	1	1
DatabaseContract.DataPoints		0%		n/a	1	1	1	1	1	1	1	1
DatabaseContract.Activities		0%		n/a	1	1	1	1	1	1	1	1
Total	62 of 671	91%	7 of 18	61%	10	29	13	127	5	20	3	5

(a) Data Management



android > com.reactlibrary

com.reactlibrary

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GPSLocationListener		69%		35%	17	27	14	43	3	10	0	1
GPSLocationService		67%		n/a	1	9	11	32	1	9	0	1
SkyhawkTrackerModule.new GPSStatusListener.Command() {...}		16%		0%	3	4	8	9	1	2	0	1
TrackedActivity		93%		62%	15	43	0	70	0	19	0	1
SkyhawkTrackerModule		94%		79%	4	23	3	73	1	16	0	1
SkyhawkTrackerPackage		0%		n/a	4	4	4	4	4	4	1	1
TrackedLocation		87%		71%	4	12	0	21	0	5	0	1
GPSStatusListener		97%		50%	3	7	1	13	1	5	0	1
GPSTrackingManager		100%		88%	1	12	0	35	0	8	0	1
GPSLocationService.LocationBinder		100%		n/a	0	2	0	2	0	2	0	1
Total	217 of 1.409	85%	54 of 126	57%	52	143	41	301	11	80	1	10

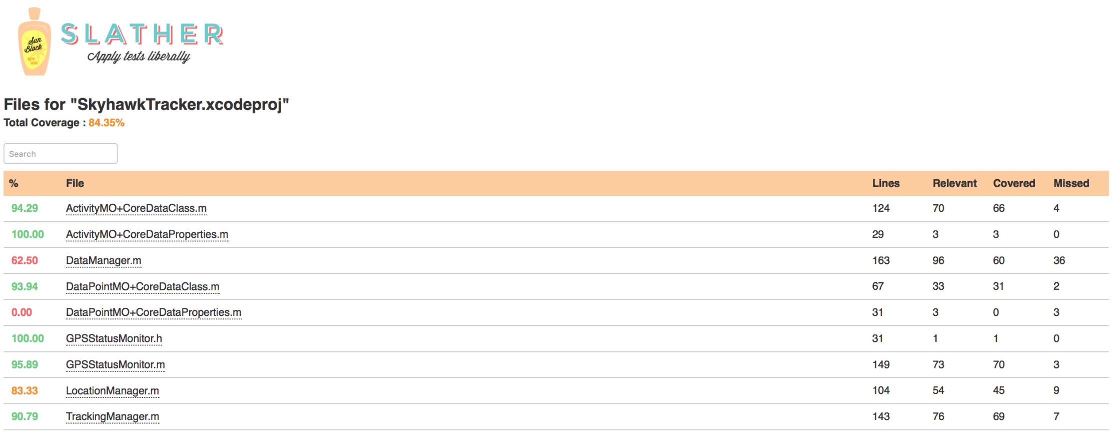
(b) Tracker

Figure 5.2: Test coverage Java implementation

OBJECTIVE C TESTING

For the Objective-C implementation, unit tests are created with the XCTest framework. The test coverage for the Objective-C code can be found in Figure 5.3 Classes which were difficult to unit test are the classes which depend on location data and on the database. Database testing was difficult because a real database of the smartphone OS was necessary. Therefore tests with the database are conducted using a local memory database during the test. To test the location providing a mocked LocationManager class is used to simulate new location point arrivals. This is not a completely covering test scenario because there are a lot of different scenarios possible in which locations can arrive, including different times in delay which are not all completely tested.

[2]Istanbul, `http://istanbul-js.org`
[3]PowerMock, `http://powermock.github.io`



Files for "SkyhawkTracker.xcodeproj"
Total Coverage : 84.35%

%	File	Lines	Relevant	Covered	Missed
94.29	ActivityMO+CoreDataClass.m	124	70	66	4
100.00	ActivityMO+CoreDataProperties.m	29	3	3	0
62.50	DataManager.m	163	96	60	36
93.94	DataPointMO+CoreDataClass.m	67	33	31	2
0.00	DataPointMO+CoreDataProperties.m	31	3	0	3
100.00	GPSStatusMonitor.h	31	1	1	0
95.89	GPSStatusMonitor.m	149	73	70	3
83.33	LocationManager.m	104	54	45	9
90.79	TrackingManager.m	143	76	69	7

Figure 5.3: Objective-C coverage generated with Slather

5.1.2. AUTOMATED TESTING

During the project, Travis is used as continuous integration server. Travis automatically builds the code and runs the unit tests. This gives a really fast and convenient way to see if the code is still without compilation errors and the unit tests are still passing.

5.1.3. USER TESTING

In the first week of the implementation phase, an initial working version was ready. This made it possible to start at an early stage with real world testing. Something that is necessary for live tracking, because of all the different scenario a user can encounter during an activity. This made it also possible to gather a lot of data and use this to test and improve the filtering of the location data. The user feedback was also useful to detect bugs or implementation problems at an early stage. These could then be solved relatively easy. Another reason for the need of user testing is because of a lot of different components which should work together. For example when you simply start tracking an activity. The user interacts with the React Native app which must make a call to a Typescript library which makes a call to a native iOS or Android implementation to start tracking. This is not easily testable in automated tests but is important for the stability of the developed feature.

To facilitate these tests on iOS Apple TestFlight[4] was used. Users were invited by sending them an e-mail. Once accepted they could easily download the application from the TestFlight app. Android, being less restrictive on apps without approval, the .apk file was sent to a group of users. iOS and Android automatically sent their results to Google FireBase[5], having a central location in which crash reports were gathered.

5.2. SIG EVALUATION

The Software Improvement Group (SIG) has evaluated the code twice. SIG is an international consultancy agency aimed at software improvement. After the first upload, SIG provided the team with feedback on how the maintainability of the code could be improved. These improvements are evaluated with the second evaluation.

FIRST FEEDBACK

The code was sent to SIG on the 31st of May, feedback was obtained on the 12th of June. The mail (in Dutch) received from SIG can be found in Appendix B. The code scored above average for maintainability. The highest score is not reached, because of a low score on code duplication. This was already improved with a refactor of the structure of the code before the feedback from SIG was obtained. The presence of test-code was according to SIG already promising, which is something we tried to keep at this level during the continuation of the project.

[4]Apple TestFlight, `https://developer.apple.com/testflight`
[5]FireBase, `https://firebase.google.com`

SECOND FEEDBACK

On June 26th, the code was sent to SIG for a second evaluation. On the 30th of June, the feedback was obtained, the mail (in Dutch) received from SIG can be found in Appendix B. The code scored higher for maintainability, although the size of the code increased. The amount of duplication decreased, but this was not enough to increase the overall score. SIG also noticed the amount of test code increased with the increased amount of development code. Overall SIG concludes the recommendations of the first evaluation have been taken into account in the development process.

5.3. REQUIREMENT ANALYSIS

At the end of the project, the team made a final analysis of the requirements. Each requirement is marked as completed (✓) or not completed (X). If a requirement is not completed an explanation is given.

5.3.1. MUST HAVES

Table 5.4 contains an analysis of the must haves. The project is considered a success since 100% of the must have requirements have been fulfilled.

Requirement	**Completed**
The implementation must be usable on Android SDK version 19 and iOS version 9.	✓
The data gathered by the implementation must be compatible with the current video rendering software of the client.	✓
The tracking feature must work when the app is not running in the foreground.	✓
Data should not be lost after the app is closed or crashed specifically during recording.	✓
The user must be able to start and stop tracking.	✓
The data must automatically be sent to a server of the client.	✓
The tracking functionality must gather longitude, latitude, altitude and time data.	✓
The applications should be able to track for at least 4 hours.	✓

Figure 5.4: Must haves

5.3.2. SHOULD HAVES

Table 5.5 contains an analysis of the should haves. 100% of the should have requirements has been successfully completed, improving the overall quality of the final product.

Requirement	**Completed**
The user should be able to pause the data gathering.	✓
The user should be warned when the application is started while location services are not available.	✓
The data should be stored in a storage efficient data format.	✓
The user should be able to select the sport he/she wants to track.	✓
The data gathering should automatically be paused when the user does not move.	✓
The data should be processed to improve the video rendering quality.	✓
The application should be able to track for at least 5 hours.	✓
The application should be able to continue a tracking activity after the application has been terminated.	✓

Figure 5.5: Should haves

5.3.3. COULD HAVES

Table 5.6 contains an analysis of the could haves. 43% of the could have requirements have been successfully completed. Due to time constraints, a robust and well-tested implementation of activity detection, real-time distance and including additional data was not possible. Dynamic changing of the sample rate was not required since modern phones can track location at a relatively high sample rate of 1 second without impacting battery life.

Requirement	**Completed**
The implementation could provide activity statistics.	✓
The implementation could automatically detect when the user is performing an activity.	X
The user could see his current speed in real time.	✓
The user could see the currently traveled distance in real time.	X
The rate at which location data is sampled could be adjusted dynamically.	X
The implementation could support additional data such as heart rate, step counter, cadence and calories.	X
The applications could be able to track for at least 6 hours.	✓

Figure 5.6: Could haves

5.3.4. WON'T HAVES

Table 5.7 contains an analysis of the won't haves. As expected, 100% of the won't haves have not been implemented.

Requirement	**Completed**
The implementation won't be compatible with the Windows Phone platform	✓
The implementation won't have to work with devices which have no GPS hardware.	✓

Figure 5.7: Wont haves

6 Process

6.1. Scrum

During the implementation phase, the team used a weekly scrum with a new working version every week. The team created an initial completely working version in the first week. This made it possible to iteratively improve this version every week. To have an overview of the backlog and tasks of the current sprint the team used Trello[1]. The project was executed at the client's office which makes it possible to have fast input about new features and design choices.

6.2. Github

Git is used in combination with Github for version control. For every new feature, the team created a new branch and used pull requests to merge the new changes in the master branch. Before a branch could be merged all the tests had to pass on Travis and some other team member had to approve the changes based on a code review. A private repository was provided by the client.

Issue tracker

The issue tracker on Github is used to keep track of current issues. This allowed the team and the client to directly see what issues were open and should be fixed.

6.3. Progress meetings

Every week a progress meeting with the client was held. During this meeting, the current state of the project was discussed. Also, the issues present on the issue tracker and plans for the coming weeks were discussed. Every two weeks we met with the TU Delft coach to discuss the current problems the team was facing and show the progress of the project.

[1]Trello, `https://trello.com`

7 Conclusion

During the 10 weeks of the project, the team developed a fully functional sports tracking module. The implementation is done as a separate React Native Typescript module which could be used in any React Native application.

The project started with two weeks of research about location tracking using smartphones. Already existing solutions, the accuracy of GPS data from smartphones and possibilities to improve location data are investigated. Based on the conducted research the requirements are created in cooperation with the client.
In the following 8 weeks, the development of the sports activity tracker is conducted. The team started with an initial completely working version and iteratively improved this version in the following weeks. The initial working version allowed the client and users to give early feedback to improve the implementation.
The main features of the product are location tracking and persistent data storage. The module allows retrieving accurate tracked location data accomplished using Kalman filtering. The location tracking can be used for different sports activities and it is possible to restore the tracking state after an app restart.

Concluding, the implemented module can be seen as a successfully implemented solution to provide a location tracking feature for sports activities.

8
Discussion

8.1. Requirement Reflections
The requirements were created in the first week of the development process. They are created based on the Research phase and in cooperation with the client. After three weeks the team decided in consultation with the client to remove one of the initial requirements. The module we created should no longer have the responsibility to send the data to the server but only the responsibility to retrieve and store the tracked data. In this way, we could create a completely independent module and give the users of the module the freedom to decide what to do with the data. All other must and should haves are implemented in the final version of the product. Not all could haves are implemented, but this has no influence on the success of the project. Lack of time is one of the main issues leading to these could haves being not all implemented. A requirement that should have been included is: The product should work over the whole world where a stable GPS signal is available. There were no indications this is not the case with the current implementation, but since this was not explicitly stated in the requirements no further research about this has been done.

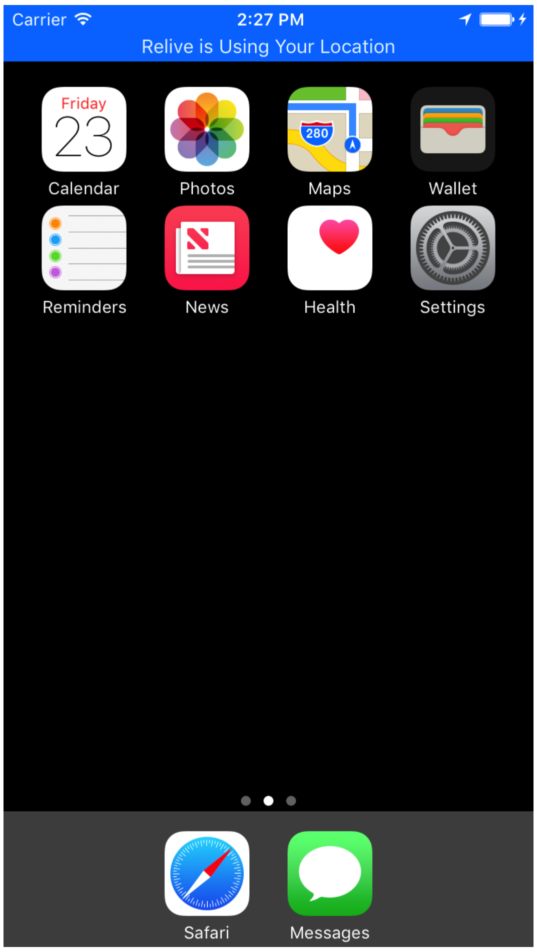
8.2. Process Reflection

Research Phase
The project started with the research phase in which the team read various papers about GPS tracking and improving location data. During this phase, the team documented all findings in a central document. The team discussed the findings every morning and decided what to focus on the coming day. After reading some papers about improving quality the team decided to create a simple test application to practically test the possibilities of GPS location retrieval.

Development Phase
During the development process, the team used the SCRUM methodology. This started by creating an initial working version in the first week. Improving this version every week was a successful strategy and led to an improved version every week. Difficulties encountered with this strategy were the differences in time needed to implement features for iOS and Android. This resulted in some features implemented in only one of the operating systems. Reasons for this was the before the project unknown programming language Objective-C. After the first few weeks, we accomplished to get both versions at the same level again. Other problems encountered during the project were the difficulties with non-compatible versions of different modules of React Native. Which slowed down the process a lot during the first weeks.

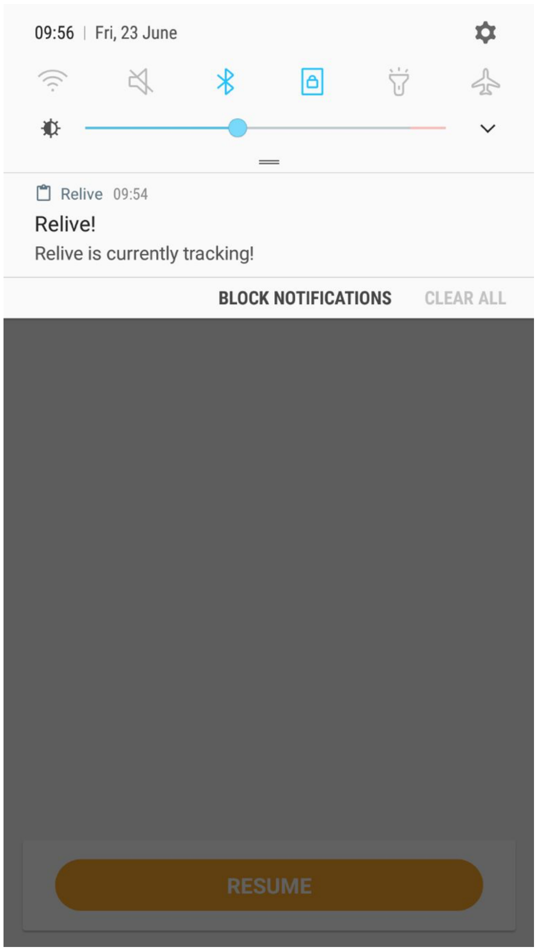
8.3. Ethical Implications
With privacy being a value of great importance to large groups of users, location tracking can become dangerous. Combining the fact that the client has a large user base and each tracked activity is stored and sent to a cloud environment it is important that user privacy is guarded. The company already has access to a large amount of data from different activity tracking providers. The experience of the team with the company shows that the company shows great value in their users, and abuse of collected locations is not likely to occur.

Since the developed tracking module is flexible in a sense that it can be used in any application, the team would like to stress the fact that deploying a location tracking solution in an application with a large user base must be done without violating rights and privacy of the end user. To make sure the user is always aware of location tracking, the tracker module implements a notification on Android and a blue bar on iOS which is always visible if location services are being used. This prevents tracking of location without the user being aware. Figure 8.1 shows these notifications on both platforms.



Relive is Using Your Location

(a) Blue Location bar on iOS.



Relive 09:54
Relive!
Relive is currently tracking!

(b) Notification on Android.

Figure 8.1: Overview of location service notifications on iOS and Android.

9 Recommendations

The team was able to create a fully working version of an activity tracking feature. There are however still improvements possible. Currently, the implementation is not able to automatically detect a user performing a sports activity, or detecting what type of activity is being executed. Another improvement is enhancing the quality of location data. As appeared from the research phase there are various possibilities to improve the location data, for instance by mapping them to a road. Due to the lack of time, the team has not implemented them. Another improvement could be the support for additional data such as heart rate, step counter or cadence. Currently, additional metadata such as average speed and moving time are available for each activity. This metadata is currently not used while rendering videos but is available in the module.

Although the product is tested with real world users, testing with a lot more users is recommended. Almost all conducted tests were performed in the Netherlands, or in close proximity to the Netherlands. Since the location tracking feature is influenced by the physical location of the users it would be useful to test with users from all over the world. Edge cases that should be considered are places with high mountains, ravines, north and south pole and places around the equator. Since all these things may influence the results of the filtered data.

A
Initial Project Description

Company description

Relive turns outdoor adventures into short video animations that people love to share with family and friends. Relive is all about the total experience of your trip. No focus on performance metrics and competition. Let's leave that to traditional sports trackers. Since April Relive has been growing extremely fast. At the moment, we're over 250,000 users, including Laurens ten Dam, Lance Armstrong and Instagram founder Kevin Systrom; great people. Each month we're processing more than 1 million videos. Right now, exciting new steps in our product development are being made. The ideas for future releases are endless. It's great. This is where you come in. GPS tracking feature Relive collects the user's GPS data from third party trackers (e.g. Strava). For many users it would be convenient to have a tracking feature within the Relive app.

Problem definition

Develop a mobile GPS tracking feature for our app. Research and problem analysis What is activity tracking in general? How does it work in the most ideal and simple situation? What can possibly go wrong when recording an activity? How can this be handled? Think of situations like device shutdown (dead battery), no GPS signal, app crash and restart, etc. What variables depend on activity type? (e.g. recording interval). How can GPS tracking functionality be integrated into our current solution? Think of development efficiency, product reliability, and user experience. Mobile phones are constantly sensing movement in the background. Can this be leveraged to improve the quality of our product? Think of auto-recording, improved battery usage, identifying interesting moments during activity recording, etc. Process During the requirements phase you'll be finding answers to questions like these. How do the requirements compare to existing solutions? Can existing components be used so the project can be focused on developing smarter solutions? Or should development start at the basics? Discuss your intermediate findings on a regular basis with our team and come up with a technical design of the product that is realistic to develop during the project.

B
Raw SIG Feedback

First Feedback
De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Duplication. Voor Duplication wordt er gekeken naar het percentage van de code welke redundant is, oftewel de code die meerdere keren in het systeem voorkomt en in principe verwijderd zou kunnen worden. Vanuit het oogpunt van onderhoudbaarheid is het wenselijk om een laag percentage redundantie te hebben omdat aanpassingen aan deze stukken code doorgaans op meerdere plaatsen moet gebeuren. In dit systeem is er duplicatie te vinden tussen getAllActivities en getActivitiesByFinished in DataManager.java. Het is beter om het gedeelde stuk naar een aparte methode te verplaatsen en deze vervolgens aan te roepen. De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt. Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

Second Feedback
In de tweede upload zien we dat zowel de omvang van het systeem als de score voor onderhoudbaarheid is gestegen. Op het niveau van de deelscores zien we een verbetering op het gebied van Duplication. Deze verbetering is echter net niet structureel genoeg om de totaalscore naar 4,5 ster te laten stijgen. Daarnaast is het goed om te zien dat jullie naast nieuwe productiecode ook aandacht hebben besteed aan het schrijven van nieuwe testcode. Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

C
INFOSHEET

This appendix contains the infosheet of the project, providing the reader with a short overview of the project and its stakeholders.

PROJECT DETAILS

Project Information	Collecting GPS Location data for sports visualization
Client	Relive B.V.
Final Presentation	Monday July 10, 2017

Description

Relive B.V. is a company which creates videos which users can watch to experience their running or cycling activity again. Currently, the company depends on external data sources from which videos are created. The main challenge was developing a custom tracking solution resulting in videos which are visually pleasant to watch. During the research phase, the team found that GPS data can be very unreliable, with points having a very low accuracy. Due to this low accuracy processing is needed to make the data suitable for video rendering. A custom React Native module was designed, implementing location collection and processing behavior. This custom module can be integrated into the existing smart phone application of Relive. To process this data, a Kalman Filter was used. Building a custom React Native module was challenging since Javascript code had to communicate with a native Android and iOS interface. The final product is a module which can be imported in the existing Javascript code of the Relive app. It was tested using a small group of users. These users collected data, which was gathered in an Amazon S3 cloud environment. This data was later used to improve data processing and filtering behavior. Due to time constraints, the team was not able to test the performance of the filter in areas with mountains outside of a simulator. The team recommends Relive to test the solution in mountainous areas. Relive has shown great interest in the developed solution, and might introduce it to their users as a means of data collection in the future.

PROJECT TEAM

DORIAN DE KONING

Interests	*Formula Student Competition, Back-End development, Embedded Software*
Roles	Android Development, TypeScript Development, Software Testing
Contribution	Android data storage system, Interaction with client software, Overall system design, Android React-Native bridge interface

JOCHEM LUGTENBURG

Interests	*Mobile App development, Web development, Smart Home*
Roles	iOS Development, TypeScript Development, User Interface Development, Data processing
Contribution	Initial iOS location gathering and data storage setup, User Interface, Data processing: Kalman filter design and implementation.

BRYAN VAN WIJK

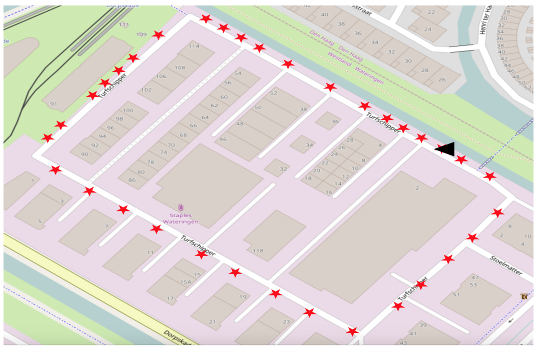
Interests	*Mobile App development, Web development*
Roles	iOS Development, Android Development, TypeScript Development, Software Testing
Contribution	React Front-End Development, iOS React-Native bridge interface, Android location gathering

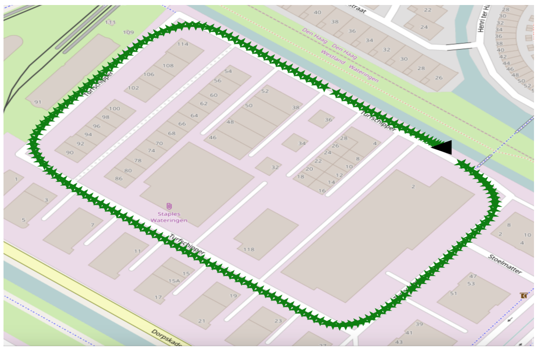
CONTACT INFORMATION

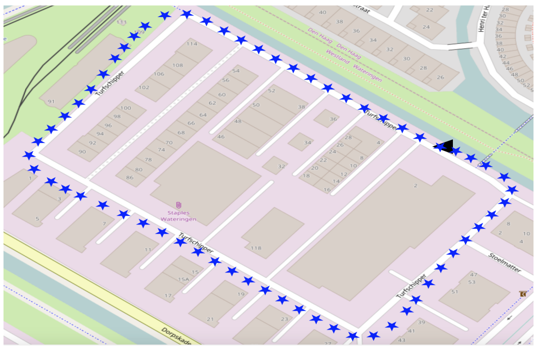
Team Members	Dorian de Koning (doriandekoning@gmail.com)
	Jochem Lugtenburg (jochemlugtenburg@gmail.com)
	Bryan van Wijk (bryanvanwijk@gmail.com)
TU Delft Coach	Dr. A. M. Zúñiga Zamalloa (Department of Software Technology)
Client	Ir. Ronald Steen (Relive B.V.)

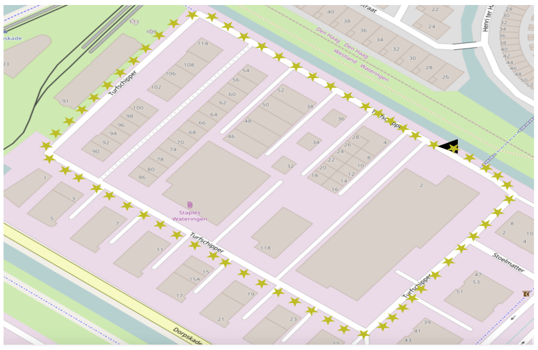
The final report for this project can be found at: http://repository.tudelft.nl

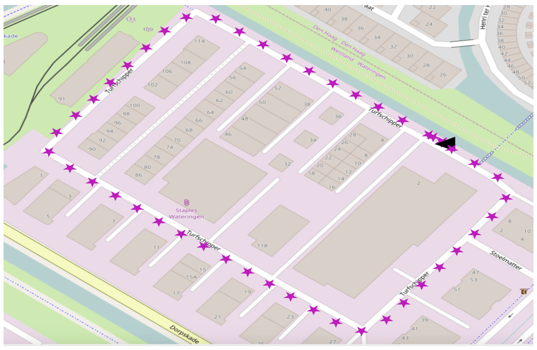
D
FIGURES


(a) Endomondo running


(b) Fitbit running


(c) MapMyRun running


(d) Runkeeper running


(e) Runtastic running

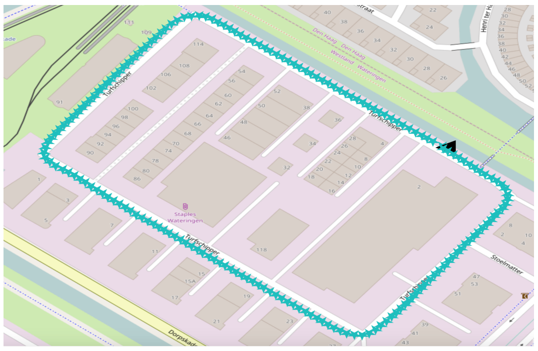
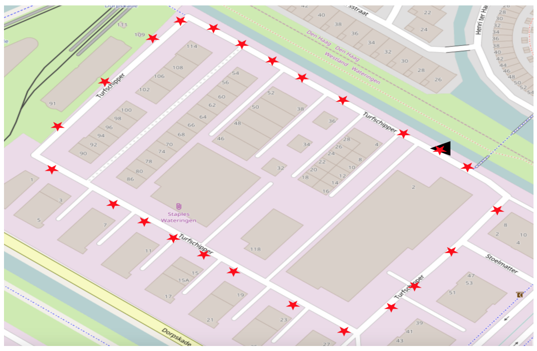
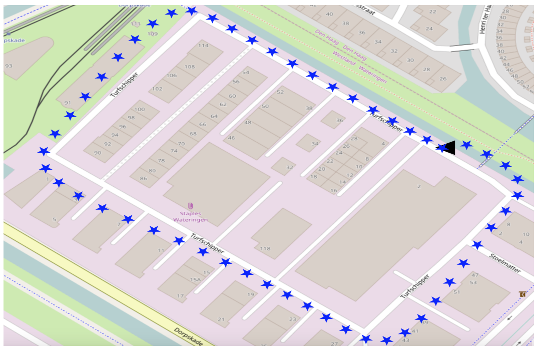
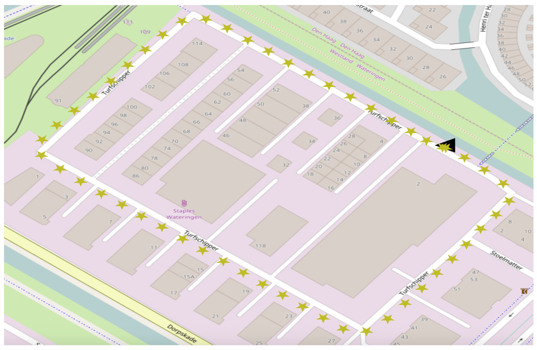
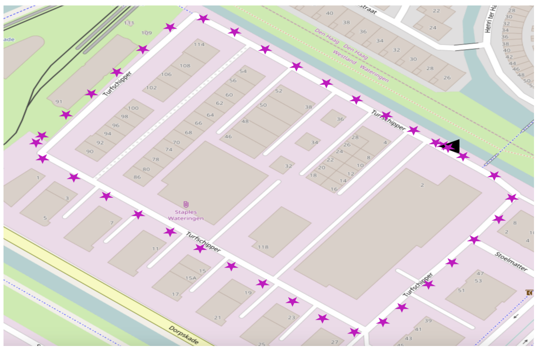

(f) Strava running

Figure D.1: Same travelled path with approximately 15-16 km/h using different android apps.
All paths were traversed on a bike to make it more easy to keep the speed constant. ● Endomondo, ● Fitbit, ● MapMyRun, ● Runkeeper, ● Runtastic, ● Strava, ▲ Start position


(a) Endomondo cycling


(b) MapMyRun cycling


(c) Runkeeper cycling


(d) Runtastic cycling

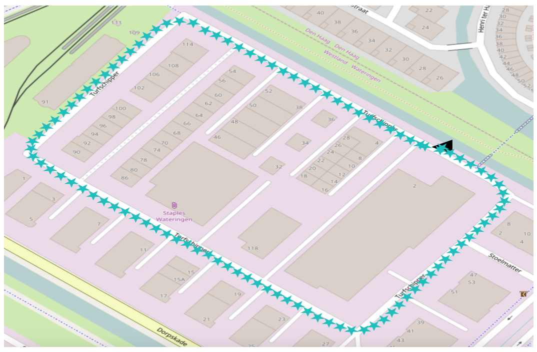
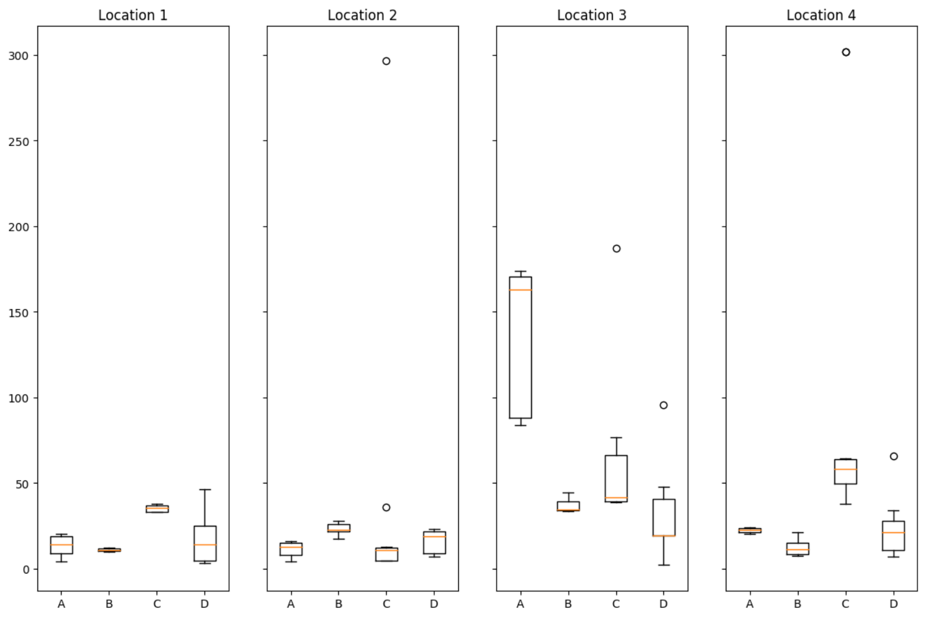
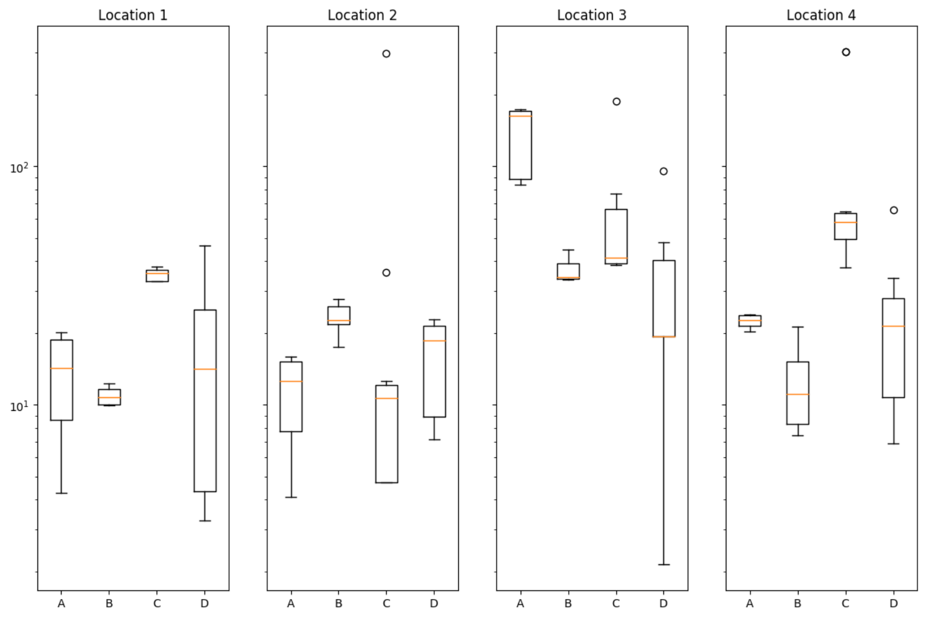

(e) Strava cycling

Figure D.2: Same travelled path with approximately 26-27 km/h using different android apps.
All paths were traversed on a bike to make it more easy to keep the speed constant. ● Endomondo, ● MapMyRun, ● Runkeeper, ● Runtastic, ● Strava, ▲ Start position

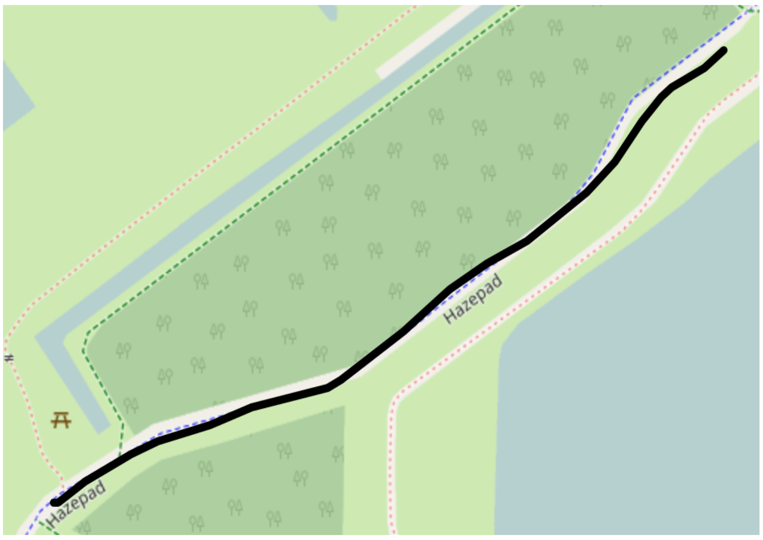


(a) Boxplot modeling phone location data on a linear scale.

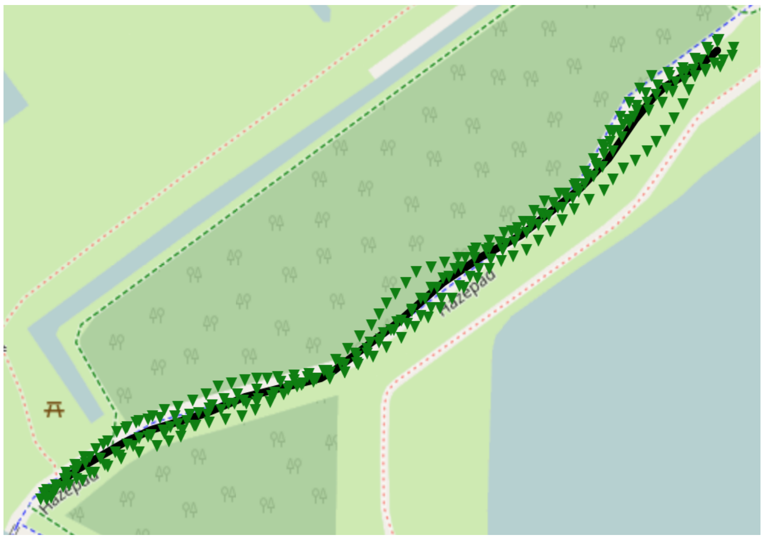


(b) Boxplot modeling phone location data on a logarithmic scale.

Figure D.3: Boxplots modeling distances to the reference positions. A: Samsung Galaxy S8+, B: Motorola XT1092, C: Apple iPhone 5, D: Apple iPhone 6

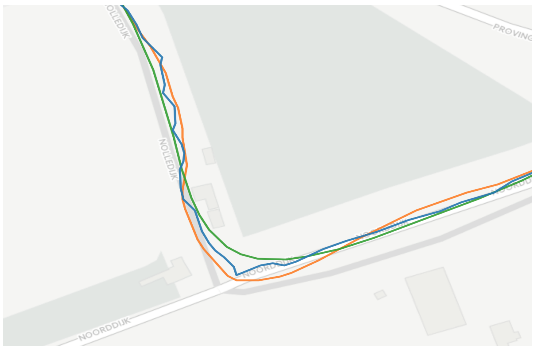


(a) Average line

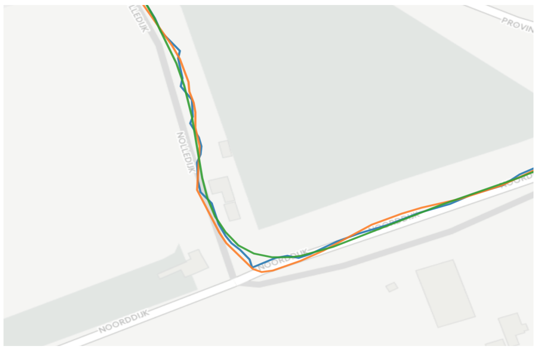


(b) Average line including original points

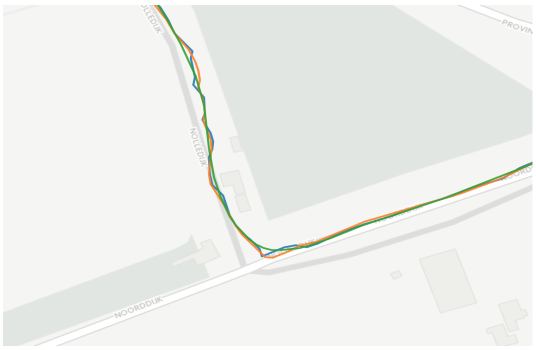
Figure D.4: Using time based interpolation on multiple tracked paths and averaging those points to estimate the correct path.
● Average line, ● original points,



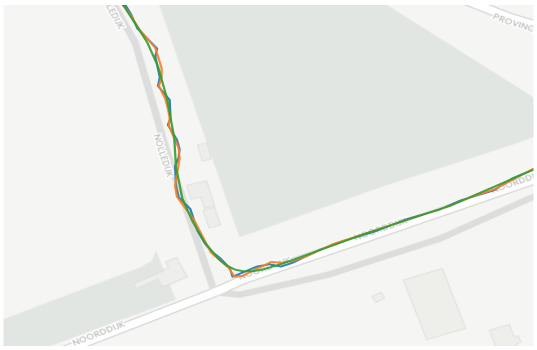
(a) Q variance 0.01



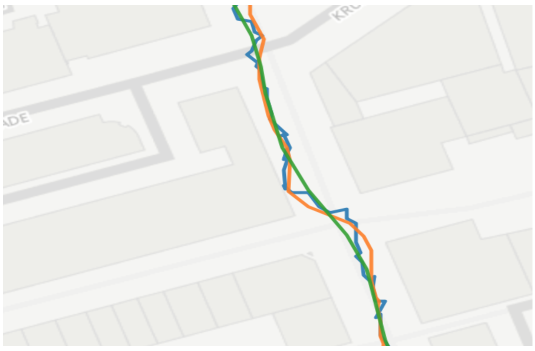
(b) Q variance 0.1



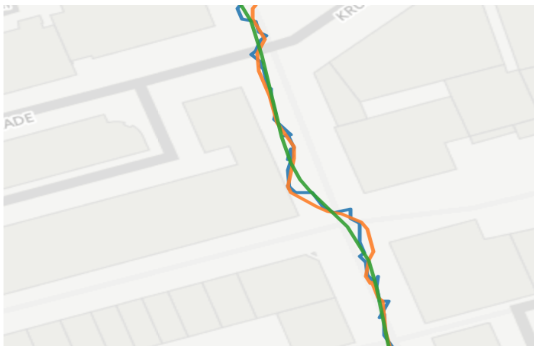
(c) Q variance 1.0



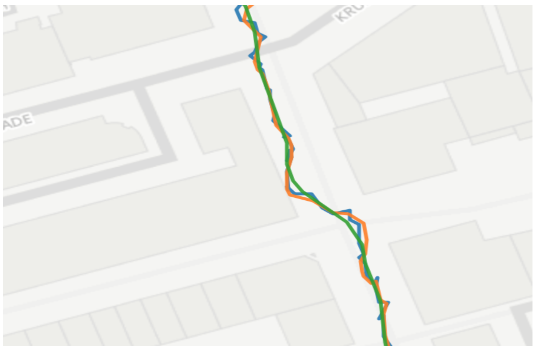
(d) Q variance 10.0



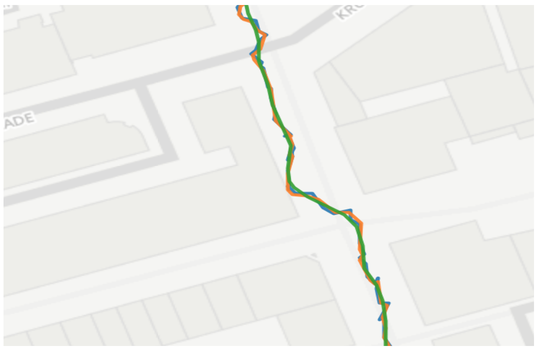
(e) Q variance 0.01



(f) Q variance 0.1

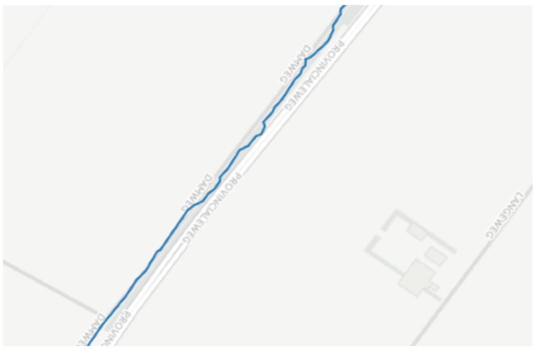


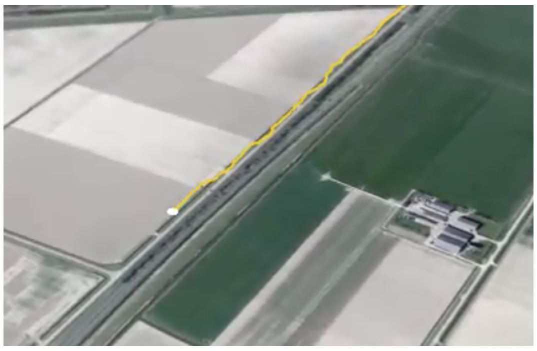
(g) Q variance 1.0

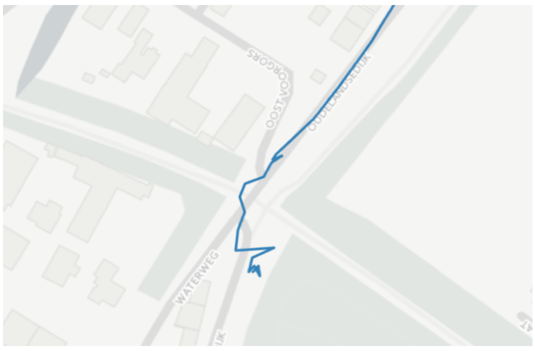


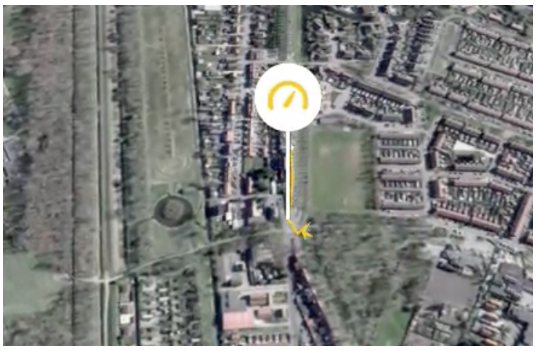
(h) Q variance 10.0

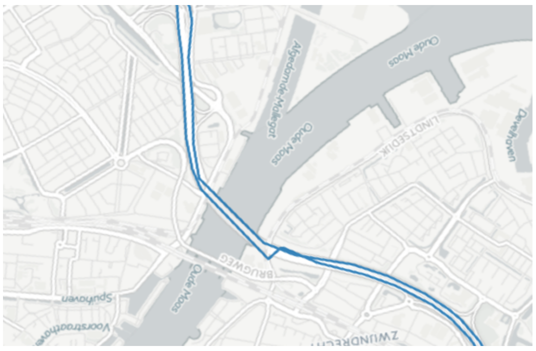
Figure D.5: Overview of different values of Q. Green represents the filtered and smoothed output, Orange represents the filtered data, Blue represents the unfiltered raw data.

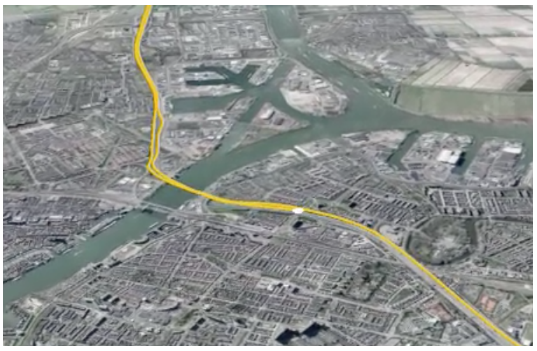

(a) Data Sample 1 - iPhone 6

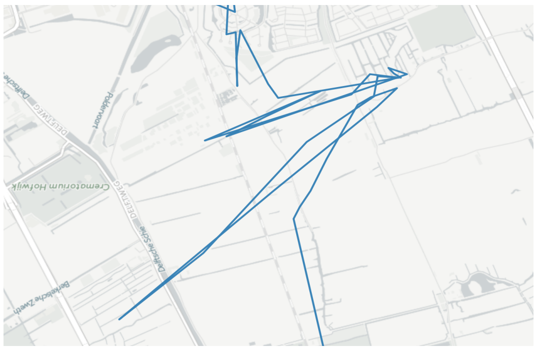

(b) Data Sample 1 - Video


(c) Data Sample 2 - iPhone 5C


(d) Data Sample 2 - Video


(e) Data Sample 3 - Samsung Galaxy S7 Edge


(f) Data Sample 3 - Video


(g) Data Sample 4 - iPhone 7

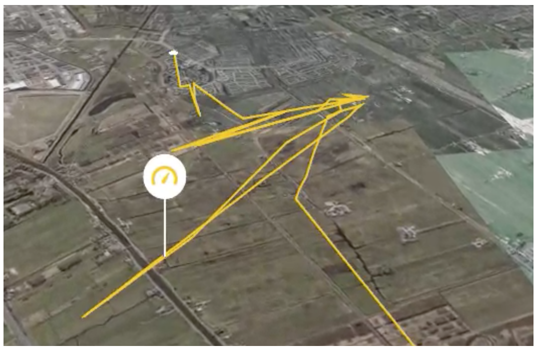
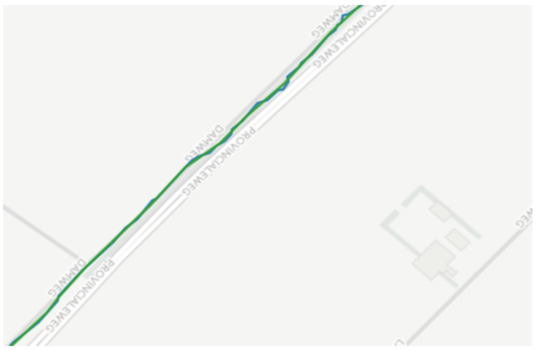
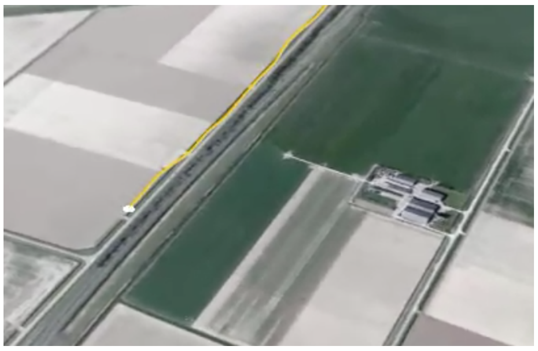

(h) Data Sample 4 - iPhone 7 - Video

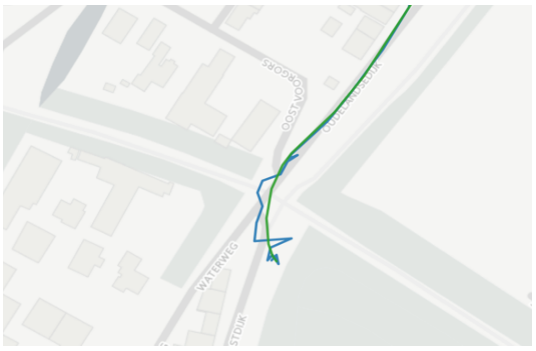
Figure D.6: Overview of unprocessed data and generated videos.



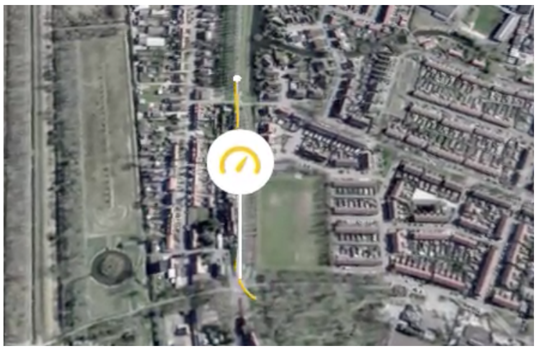
(a) Data Sample 1 - iPhone 6 - Filtered



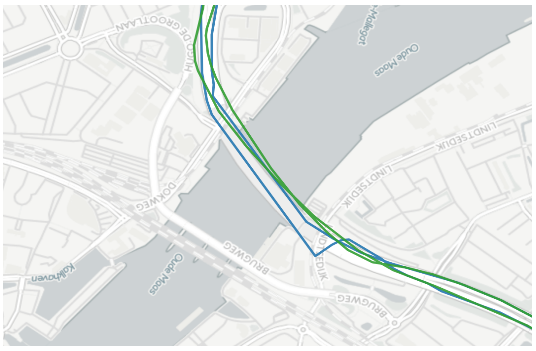
(b) Data Sample 1 - Video - Filtered



(c) Data Sample 2 - iPhone 5C - Filtered

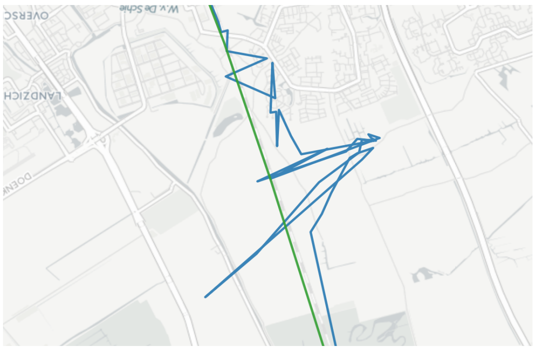


(d) Data Sample 2 - Video - Filtered



(e) Data Sample 3 - Samsung Galaxy S7 Edge - Filtered

(f) Data Sample 3 - Video - Filtered



(g) Data Sample 4 - iPhone 7 - Filtered

(h) Data Sample 4 - iPhone 7 - Video - Filtered

Figure D.7: Overview of processed data and generated videos. Blue represents the unprocessed data, Green represents the output after applying the filter.

E

SOFTWARE IMPLEMENTATION DIAGRAMS

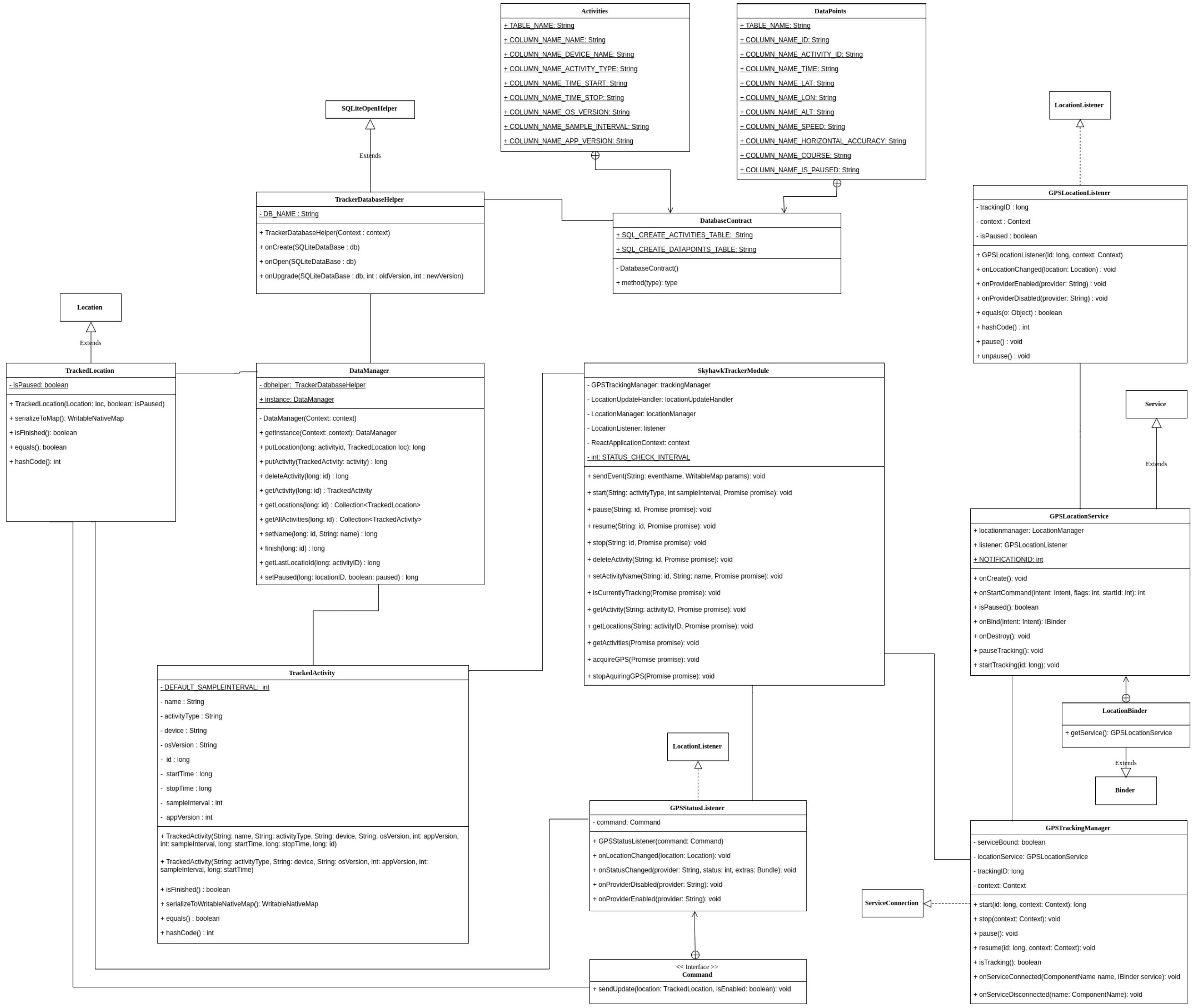

Activities
+ TABLE_NAME: String
+ COLUMN_NAME_NAME: String
+ COLUMN_NAME_DEVICE_NAME: String
+ COLUMN_NAME_ACTIVITY_TYPE: String
+ COLUMN_NAME_TIME_START: String
+ COLUMN_NAME_TIME_STOP: String
+ COLUMN_NAME_OS_VERSION: String
+ COLUMN_NAME_SAMPLE_INTERVAL: String
+ COLUMN_NAME_APP_VERSION: String
DataPoints
+ TABLE_NAME: String
+ COLUMN_NAME_ID: String
+ COLUMN_NAME_ACTIVITY_ID: String
+ COLUMN_NAME_TIME: String
+ COLUMN_NAME_LAT: String
+ COLUMN_NAME_LON: String
+ COLUMN_NAME_ALT: String
+ COLUMN_NAME_SPEED: String
+ COLUMN_NAME_HORIZONTAL_ACCURACY: String
+ COLUMN_NAME_COURSE: String
+ COLUMN_NAME_IS_PAUSED: String
LocationListener
SQLiteOpenHelper
Extends
TrackerDatabaseHelper
- DB_NAME : String
+ TrackerDatabaseHelper(Context : context)
+ onCreate(SQLiteDataBase : db)
+ onOpen(SQLiteDataBase : db)
+ onUpgrade(SQLiteDataBase : db, int : oldVersion, int : newVersion)
DatabaseContract
+ SQL_CREATE_ACTIVITIES_TABLE: String
+ SQL_CREATE_DATAPOINTS_TABLE: String
- DatabaseContract()
+ method(type): type
GPSLocationListener
- trackingID : long
- context : Context
- isPaused : boolean
+ GPSLocationListener(id: long, context: Context)
+ onLocationChanged(location: Location) : void
+ onProviderEnabled(provider: String) : void
+ onProviderDisabled(provider: String) : void
+ equals(o: Object) : boolean
+ hashCode() : int
+ pause() : void
+ unpause() : void
Location
Extends
TrackedLocation
- isPaused: boolean
+ TrackedLocation(Location: loc, boolean: isPaused)
+ serializeToMap(): WritableNativeMap
+ isFinished(): boolean
+ equals(): boolean
+ hashCode(): int
DataManager
- dbhelper: TrackerDatabaseHelper
+ instance: DataManager
- DataManager(Context: context)
+ getInstance(Context: context): DataManager
+ putLocation(long: activityid, TrackedLocation loc): long
+ putActivity(TrackedActivity: activity) : long
+ deleteActivity(long: id) : long
+ getActivity(long: id) : TrackedActivity
+ getLocations(long: id) : Collection<TrackedLocation>
+ getAllActivities(long: id) : Collection<TrackedActivity>
+ setName(long: id, String: name) : long
+ finish(long: id) : long
+ getLastLocationId(long: activityID) : long
+ setPaused(long: locationID, boolean: paused) : long
SkyhawkTrackerModule
- GPSTrackingManager: trackingManager
- LocationUpdateHandler: locationUpdateHandler
- LocationManager: locationManager
- LocationListener: listener
- ReactApplicationContext: context
- int: STATUS_CHECK_INTERVAL
+ sendEvent(String: eventName, WritableMap params): void
+ start(String: activityType, int sampleInterval, Promise promise): void
+ pause(String: id, Promise promise): void
+ resume(String: id, Promise promise): void
+ stop(String: id, Promise promise): void
+ deleteActivity(String: id, Promise promise): void
+ setActivityName(String: id, String: name, Promise promise): void
+ isCurrentlyTracking(Promise promise): void
+ getActivity(String: activityID, Promise promise): void
+ getLocations(String: activityID, Promise promise): void
+ getActivities(Promise promise): void
+ acquireGPS(Promise promise): void
+ stopAquiringGPS(Promise promise): void
Service
Extends
GPSLocationService
+ locationmanager: LocationManager
+ listener: GPSLocationListener
+ NOTIFICATIONID: int
+ onCreate(): void
+ onStartCommand(intent: Intent, flags: int, startId: int): int
+ isPaused(): boolean
+ onBind(intent: Intent): IBinder
+ onDestroy(): void
+ pauseTracking(): void
+ startTracking(id: long): void
LocationBinder
+ getService(): GPSLocationService
Extends
Binder
TrackedActivity
- DEFAULT_SAMPLEINTERVAL: int
- name : String
- activityType : String
- device : String
- osVersion : String
- id : long
- startTime : long
- stopTime : long
- sampleInterval : int
- appVersion : int
+ TrackedActivity(String: name, String: activityType, String: device, String: osVersion, int: appVersion, int: sampleInterval, long: startTime, long: stopTime, long: id)
+ TrackedActivity(String: activityType, String: device, String: osVersion, int: appVersion, int: sampleInterval, long: startTime)
+ isFinished() : boolean
+ serializeToWritableNativeMap(): WritableNativeMap
+ equals() : boolean
+ hashCode() : int
LocationListener
GPSStatusListener
- command: Command
+ GPSStatusListener(command: Command)
+ onLocationChanged(location: Location): void
+ onStatusChanged(provider: String, status: int, extras: Bundle): void
+ onProviderDisabled(provider: String): void
+ onProviderEnabled(provider: String): void
<< Interface >>
Command
+ sendUpdate(location: TrackedLocation, isEnabled: boolean): void
ServiceConnection
GPSTrackingManager
- serviceBound: boolean
- locationService: GPSLocationService
- trackingID: long
- context: Context
+ start(id: long, context: Context): long
+ stop(context: Context): void
+ pause(): void
+ resume(id: long, context: Context): void
+ isTracking(): boolean
+ onServiceConnected(ComponentName name, IBinder service): void
+ onServiceDisconnected(name: ComponentName): void

Figure E.1: UML Diagram of the Java class structure for Android.

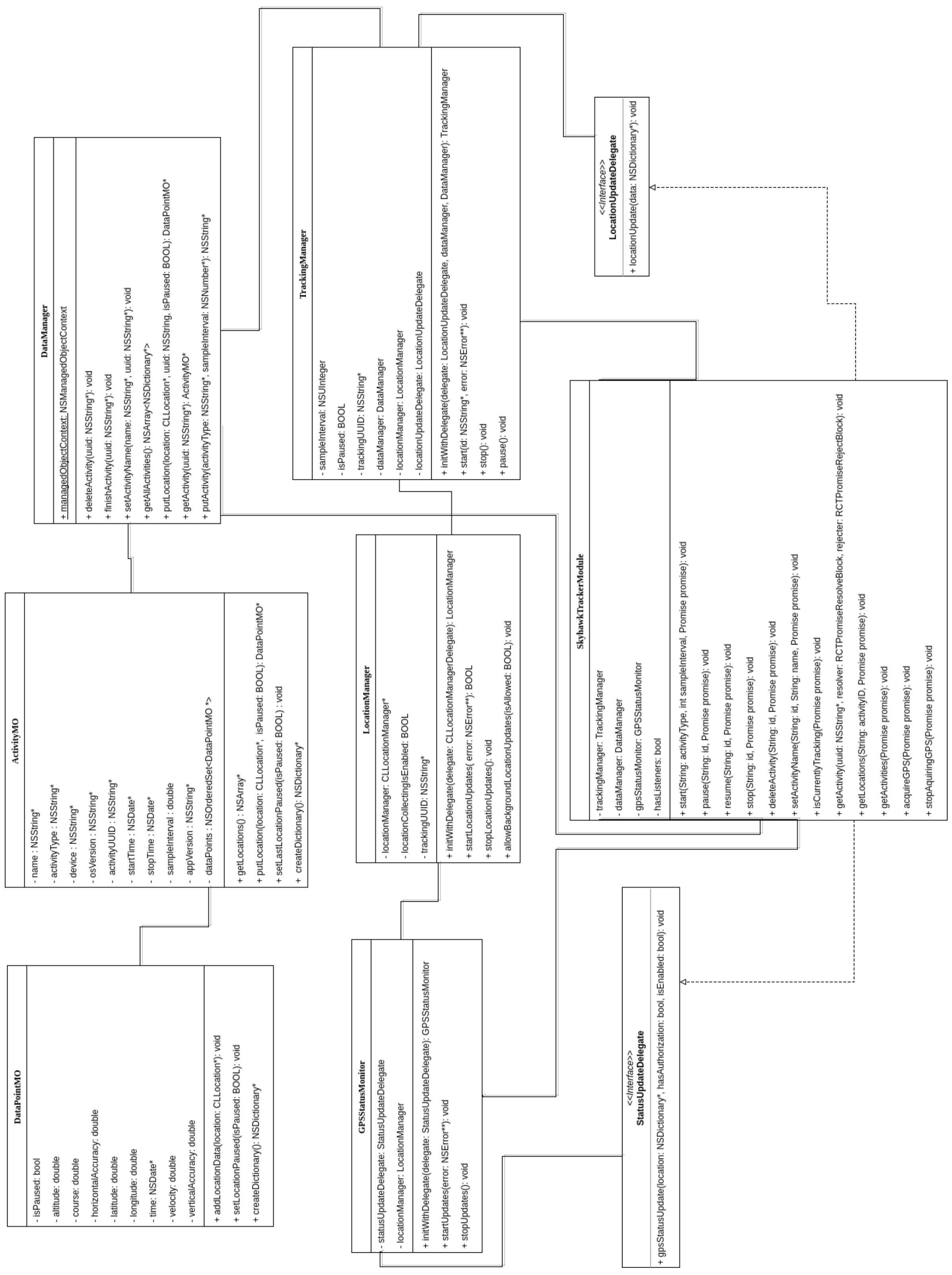

DataManager
+ managedObjectContext: NSManagedObjectContext
+ deleteActivity(uuid: NSString*): void
+ finishActivity(uuid: NSString*): void
+ setActivityName(name: NSString*, uuid: NSString*): void
+ getAllActivities(): NSArray<NSDictionary*>
+ putLocation(location: CLLocation*, uuid: NSString, isPaused: BOOL): DataPointMO*
+ getActivity(uuid: NSString*): ActivityMO*
+ putActivity(activityType: NSString*, sampleInterval: NSNumber*): NSString*
TrackingManager
- sampleInterval: NSUInteger
- isPaused: BOOL
- trackingUUID: NSString*
- dataManager: DataManager
- locationManager: LocationManager
- locationUpdateDelegate: LocationUpdateDelegate
+ initWithDelegate(delegate: LocationUpdateDelegate, dataManager, DataManager): TrackingManager
+ start(id: NSString*, error: NSError**): void
+ stop(): void
+ pause(): void
<<Interface>>
LocationUpdateDelegate
+ locationUpdate(data: NSDictionary*): void
ActivityMO
- name : NSString*
- activityType : NSString*
- device : NSString*
- osVersion : NSString*
- activityUUID : NSString*
- startTime : NSDate*
- stopTime : NSDate*
- sampleInterval : double
- appVersion : NSString*
- dataPoints : NSOrderedSet<DataPointMO *>
+ getLocations() : NSArray*
+ putLocation(location: CLLocation*, isPaused: BOOL): DataPointMO*
+ setLastLocationPaused(isPaused: BOOL) : void
+ createDictionary(): NSDictionary*
LocationManager
- locationManager: CLLocationManager*
- locationCollectingIsEnabled: BOOL
- trackingUUID: NSString*
+ initWithDelegate(delegate: CLLocationManagerDelegate): LocationManager
+ startLocationUpdates(error: NSError**): BOOL
+ stopLocationUpdates(): void
+ allowBackgroundLocationUpdates(isAllowed: BOOL): void
SkylawkTrackerModule
- trackingManager: TrackingManager
- dataManager: DataManager
- gpsStatusMonitor: GPSStatusMonitor
- hasListeners: bool
+ start(String: activityType, int sampleInterval, Promise promise): void
+ pause(String: id, Promise promise): void
+ resume(String: id, Promise promise): void
+ stop(String: id, Promise promise): void
+ deleteActivity(String: id, Promise promise): void
+ setActivityName(String: id, String: name, Promise promise): void
+ isCurrentlyTracking(Promise promise): void
+ getActivity(uuid: NSString*, resolver: RCTPromiseResolveBlock, rejecter: RCTPromiseRejectBlock): void
+ getLocations(String: activityID, Promise promise): void
+ getActivities(Promise promise): void
+ acquireGPS(Promise promise): void
+ stopAquiringGPS(Promise promise): void
DataPointMO
- isPaused: bool
- altitude: double
- course: double
- horizontalAccuracy: double
- latitude: double
- longitude: double
- time: NSDate*
- velocity: double
- verticalAccuracy: double
+ addLocationData(location: CLLocation*): void
+ setLocationPaused(isPaused: BOOL): void
+ createDictionary(): NSDictionary*
GPSStatusMonitor
- statusUpdateDelegate: StatusUpdateDelegate
- locationManager: LocationManager
+ initWithDelegate(delegate: StatusUpdateDelegate): GPSStatusMonitor
+ startUpdates(error: NSError**): void
+ stopUpdates(): void
<<Interface>>
StatusUpdateDelegate
+ gpsStatusUpdate(location: NSDictionary*, hasAuthorization: bool, isEnabled: bool): void

Figure E.2: UML Diagram of the Objective-C class structure for iOS.

F
Project Plan

F.1. Introduction

This document provides a global overview of the project executed for the TI3806 Bachelor project. First, an overview of the project will be given. Second, the involved stakeholders will be listed. Third an organizational overview of the project is given. Finally, we will conclude with a short overview of important quality aspects which have to be taken into account during the execution of the project.

F.2. Project Description

The client has provided the team with a problem definition during meetings in the weeks before the start of the project. This section summarizes this problem definition.

F.2.1. Client description

To make the reader familiar with the company, a short description of Relive has been provided by the client.

> Relive turns outdoor adventures into short video animations that people love to share with family and friends. Relive is all about the total experience of your trip. No focus on performance metrics and competition. Let's leave that to traditional sports trackers.
>
> Since April Relive has been growing extremely fast. At the moment, we're over 400,000 users, including Laurens ten Dam, Lance Armstrong, and Instagram founder Kevin Systrom; great people. Each month we're processing more than 1 million videos.
>
> Right now, exciting new steps in our product development are being made. The ideas for future releases are endless. It's great. This is where you come in.

F.2.2. Project Definition

Relive collects GPS data from third party trackers such as Strava[1] and Garmin Connect[2]. Recently Relive has released a mobile application, providing easy access to previously generated video animations. It would be convenient for Relive to have a tracking feature within the Relive app making the company less dependent on other companies for data collection. This tracking feature should be reliable, in the sense that it should be accurate and be able to deal with different forms of failure such as device shutdown, app crash, and an unreliable GPS signal. It should also be flexible, in the sense that it works under different conditions and sports activities. Also important are aspects such as battery usage, sensing interesting moments during an activity and the overall user experience. The team should investigate existing solutions and decide if existing components can be used, or new smarter solutions have to be developed. On a regular basis, the team should discuss intermediate findings with Relive and come up with a technical design of the product that is realistic to develop during the project.

[1] Strava Inc, `http://www.strava.com`
[2] Garmin Ltd, `https://connect.garmin.com`

F.2.3. Project Goal
The development of a mobile GPS tracking feature compatible with the Relive app on iOS and Android.

F.2.4. Involved Stakeholders
Relive
Ronald Steen will represent the interests of Relive during the project. He will represent the client within the Bachelor Committee. The main interest of Relive is having a working GPS tracking solution that can be deployed within the existing Relive Application. This solution has to be proven to be robust and reliable to be used in a production environment.

Delft University of Technology
Marco Zuniga represents the educational interests of the TU Delft. Representing TU Delft within the Bachelor Committee in the role of TU Coach he will act as a guardian of the educational interests of the university. He acts as a counterweight to the interests of the client. He is also responsible for giving feedback to the students on the research report, final report and final presentations.

Students
Dorian de Koning, Jochem Lugtenburg, and Bryan van Wijk are the team executing the project. Their interest is successful completion of the project. To achieve this, the interests of both Relive and TU Delft have to be met. For the team, this is a fun and interesting way to apply knowledge gathered during the Bachelor program.

F.3. Project Organization
This section provides a top-down overview of the entire process which spans ten weeks.

F.3.1. Scrum
During this project we will use Scrum[3] to work in an iterative way. We will use sprints of one week. A prioritized product backlog will be used to have a clear list of things to do. Every day we will have a scrum meeting in the morning to discuss shortly problems of the previous day and what needs to be done that day. In weekly Scrum sessions, the tasks for the next week are divided and a presentation is given to the client to show the result of the previous week. The goal of every sprint is to have at the end an improved working version of the software implementation. The weekly scrum meetings are documented to ensure no loss of important information about decisions made.

F.3.2. Planning
The project can be split up in two phases: two weeks of research followed by eight weeks of development. In the research phase, the possibilities for the GPS tracking feature will be investigated. After finding a feasible solution in the research phase this will be implemented in the development phase. The development phase will be split up in eight sprints of one week starting from May 8th. Each sprint begins on Monday with a short planning meeting and ends on Friday with an evaluation meeting. At least once every two weeks there is a meeting with the supervisor about the current state of the project.

- 2017, April 26 Meeting supervisor
- 2017, May 5 Research report done
- 2017, May 12 First working version
- 2017, June 1 First SIG submission
- 2017, June 26 Second SIG submission and deadline info sheet
- 2017, June 30 Final version completed
- 2017, July 10 Final presentation

[3] Scrum, https://www.scrumalliance.org

F.3.3. LOCATION

The team will work 4 days a week at the location of the client. The client's office is located in the Groot Handelsgebouw, unit E1.154. The Groot Handelsgebouw can be found next to Rotterdam Central Station (Stationsplein 45).

F.3.4. COMMUNICATION

Besides the physical meetings the team will use Slack[4] to communicate.

F.3.5. LEGAL

The team signed a Non Disclosure Agreement as issued by Relive, this agreement will not be made available. Included in this agreement is that Relive can ask the team to remove parts from the report before it is made publicly available. Parts will can only be removed from the report provided the report can still be considered correct and complete representation of the project.

F.4. QUALITY ASSURANCE

The Quality of the end product is very important to the success of the project. To ensure the quality the methods listed in the following subsections will be used.

F.4.1. AUTOMATED TESTING

Automated testing will be used to ease testing the application and to minimize the amount possible of bugs. A minimum coverage of 80% of the lines of code will be enforced throughout the whole project.

F.4.2. CONTINUOUS INTEGRATION

To ensure the code base remains well-tested and will build a continuous integration server.

F.4.3. GITHUB

The codebase will be stored and kept under version control using Git and GitHub. To make sure that all contributions are well reviewed in terms of code quality and functionality, strict criteria will be enforced. Before a piece of code can be merged into the stable master branch it has to:

- Pass all tests run by a Continuous Integration Server.
- Be approved by all team members.

GitHub will enforce these criteria, by preventing a merge if one of the conditions has not been met.

[4] Slack, https://slack.com

Bibliography

[1] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, *A survey of mobile phone sensing,* IEEE Communications Magazine **48**, 140 (2010).

[2] United States Air Force, *GPS: The Global Positioning System,* (2017).

[3] R. E. Kalman *et al.*, *A new approach to linear filtering and prediction problems,* Journal of basic Engineering **82**, 35 (1960).

[4] A. H. Mohamed and K. P. Schwarz, *Adaptive kalman filtering for ins/gps,* Journal of Geodesy **73**, 193 (1999).

[5] S. J. Julier and J. K. Uhlmann, *Unscented filtering and nonlinear estimation,* Proceedings of the IEEE **92**, 401 (2004).

[6] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, L. Girod, *et al.*, *Accurate, low-energy trajectory mapping for mobile devices.* in *NSDI* (2011).

[7] R. Gotsman and Y. Kanza, *Compact representation of gps trajectories over vectorial road networks,* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013) pp. 241–258.

[8] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, *Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones,* in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09 (ACM, New York, NY, USA, 2009) pp. 85–98.

[9] R. S. Babu, B. Radhakrishnan, and L. P. Suresh, *Detection and extraction of roads from satellite images based on laplacian of gaussian operator,* in *2016 International Conference on Emerging Technological Trends (ICETT)* (2016) pp. 1–7.

[10] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)* (Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006).

[11] D. C. Brabham, *Crowdsourcing as a model for problem solving: An introduction and cases,* Convergence **14**, 75 (2008).

[12] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, *Crowdsourcing with smartphones,* IEEE Internet Computing **16**, 36 (2012).

[13] C. Brunsdon, *Path estimation from gps tracks,* in *Proceedings of the 9th International Conference on GeoComputation* (National Centre for Geocomputation, Maynooth University., 2007).

[14] H. Li, L. Kulik, and K. Ramamohanarao, *Robust inferences of travel paths from gps trajectories,* International Journal of Geographical Information Science **29**, 2194 (2015).

[15] R. Labbe, *Kalman and bayesian filters in python,* `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/tree/d15845b` (2014).

[16] H. E. RAUCH, C. T. STRIEBEL, and F. TUNG, *Maximum likelihood estimates of linear dynamic systems,* AIAA Journal **3**, 1445 (1965).

[17] GIS Wiki, *Geodetic system,* (2013).