BACHELOROPPGAVE:

DriSMo - the driving quality application

FORFATTERE:
Fredrik Hørtvedt
Fredrik Kvitvik
Jørn André Myrland

DATO:
25.05.2011

Sammendrag av Bacheloroppgaven

Tittel:	DriSMo - kjørekvalitets-applikasjonen	Nr: -
		Dato: 25.05.2011
Deltakere:	Fredrik Hørtvedt	
	Fredrik Kvitvik	
	Jørn André Myrland	
Veiledere:	Simon McCallum, Høgskolen i Gjøvik	
	Patrick Bours, NISlab	
Oppdragsgiver:	Patrick Bours, NISlab	
Kontaktperson:	Patrick Bours, patrick.bours@hig.no, +47 61 13 52 50	
Stikkord	Bilkjøring, kvalitet, gps, akselerometer, java, android	
Antall sider: 126	Antall vedlegg: 8	Tilgjengelighet: Åpen

Kort beskrivelse av bacheloroppgaven:

DriSMo er en applikasjon utviklet for smarttelefoner og andre enheter på Android-plattformen. Applikasjonen benytter sensorer i enheten, for å gi tilbakemeldinger om kvaliteten på bilkjøringen til enhver tid. Tanken bak DriSMo er å gi brukerne muligheten til å sammenligne egne kjøreturer, både i forhold til tidligere målinger av seg selv og andre. Etter endt kjøretur vil man få en detaljert oversikt med avlesninger langs hele ruta, samt en oppsummerende poengsum.

Vi definerer kjørekvalitet som en målbar sum av sjåførens evne til å tilpasse sin bilkjøring etter omgivelsene, slik at komfortnivået maksimeres ut fra gjeldende forutsetninger.

Det er tilrettelagt for bruk og logging av GPS i DriSMo, slik at man får tegnet opp kjøreruta på et kart. Kjøreruta vil være fargelagt, med farger som representerer kjørekvaliteten på de aktuelle punktene. Denne kartvisningen er tilgjengelig underveis på turen, med kontinuerlige oppdateringer. Man kan også få oppsummert hele økta i en slik visning. Alle kjøreturer som er overvåket av applikasjonen, logges i egne filer slik at man enkelt kan gå i arkivet å se på tidligere turer. Disse filene kan også eksporteres til andre programmer, hvis det er ønskelig.

DriSMo ønsker å rette fokus mot kvalitetskjøring, ved å appellere til konkurranse-instinktet i brukerne. Hvis sjåføren fokuserer på å oppnå gode resultater i applikasjonen, kan det samtidig føre til redusert drivstoff-forbruk. DriSMo er i dag tilgjengelig via Android Market.

Summary of Graduate Project

Title:	DriSMo - the driving quality application	Nr: -
		Date: 25.05.2011
Participants:	Fredrik Hørtvedt	
	Fredrik Kvitvik	
	Jørn André Myrland	
Supervisor:	Simon McCallum, Høgskolen i Gjøvik	
	Patrick Bours, NISlab	
Employer:	Patrick Bours, NISlab	
Contact person:	Patrick Bours, patrick.bours@hig.no, +47 61 13 52 50	
Keywords	Driving, quality, gps, accelerometer, java, android	
Pages: 126	Appendixes: 8	Availability: Open

Short description of the main project:

DriSMo is an application developed for smartphones and other devices on the Android platform. The application uses sensors in the device, to give feedback on the driving quality at any given time. The idea behind DriSMo is to give the users a tool for quality measurement and comparison, either in relation to their own previous trips or others'. After a trip recording is completed, the user will be presented with a summarized average quality rating, along with a detailed overview.

We define the driving quality as a measurable sum of the driver's ability to adapt the driving according to the surroundings, so that the level of comfort is maximized based on the current conditions.

DriSMo is adapted for use with the device's GPS receiver, and will draw your trip path on a map. The path will be colored according to the driving quality at each set of coordinates. This map view is available with continuous updates in the live monitor, as well as a complete trip path in the overview afterwards. All trips that are being monitored by this application will be stored in separate log files, for easy retrospective browsing with the DriSMo trip viewer. It is also possible to export logs to external applications.

By appealing to the user's competitive instinct, DriSMo wants to contribute to increased focus on driving quality. If the driver focus on achieving good quality within the application, this can also lead to increased fuel efficiency. DriSMo is available through the Android Market.

Preface

We were very happy when we discovered that this bachelor assignment was released in the middle of November 2010. We were intrigued by the idea of making a mobile application to assess driving quality. After a meeting with the employer, Patrick Bours, we decided to take this assignment as our bachelor project.

We would like to thank all the beta testers of DriSMo, who have dedicated some of their time to help us improve the application; Anja Myrseth, Hallvard Westman, Lars Erik Pedersen, Torstein Andreassen, Jørgen Rolandsen, Nina Skramstad, Javier Garcia (ESP) and Mark Day (US).

We would also like to thank our supervisor, Simon McCallum, for leading us in the right direction at the start of our project, and for introducing us to both GNUplot and LaTeX. He also provided us with the Bachelor Thesis LaTeX template for Gjøvik University College.

Last, but not least, we would like to thank Patrick Bours for entering a part time role as supervisor, in addition to being the employer for the project.

Contents

List of Figures

List of Tables

1 Introduction

In recent years it has become more common to own a smartphone. As a result, we have seen the rise of the "app". Small applications that take advantage of the technologies that these smartphones provide. These apps are easily accessed, as most people carry their phones in their pockets most of the time. This provides an easier way to apply technology to everyday situations, and is also what initiated the DriSMo project.

1.1 Project description

The Driving Skill Monitor (DriSMo) was born from an idea of being able to compare your own driving skills to someone else's. The result was going to be simplified to a display of red, orange or green screen based on the quality of driving.

Even though the idea was based on skill comparison, it quickly became clear that we had to separate the skill and quality terms, as driving quality are determined by multiple factors. Driving skill also involve several aspects that makes it too complex to build into a standalone smartphone application. It would for example require extra equipment to follow eye movement, to see if the driver is using the mirrors properly. However, this does not mean we have to ignore skill as a factor altogether.

We have identified three important factors when it comes to driving quality; skill, vehicle and environment. DriSMo looks at all these factors combined, in regards to comfort and quality of driving. As driving skills are still vital for the quality outcome, we decided to keep the skill reference in the application name. To monitor the driving quality, we had to make the smartphone a passenger in the car. This way, it can observe motion with its accelerometer, to decide if different events on the road are comfortable or not.

When our employer - Patrick Bours - presented us with his idea, it did not take us long to initiate the process of expanding it. Our first thought was that we would like to log the collected data, in addition to giving real time feedback. This will let the driver focus on driving, and review

the results afterwards. When we made the decision to log the trip, we also made it possible to give the user more specific feedback at the end of the trip. To take the application even further, we also decided to include GPS data in the log. DriSMo does not only rate your driving quality, but also specifies *where* it was good or bad.

1.2 Potential problems

At the beginning of the project, we had some worries about things that could potentially slow down our progress. One of these issues was our initial lack of equipment. First of all, only one of the group members owned a car. This did not change during the project timeline. Secondly, we only had one single smartphone among us at that point in time. Our device was a HTC Desire, running Android 2.2 operating system. Gjøvik University College have some devices available to assist students in this kind of projects, so we were hoping to make use of that. Soon after the project started, one of the group members decided to invest in his own HTC Desire HD. When our supervisor supplied us with another Desire, we had a good start. We were also able to borrow some alternate devices in the most crucial parts of our timeline, but either way, we still only had a small range of products to work with.

To make sure our application measure driving quality properly, it was important to do a lot of testing during the development process. At the early stages we did this by testing on our own. In the beginning, the quality evaluation was a bit off the mark, and we had to adjust our quality algorithm accordingly. The problem with this kind of testing is that our own opinions will only get us so far. It is vital that the users agree with the results we display to them. Therefore we had to involve some beta testers to get our feedback evaluated from a bigger audience.

1.3 Targeted audience

The idea behind the DriSMo project was to provide a solution to end discussions of who is the better driver. We had some ideas that could potentially be used in professional domains as well, but it was decided that this was beyond the scope of the current project. A discussion of the various ways to include professional use appears in Section 7.2.

1.3.1 Application audience

DriSMo is aimed at private users, curious about their driving quality or looking to prove themselves as good drivers. Based on this audience, we decided to stick with the skill reference in the name, even if it only represent one out of the three factors that impact the results. By referring to skills, we think the application will have a better chance of reaching it's audience. The name is not misleading, as the users' skill contributes significantly to the final assessment of quality.

1.3.2 Report audience

We decided to write our report in English, with respect to a multilingual audience. Both our employer and supervisor prefer English over Norwegian, so that made the decision easier. As we are planning to make DriSMo open source, this document also has potential developers in mind. It will be a good introduction to our work, for those who show interest in the project. The report also targets the academic community at Gjøvik University College, in relation to software development, and of course the project examiner. Based on this audience, we expect at least basic technological knowledge from our readers, and our report is written with this in mind.

1.4 Project purpose

The purpose of this project was first and foremost to provide a solution that gives realtime driving quality feedback. This solution is distributed as a smartphone application, and works independently on the device.

Secondary, we hope that the application will lead to people striving to drive smoothly, and obtain better driving quality. The US Department of Energy estimates that "smooth driving contributes to increased fuel efficiency and rough or rash driving retards fuel efficiency by as much as 33%" [1]. Better fuel efficiency is good for the environment, and however marginal our contribution is, we are still proud to be a part of that.

1.5 Academic background

All three developers on this project have the same academic background. We are all studying software development, and have been programming mostly in C++ and Java. In addition to this, we have touched several other programming and scripting languages, and learned to adapt

our skills to new platforms. As none of us had previous experience working on mobile platforms, we needed to adapt our knowledge to this new environment.

We also have 20 ECTS[1] in software engineering included in our studies, and have relatively good knowledge about the most common Software Development Methods (SDM). Most of our knowledge in this field was still theoretical. This was our largest project yet, and a good opportunity for us to put theory into practice. Our background in software engineering was useful when approaching this project, and deciding what SDM to work with.

To make a good end product, it was important that we understood how the mobile technology relate to the real world. The mobile sensors reacts to different events in the car, and we had to be able to catch and interpret these events accurately. If we were unable to do this to a satisfactory degree, we would have failed our mission. Our car knowledge was varying, both in theory and practice, as only one of us owned a car. Fredrik Hørtvedt had to lead the way in this field. We also planned to use some external resources to fill our blanks, or offer supplementary knowledge.

1.6 Framework

1.6.1 Software Development Methodology

After weighing our options, we decided to work with an incremental development methodology. We came to this conclusion relatively fast, after discussing our vision for the project. While constructing a draft of potential features, we realised that our application would have many properties with no direct connection to each other. This opened the possibility for us to develop these features separately, in increments. Working with that kind of model made us agile, in terms of allowing our employer to share new ideas along the way. Since the project description was more of an idea than a set of requirements, it was not unlikely that new features would be requested. After deciding to work incremental, it was only natural for us to do iterations as well.

As mentioned in Section 1.5, we have all taken some courses in software engineering. Through this we built our theoretical knowledge about SDMs in general, with some emphasis on agility. As we know that Scrum is both incremental and iterative, this was the first methodology that came to mind. We are also aware that Scrum is a popular choice for software development projects today, and we could benefit from getting some experience working this way.

[1] European Credit Transfer and Accumulation System

After each of us had looked a bit closer at the Scrum methodology, it was time to discuss whether we wanted to go with it or not. All of us were initially very excited about doing it the Scrum way. This may have affected our research to a certain degree, but at least we took the time to look at the negatives before moving any further. Some of the things that came to mind was related to group size, working environment and experience from working together on other projects.

First of all, as we were only three developers, our structure with the scrum roles would not be ideal. It was possible, but some roles would need to overlap. E.g. one of the developers would need to take the role as Scrum Master. We thought this was manageable, and started to draft a product backlog for the project. As long as we had the initial idea in mind, our employer pretty much gave us free rein to add features on our own. After the backlog was populated, we scheduled a meeting (see Appendix B, meeting #3) with Patrick, to let him have his say on how we prioritize the different features. We got some good responses from him, and felt that we were on the right track with our features. However, we got the impression that he wanted us to make most decisions on our own, rather than him taking the typical Scrum Product Owner role. This was of course not ideal for us trying to work with Scrum, and a team that was arguably already too small to handle all aspects. We therefore concluded that we had to modify the framework to make it work to our advantage.

1.6.2 Implementation

With our previous experiences working together, combined with the planned working environment for this project, we decided not to implement the daily scrum meetings in it's traditional form. Instead, we combined our coffee breaks with small meetings throughout the day. Working so close to each other, we had good communication all the time, and managed a good structure without too many formalities. The same applied for the sprint review meeting. We transformed this into a more frequent activity. Since we did not have the typical demo meetings regularly with the product owner, we could just demo to each other pretty much on the fly. Again, we thought that this worked well in our circumstances.

Seeing as we would spend the vast majority of our working hours together, we were able to take a very agile approach to the tasks at hand. While figuring out how we would like to

implement Scrum, it became clear that what we were really doing was just borrowing a few elements, and created our own super agile über framework.s

As mentioned, we had already written the initial product backlog. After populating the backlog, we played some planning poker to set our time estimates before the meeting with Patrick. During this meeting we established what features that was most important in our employers view. The backlog, including the priority ratings from Patrick, was used as a reference in building the functional requirements, discussed in Section 2.1. The backlog served as an overview of the basic application features, but was not continuously modified, since we decided to deviate from our initial plan of Scrum implementation.

Ideally, our aim was to finish two features for each two week iteration. This was of course dependant on the size of each feature, so some discussions and quick estimates had to be done at the beginning of each iteration. As we did not demo for our employer after *every* iteration, we could allow extended dealines in some cases. We still had a pretty strict approach to our milestones. As we did not have much experience, it was also fair to assume that there were going to be some faulty estimates. However, this could not be used as an excuse to lower the work rate. Working in iterations of two weeks was preventive in regards to bad estimates, because we noticed it faster when we fell behind, and learned from our experiences more often.

1.6.3 Schedule

During the project pre-planning, we set up a Gantt diagram for our expected progress. As you can see page 13 in our pre-project report (Appendix D), this was only a high-level overview of the main phases in our development process. We also included some important milestones. Our parent structure consisted of logging data, analysing data and application development. We also included a time frame to improve our application, based on feedback after the beta release milestone. Apart from this, we applied our two week iterations to the diagram. As it stands, they have no specific content. This was determined gradually. Including the iterations in our schedule made it easier for us to make assumptions on how many feature implementations we could handle.

1.7 Project organization

Our group was assigned a room at the university college for the duration of the project. We shared the room with two other groups, but agreed on a schedule that left the room at our disposal for at least three days a week. This room was where we spent most of our time working on the project. By working face to face with each other, we achieved a good working environment. We had insight into each others code, so that everyone knew what went on, and could be of assistance or participate in taking decisions when needed.

The three days we had scheduled the room was also the three work days that were the absolute minimum for all group members. We agreed on a 9 - 16 schedule on working days, but both the number of days and working hours was to be expanded if necessary. If any disagreements occurred regarding working such overtime, we had constructed a rule document to resolve it. It did not take long before we started working overtime, but we had no problems doing this, without referring to any rules. You can find these rules written in Norwegian in our pre-project report at page 14 (Appendix D).

When considering who should get the role as project manager, we agreed that this was something all of us could benefit from experiencing. Therefore we decided to pass the role around on a four week basis at first. This structure was evaluated when all members had finished their trial. We decided do continue the rollover, instead of appointing a permanent leader. To keep our external communication structured, we also decided to appoint Fredrik Kvitvik as project contact. In addition to this, we asked IT services to create a project e-mail address, that distributes all incoming mail to all group members. They created drismo@hig.no, which was used as contact e-mail for the application, e.g. to get feedback from beta testers.

Simon McCallum was assigned as the supervisor for this project. He leads the Game Technology Lab and is responsible for multiple courses at GUC, including Mobile System Programming. This suited us well, as we had no experience with mobile development beforehand. Our employer, Patrick Bours, also offered to take a part time role as supervisor, since Simon would not be working 100% this semester. This was an advantage for us. Since they have different backgrounds, we had resources in even more fields that was of relevance to our project. The only possible downside to this structure, was that it could be difficult to arrange meetings on the

fly. We were hoping to avoid this issue, by planning well enough ahead. Sometimes it was not managable, but we were generally very happy with this setup either way.

1.8 Document structure

We have chosen to structure our report in eight different chapters. For a short description of each chapter, see below:

1. Introduction - In this chapter you can read about the background and purpose of the DriSMo project. We have also described how we plan to organize it, and what kind of framework we will be using in the development process.
2. Specification - This chapter describes the functional and supplementary requirements for the application.
3. Analysis - Describing our research, collection and analysis of accelerometer data.
4. Design - In this chapter we go through the basic architecture of DriSMo.
5. Realisation and implementation - In this chapter we will explain how we realised and implemented the application.
6. Testing and quality assurance - Our procedures for blackbox and whitebox testing is explained here. We also go through some of the test results.
7. Ending - In this chapter we evaluate our own work and the project in general. There is also a section on further development.
8. Appendices

2 Specification

In this chapter we will cover functional and supplementary requirements, in order to realise the application. The functional requirements is derived of the product backlog, created in the early stage of the project.

2.1 Functional requirements

In essence we are developing a mobile application to evaluate the driving quality. The application must have the ability to provide realtime quality feedback to the user, as well as the ability to record the trip. This way the user may view and analyse the trip in retrospect. The user must also be provided a way to manage (delete/rename/export) these trips.

In order to evaluate the driving quality, we must measure the forces the vehicle is exposed to. This means the smartphone must receive accelerometer data to do the evaluation. It should also be possible to log the location and speed of the vehicle. This feature requires that the smartphone is able to recive GPS data.

The application should be implemented with multiple monitors to display the realtime driving quality. One of the monitors should be primitive, just displaying a text on a coloured background representing the current quality. Another monitor should function as a quality meter, displaying a meter with colors representing quality. Based on the current quality, an arrow is pointing at the color to represent the current driving quality. A third monitor should display the current location of the vehicle on a map, followed by a coloured tail (representing the current quality) of where the vehicle has been.

Viewing a previously recorded trip should be implemented in three different ways:

- A graph displaying the different quality ratings for the duration of the trip.
- If GPS data is present, a map displaying a coloured driving route based on the quality.

- A concrete text summary of the trip.

A tutorial to help the user must be implemented. This tutorial should provide necessary information and explain features of DriSMo, helping users understand what DriSMo is all about. The tutorial should automatically start at the first launch of the application.

Providing the ability to automatically respond with a "I'm driving"-notice on incoming text messages or calls, have become more popular in car applications. We should therefore support this in DriSMo as well.

Settings must also be included in the application, managing different preferences. This will allow the user to tweak DriSMo to their needs.

2.1.1 Use case model

The use case model is illustrated in Figure 1.

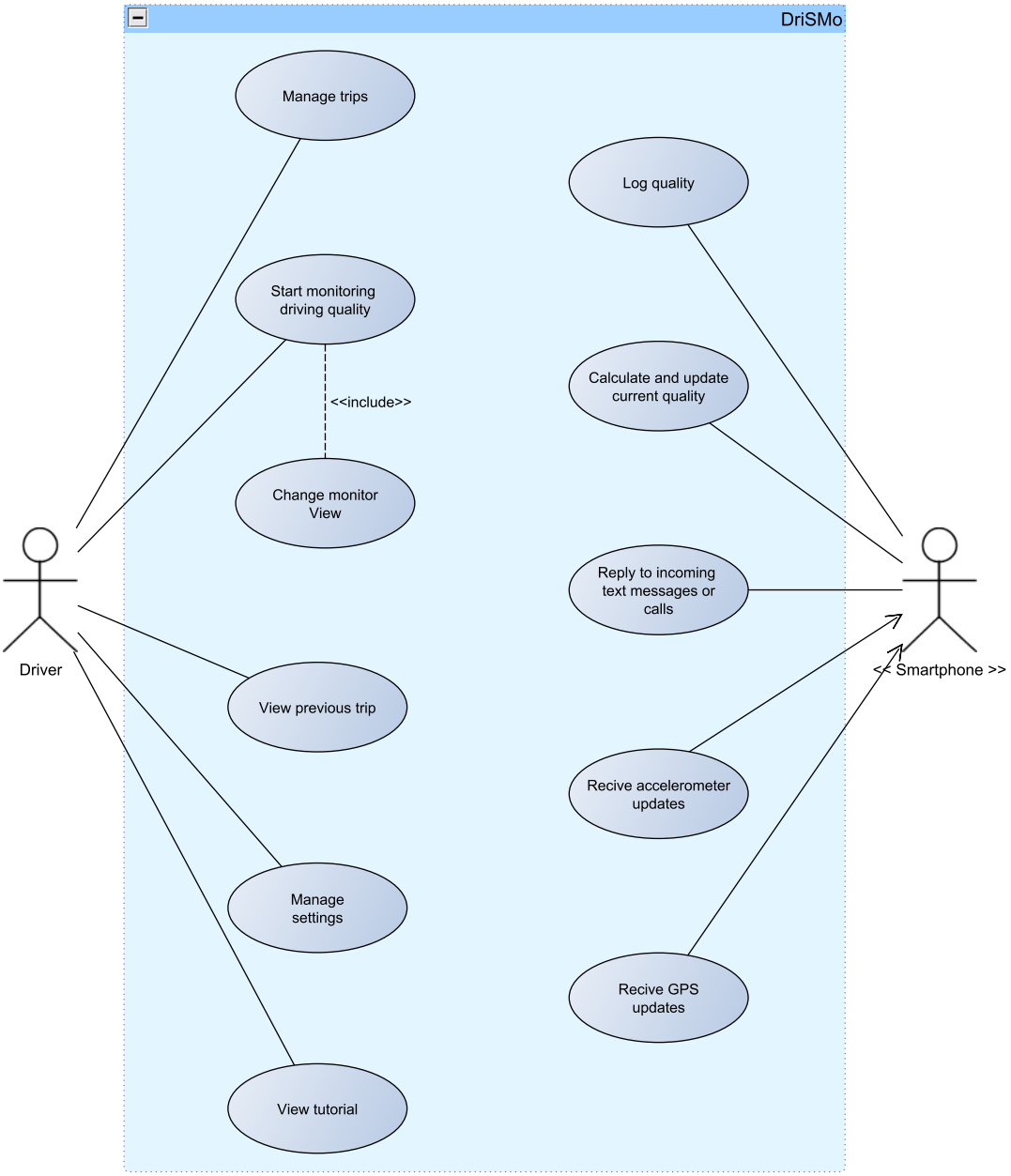


Figure 1: The figure shows the different use cases.

2.1.2 Highlevel use case

Use case:	View tutorial
Actor:	Driver
Goal:	The actor learns how to use DriSMo.
Normal flow:	1. The actor starts the tutorial via the main menu. 2. The actor follows the the tutorial.
Variations:	Starts automatically on the first run of DriSMo.

Use case:	View previous trip
Actor:	Driver
Goal:	View statistics of a trip and analyse the driving.
Normal flow:	1. The actor selects "view archive". 2. The actor selects a trip. 3. The actor can toggle between modes to view the trip: • View Quality Graph • View Quality Map • View Quality Text info
Variations:	The actor can also view a trip by selecting "View trip" after a long press on an item in the archive. If there is no GPS data, the map view is disabled.

Use case:	Manage settings
Actor:	Driver
Goal:	Allow the actor to configure the application.
Normal flow:	1. The actor opens the settings via the main menu. 2. The actor select/deselect the different settings.
Variations:	The "About" section has no configuration and is just for informational purpose.

Use case:	Manage trips
Actor:	Driver
Goal:	Rename, delete or export trip.
Normal flow:	1. The actor selects "View archive" via the main menu. 2. The actor selects a trip. 3. The actor chooses what to do with the trip: • View trip • Delete trip • Rename trip • Export trip
Variations:	If the actor tries to rename and the name already exists, the actor gets feedback about this. If the actor tires to export the trip, a list of possible apps. to handle the export, will be displayed. The actor can use the menu button to get the opportunity to delete all trips. If the actor chooses to delete one or all trips, a dialog with the current progress will be displayed.

Use case:	Start monitoring driving quality
Actor:	Driver
Goal:	The actor gets realtime feedback about the driving.
Normal flow:	1. The actor opens the "Monitor" via the main menu. 2. The phone starts to log the quality, and gives realtime feedback with the default monitor.
Variations:	If the GPS is off and the actor have set the application to log it, the actor will get the "Location settings" menu displayed. The user can toggle between monitors (see UC "Change monitor view").

Use case:	Change monitor view
Actor:	"Start monitoring driving quality", Driver
Goal:	Let the actor choose different monitors.
Normal flow:	1. The actor presses the "menu" button (within the monitor). 2. The actor select the preferred monitor.
Variations:	If the actor choose the map monitor and the GPS is off, the actor will get a warning.

Use case:	Log quality
Actor:	Smartphone
Goal:	Log the quality so it is available to the actor in retrospect.
Normal flow:	While the realtime monitoring is running, write the quality to a local log file.
Variations:	If the GPS is enabled write the current location to the log.

Use case:	Calculate and update current quality
Actor:	Smartphone
Goal:	Calculate the current score
Normal flow:	Calculate the current score, based on the values from the accelerometer.
Variations:	N/A

Use case:	Reply to incoming text messages or calls
Actor:	Smartphone
Goal:	Reduce driver distraction
Normal flow:	Auto replay on incoming SMS and calls, with a preset message.
Variations:	If this feature is not enabled, do nothing. If the preset message has already been sent to the specified receiver, it will not be sent again, unless it has been modified.

Use case:	Recive accelerometer updates
Actor:	Smartphone
Goal:	Get new values so we can calculate a new quality score
Normal flow:	Recive new values from the accelerometer.
Variations:	N/A

Use case:	Recive GPS updates
Actor:	Smartphone
Goal:	Get the current location
Normal flow:	Get the current location and speed.
Variations:	If GPS tracking is disabled, do nothing.

2.2 Supplementary requirements

2.2.1 System requirements

Platform

When developing an application for mobile devices, there are some different platforms to choose from. The most common mobile platforms are: Android, iOS and Symbian. Symbian is currently only shipped on Nokia devices. Since Nokia recently made a deal with Microsoft, Symbian will most likely vanish from the market in the near future. This leaves us with Android and iOS.

Android is on the rise and have overtaken iOS in the U.S smartphone market, according to Nielsen's article [2]. This seems like a global trend according to a research report from Gartner

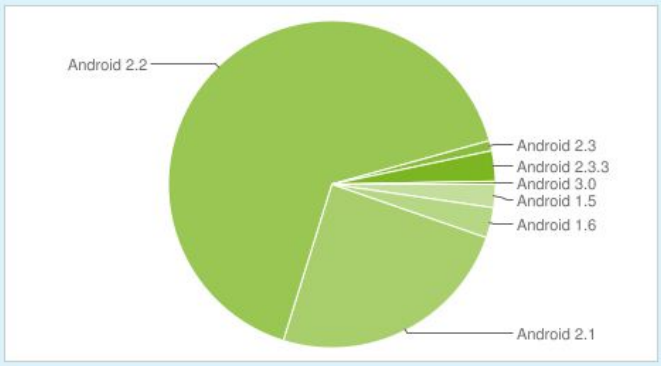
predicting the future of the smart phone market [3]. They predict that Android will account for 49.2% of the global smartphone market, and 38.5% of the global operating systems by the end of 2012. Developing for iOS would probably be easier considering less hardware variations across devices. On the other hand the Android market is more open, and you can publish your application without google approving it first. App Store from Apple have to approve the application before you can publish it. We choose the Android platform and not iOS, because iOS seems too restricted. In addition, the Android platform is rapidly growing.

Due to the time limit of our project, we are targeting only one platform. Since we have limited time, and since all platforms are new to us, we realised that we would have problems finishing if we were to release the application on more than one platform.

Platform compability

To reach the maximum number of user, we would like the application to be compatible with the most popular and widely used versions of Android. The majority of Android devices do not yet support the latest Android versions. Because of this, we choose to develop for Android 2.1 and above.

Based on the current platform distribution [4] by Android, we came to the conclusion of developing DriSMo with the compatibility of API-level 7. In other words; Android version 2.1 and above, reaching 94.7% of the potential users (based on the current platform distribution seen in Table 1).



Platform	API Level	Distribution
Android 1.5	3	2.3%
Android 1.6	4	3.0%
Android 2.1	7	24.5%
Android 2.2	8	65.9%
Android 2.3	9	1.0%
Android 2.3.3	10	3.0%
Android 3.0	11	0.3%

Data collected during two weeks ending on May 2, 2011

Table 1: Android platform distribution statistics.

General system requirements

The minimum system requirements to run DriSMo includes having an accelerometer (also called G-sensor), with an update frequency of at least 30Hz and minimum version 2.1 of Android. This is absolutely necessary for the application to work. The minimum hardware requirements can be related to the HTC Wildfire [5] (∼500 MHz processing power, ∼400 MB RAM). Internal GPS antenna and 3G connectivity is recommended, but not required.

A recommended system to run the application has:

- Android version 2.2 (or above)
- Accelerometer with 60 Hz or higher update frequency
- 1 GHz processing power
- 500 MB RAM
- 3G Internet connection
- Internal GPS antenna

2.2.2 Performance

Performance is key to creating a high quality application on mobile devices. Due to the limited processing and battery power of a mobile device, unnecessary/excessive work must be eliminated.

Evaluating driving quality must be realtime, and computing this should not take longer than 100 - 250ms. If this interval is any longer, the user will perceive the realtime quality feedback as slow and "laggy".

To preserve battery and computing power, updating a trip file (while monitoring) will take place every 2 seconds. An alternative power-saving option should be provided, updating every 6 seconds. These update intervals will also apply to the rate of GPS update requests, if enabled.

To get the best possible quality assessment, we use the highest possible update frequency of the accelerometer.

2.2.3 Internationalisation

We were never in doubt that the application should be international, and therefore use English as the default language. However, since our vision is to make the project open source, we want to lay grounds for translation to several languages. Step one to do this is to have external language resources, instead of hard coding clear text in our code. To make sure everything worked as intended, we implemented Norwegian language support in our initial release. Since we are based in Norway, some of our local beta testers used this setup.

All apps in my category	
United States	44.3%
Japan	5.5%
United Kingdom	4.8%
France	4.3%
Taiwan	3.6%
Germany	2.7%
India	2.1%
China	2.0%
Russia	1.6%
Spain	1.6%

Table 2: Geograpic distribution of transportation applications from Android Market as of May 13, 2011.

Seeing as we want to reach as many people as possible with our application, there are several things to consider for customization. In addition to language, measurement units also has geographic variations. In America they use miles as a length measurement, while they use kilometers in Europe. According to Android Market, 44.3% of all downloads in the transportation category comes from devices based in America. So, by not providing a choice of measurement unit, we could exclude a lot of potential users. We did not want to do that, therefore we decided to implement this kind of options in the preferences.

2.2.4 Usability

For the application to be successful, it is of utmost importance that it has good usability. If the users have a good experience with our GUI, they are more likely to use the application again. We do of course want our users to drive with DriSMo on a regular basis, and thus have to provide good usability in addition to the underlying features. System response and universal design are important keywords for us in this aspect.

We will be giving the users direct feedback in regards to what is going on behind the scenes, when this is necessary (e.g. when a file has been deleted). When the application is waiting for background tasks to complete, this will be visualised by a loading dialog. That way the user will be aware that some work is being done, and have more patience. The application will also appear to have a better flow.

Another important aspect we have to think about is how cumbersome it is to do certain operations. For instance if the user is located in the main menu, and wants to look at a previous recorded trip; How many interactions will be needed to achieve that goal? Too many interactions for simple tasks might leave the user annoyed. This is something we will have in mind when structuring the DriSMo GUI.

Developing on Android, we have a lot of options in regards to how we facilitate our graphic design. With the possibility to have custom graphics based on language and screen size, we are able to create an interface that looks good on devices in many different sizes and setups. In addition to providing different resolution resources for different devices, we will also use relative layout methods and of course scrollable layouts to make it as universal as possible without compromising the overall design. We will also have Android layout standards in mind, to avoid unnecessary confusions.

When considering usability, we need to think about other things beside the graphic design as well. To increase the accessibility, we are going to include support for navigation with a trackball or directional controller (d-pad). This is important for us, as we have a goal of supporting a wide range of Android devices. It is also advised in Androids developer guidelines [6]. DriSMo should be functional on devices without a touch screen.

2.2.5 Partial releases

First we plan on developing an alpha version of the application. This version will include the main functionalities of DriSMo. These functionalities will be tested internally, and the alpha version will not be released on Android Market[7].

We plan to release our first beta on Android Market, the 12th of April. After this we plan to release patches at the end of each increment, as long as it is necessary. The first beta release should contain most of the functionality, and there should not be too many known bugs. Releasing the beta this early will hopefully help us discover unknown bugs, while we will still have the time to fix them.

A final release is planned on the 24th of May. By this time, we should have most of the functionality fully operative on many different devices.

2.2.6 Licensing

The application is going to be released as open source software. Therefore, we have chosen to release the source code under the GPL version 3 license [8]. This license is compatible with the Apache 2.0 License, used by Android.

The essence of GPL, is that a program distributed in compiled form must also be available to users in the form of source code. All files required to produce the distributed version shall be available for free use.

3 Analysis

The main goal of our application is to give feedback on the driving quality. Before we can say anything about how the quality is, we need to do some research and analyse a lot of driving. By doing this, we get a good understanding of what good and bad driving quality is. Based on this analysis and research we can construct an algorithm to evaluate driving quality.

3.1 Driving quality research

To find a way to evaluate driving quality, we must first determine *what* driving quality is. We have divided the research in two parts; theoretical and practical research. To increase our knowledge of driving quality, we read related articles and subject material. We also did some test driving, to get a better understanding of what driving quality really is.

Based on the testing and research done, we have come to the conclusion that there is three factors defining driving quality. These factors are the driver, the environment and the vehicle, represented in Figure 2. By analysing these three factors, we are able to estimate the driving quality. We define the relationship between good and bad driving quality as the level of comfort experienced in the vehicle. Ergo, good quality represents good comfort, and vice versa.

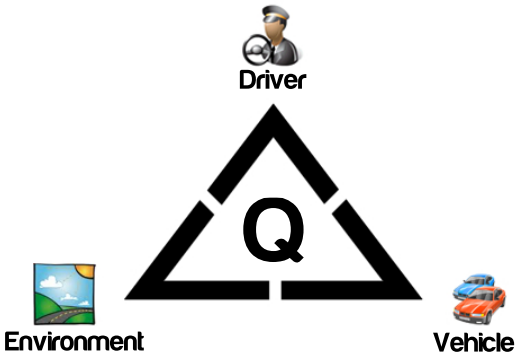


Figure 2: Illustration of the different factors defining driving quality.

It is obvious that the driver is able to affect the driving quality, based on the driving skills of the driver. The better the driver, the better the quality. The website "lets talk driving" [9] defines driving skill in essence, quote:

"It is important to be able to use all of the controls smoothly and efficiently. Don't be happy

> with the fact that you can change gear. This should be done without jerking your passengers about in the car. Likewise when braking, accelerating and cornering you don't want to see them swaying about all over the place. This isn't SKILLFUL driving. This is probably how you will drive when you start learning and it will continue that way for a considerable time; but with practice it should improve."

The environment (bumpy road, highway, etc.) plays a big part when determining the driving quality. Driving on a country road full of bumps is not quite a comfortable experience. The environment as a factor, is also determined by the level of traction. Traction defines the maximum acceleration. E.g. it is not possible to brake as abruptly on an icy surface, as on dry tarmac. Braking abruptly on the icy surface will feel more comfortably, in relation to the dry tarmac. This is because the lack of traction distributes the force over a longer time period.

The vehicle is also a big part of determining the driving quality. Different cars have different suspension with different goals. Some cars got soft suspension and chassis to improve the driving comfort, while others have stiff suspension that is made for maximum traction and feedback to the driver. Driving a sporty car (with stiff suspension) will require more skills from the driver in relation to the driving comfort. On the other hand, driving a car set up to be comfortable requires less skills from the driver. There is also a big difference between different gearboxes. Some cars have fancy automatic gearing, and are able to change gears without the driver or passenger even noticing it. This will result in good driving quality. Older cars with old automatic gearboxes might be more "jerky" than a traditional stick shift and thus it will be negative for the quality. The conclusion is, the vehicle has an impact on the environment, which in turn affects the driving quality.

When we did our research we found out that the things that make up good quality also resembles fuel efficient driving. The key points in fuel efficient driving is smooth driving and planning ahead, so you can keep a good flow. This claim is also stated by different sources [1] [10].

In theory, two drivers can compare their skills by using the same vehicle on the same route. By doing this, the environment and vehicle are removed as determinants of the driving quality (in reference with Figure 2). Ergo, both drivers have equal terms for achieving the best result.

The only difference is now determined by the drivers[1], therefore they can compare the results to see who is the best driver.

3.2 Data collection

To be able to determine if someone drives with a good quality, we will need to find out how the car is acting. When we collect test data, we have to position the phone so it represents the vehicle. The accelerometer gives us updates about the current force the device is subjected to, in SI units (m/s^2). We did not know the accuracy of these sensors, nor the update frequency. After reviewing some minor tests on the HTC Desire and Wildfire, we found out that the frequency was between 30Hz and 80Hz.

When we started this project, we realised that the first thing we had to do was to create a logger. The logger would have to record the accelerometer and GPS data, so we could try to analyse it, to understand what values we get when we drive good or bad.

When collecting accelerometer data, we will mount the phone in relation to the vehicle. This means that values of the Y axis will represent acceleration/braking, values of the X axis will represent turning and values of the Z axis will represent the road surface.

Some test cases were created to get the required data for later analysis:

- Emergency brake
- Soft, comfortable brake.
- Sharp, bad turn.
- Soft, good turn.
- Bad, gear change.
- OK, gear change.

We recorded each of the test cases many times to make sure we got some valid data. The main reason we made these test cases is because it makes it easier to analyse. Since we have data from both bad and good braking, we can later on find out what separates the good from the bad. We used two different phones for collecting data, to see if the data matched. The phones used was a HTC Desire and a HTC Wildfire. The collected data is provided in Appendix G.

[1] Excluding the margins of error.

3.3 Data analysis

The first thing we noticed when we started to look at the collected data, was noise. Test 1 on page 116 (Appendix G) illustrates data from an abrupt brake. There was too much noise in the recorded the data, so something had to be done before we could start our real analysis. The implementation of noise reduction is explained later in Section 5.3. When our noise reduction algorithm was optimized, it was implemented in the application. This way we could analyse noise reduced data on the fly, and give the user accurate realtime feedback.

When we analysed the data, we saw some tendencies to what separated good and bad driving. Both the constant and the changes in force, has an impact on the driving quality. This means that if the constant force in a turn exceeds a certain point, it is concidered bad or ugly driving. This also applies when we initiate the turn. E.g. for initiating a smooth turn the force changed less than $2\mathrm{m/s^2}$ over a time period of 900ms. When we initiated a hard turn we got a change of $4.5\mathrm{m/s^2}$ over a period of 700ms. This was true for both of the phones we used to record data with. We did these kind of analysis on all the test data we had collected. According to the tests we ended up with these thresholds:

Threshold	X	Y
Bad	1.5 $\mathrm{m/s^2}$	2.3 $\mathrm{m/s^2}$
Ugly	4.0 $\mathrm{m/s^2}$	5.0 $\mathrm{m/s^2}$

Table 3: Initial accelerometer quality thresholds.

As illustrated in Table 3, we did not have any threshold values for Z. In the beginning we concentrated the testing on the driving and not the road. Later on we produced Z thresholds with help from realtime monitoring and testing. All of the tests, followed by a short analysis, can be found in Appendix G.

These analysis made the foundation for the first draft of our algorithm. When we had a set of thresholds representing good, bad and ugly driving, we tested it realtime. We did this by driving around with a phone and a laptop, adjusting the thresholds on the fly. This made it possible to observe the changes we made instantly, and confirm if they were right. We repeated the test cases until we thought they were correct, and then we drove on the road testing everything in combination. This gave us the opportunity to validate the thresholds in relation to each other.

We have two possibilities to assess driving quality; by evaluating each axis independently or

by evaluating the total acceleration force. We discussed the matter, and came to the conclusion of evaluating each axis independently (discussed further in Section 7.1.2).

In order to separate each axis, we must calibrate the device to virtually align according the vehicle, as seen in Figure 3. We have to do this, because it would be nearly impossible and very impractical for the users if they had to align the phone in a special position.

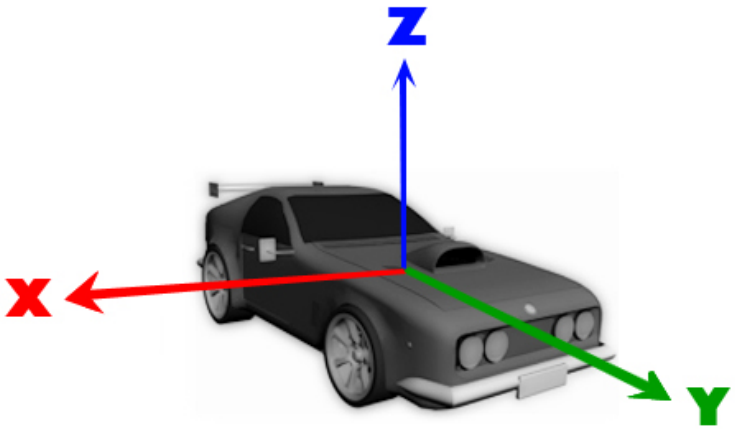


Figure 3: The accelerometer have 3 axes, X,Y and Z. The accelerometer is aligned according to the vehicle, meaning the Y-axis represents the accelerating/braking, the X-axis represents turning and the Z-axis represents the road surface.

Conclusion

After *much* testing and adjusting of the thresholds we ended up with the thresholds shown in Table 4. We think these thresholds are well balanced to each other, and give the correct quality calculations. These thresholds are used for evaluating both the changes in force, and the constant force applied to the vehicle.

Threshold	X	Y	Z
Bad	2.0 m/s^2	1.0 m/s^2	0.8 m/s^2
Ugly	3.0 m/s^2	2.2 m/s^2	1.6 m/s^2

Table 4: The final Accelerometer quality thresholds.

4 Design

In this chapter we will go through the basic architecture of DriSMo and dig deeper into the architecture, to explain some of the key designs used to make the application as structured as possible. Several design patterns have been used or modified to achieve a good structure, and we assume that the reader is familiar with these concepts.

4.1 Best practices

We have been using several of Android's best practices when designing DriSMo. The most weighted practice is to design the application for performance [11]. In our case it was extremely important that the application is as optimized as possible, in terms of performance and efficiency.

Designing for responsiveness was also important, preventing the user from ever seeing the infamous ANR[1] message [12]. We do not want the user to think of DriSMo as a slow and "laggy" application.

We also had to bear in mind that the application is going to support multiple devices, therefore we had to ensure that the design of DriSMo is compatible with each of these devices [13].

When many activities are implemented, it is important that these are controlled correctly. We followed the guidelines provided by Android regarding this [14].

4.2 Basic structure

The basic structure of DriSMo is as shown in Figure 4. This is a compressed version of the application structure, but it contains the key classes and dependencies to provide a good overall picture. The complete UML representation of DriSMo can be found in Appendix E.

The application is divided into three packages; `drismo.logic`, `drismo.gui` and `drismo.model`, to separate view, logic and data into separate layers. The purpose of the model layer is to keep data and present it to the above layers, namely the `drismo.logic` and `drismo.gui` packages.

[1] Application Not Responding

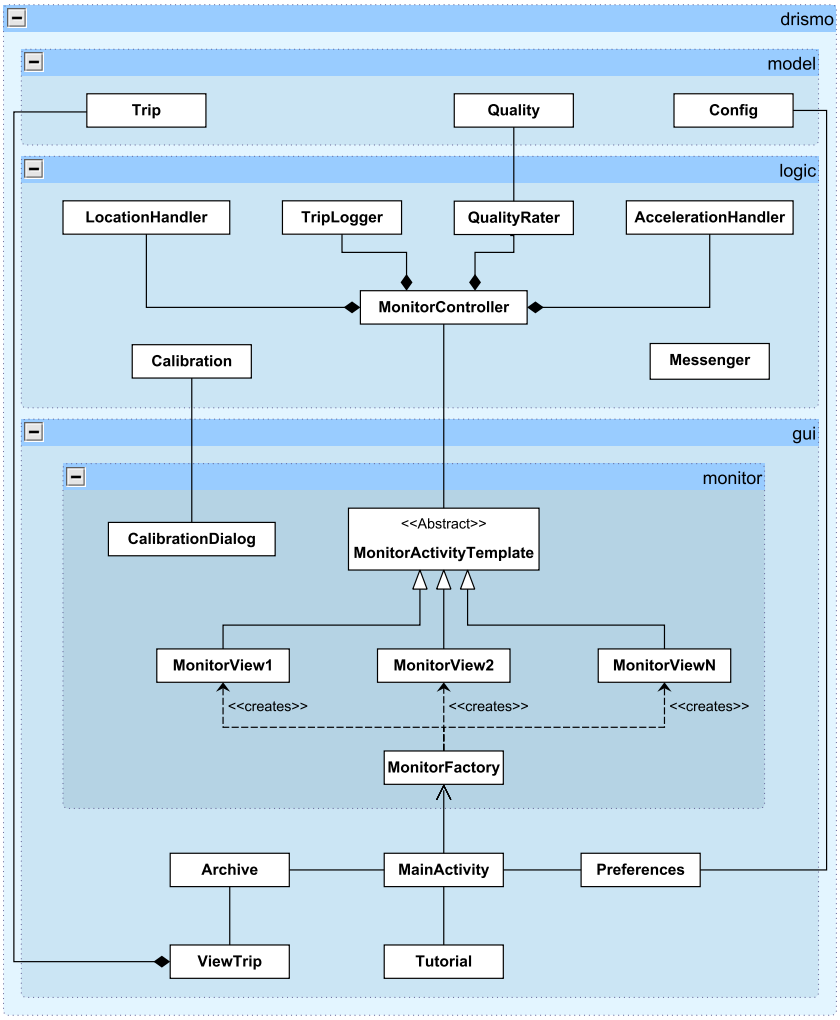


Figure 4: This figure shows the basic structure and dependencies in the application

Classes in the gui package is either derived of `android.app.Acitivity` [15] or `android.view.View` [16]. The `Activity` class takes care of creating a window, where atleast one `View` is added to form the actual GUI.

4.3 Monitor template

Since the application must provide multiple ways of displaying realtime quality changes (as stated in Section 2.1), the design is focused on providing an easy way to write a custom monitor view.

The sub-package `drismo.gui.monitor` is dedicated to classes related to monitoring driving quality. Each monitor (represented in Figure 4 as `MonitorView1`, `MonitorView2` and `MonitorViewN`) is supposed to display the quality in different manners, but the underlying algorithm is the same.

For example, monitor A could change the background color of the application, based on the given quality. Monitor B could draw the quality color as a little square on top of a map view, displaying the past driving route.

Instead of duplicating these monitor classes, we implement the Template Method Pattern [17]. According to the pattern, we implement the abstract class `MonitorActivityTemplate` which is applied as the skeleton for each monitor. By making all the monitor classes derivatives of the `MonitorActivityTemplate`, we avoid having multiple classes with almost identical functionality. The skeleton class will further control the sub-class as needed, and the monitor classes are only required to implement methods specified by the skeleton class. This makes it extremely easy to develop and implement new monitors.

4.4 Listeners and handlers

As specified in the section above, monitors may slightly differ in terms of functionality. As well as receiving quality updates, some monitors might require acceleration updates while others require location updates. Other components than the monitors might also require updates, such as the `TripLogger`. All of these updates are event based, which means they are triggered by some event. To redistribute these updates, we have implemented the Observer Pattern [18]. According to the pattern, we use the classes `AccelerationHandler`, `LocationHandler` and

`QualityRater` as Subjects and the interfaces `QualityListener`, `AccelerationListener` and `LocationListener` as Observers. A class which implements a `Listener` is considered a Concrete Observer.

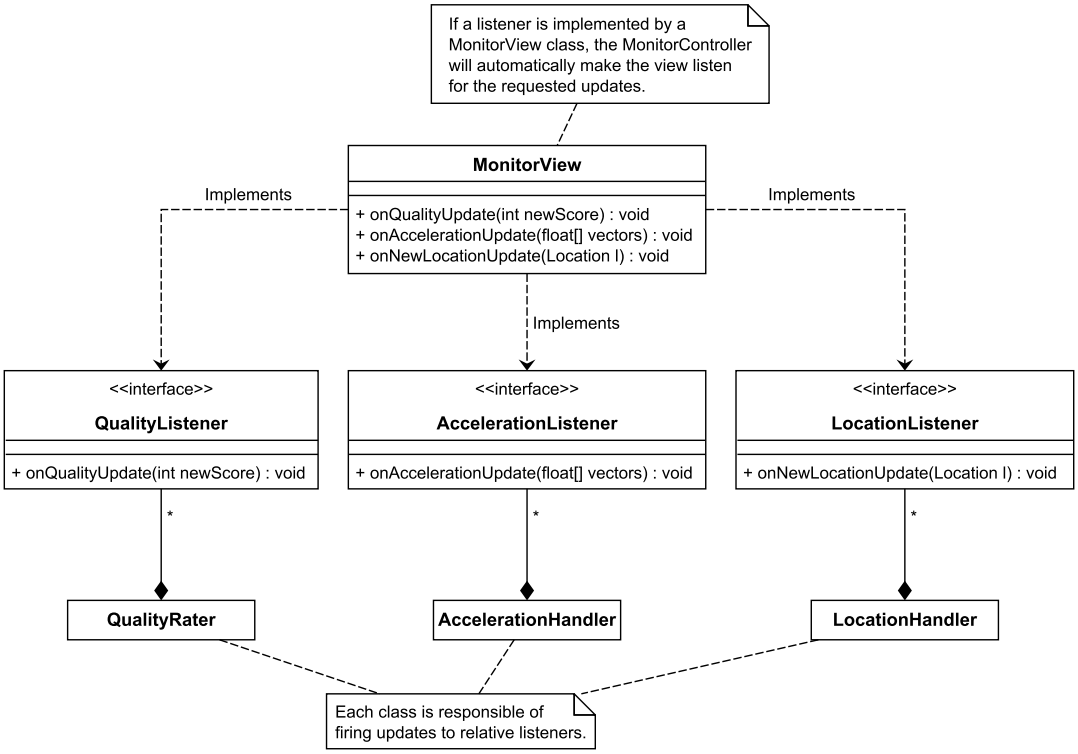


Figure 5: UML representation of a monitor implementing different listeners.

Based on the diagram in Figure 5, we can see how the Subjects and the Concrete Observer are related. The Observer's responsibility is to provide the necessary methods for the Concrete Observer to define, while the subject's responsibility is to update each registered observer when an event occurs.

Figure 5 presents a UML diagram of an example `MonitorView` which requires quality-, location- and acceleration-updates. To make the `MonitorView` able to receive updates, we make the class implement the necessary interfaces. This way the Subjects can call the Observer's related method when an update occurs.

An Observer will receive updates from the Subject, if (and only if) the component is implementing the required listener interface, and it is registered to receive updates from the Subject.

Monitors are automatically registered by the `MonitorController` (see Section 4.5), which means that the `MonitorView` only needs to implement the necessary interface(s) required to receive requested updates.

The Subjects are constantly reciving data from an external source, modifying it and redistributing it. However, to save battery capacity, these classes are constructed in such a way that they will only receive and modify data if one or more listeners are attached. In other words; the Subjects is self-controlling, starting and stopping itself as needed.

4.5 Monitor controller

The `MonitorController` is responsible of maintaining and controlling everything related to a `MonitorView`. It's purpose is to be the link between the actual monitor view and the logic behind the scenes. This controller will only be instantiated once and will be used by the different monitors as they are selected, therefore we have implemented the Singleton pattern [19]. Only one monitor view can be active at a time, therefore switching the controller between different monitor views is practically done as displayed in Figure 6.

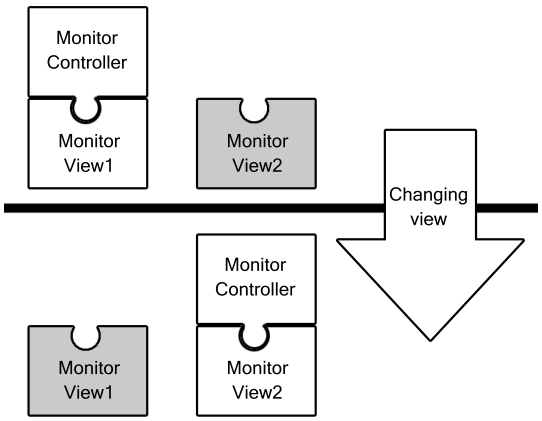


Figure 6: Switching between monitor views.

As mentioned in the previous section, one of the responsibilities of the controller is to register the view as a listener in each handler it is requesting updates from. This means that all the handlers are created and stored in the controller. When switching between monitor views, the controller receives the newly selected monitor view. Before registering the new view (as mentioned in Section 4.4), the controller eliminates unnecessary processing power by purging the outdated monitor view, removing it as a listener since it no longer require updates.

4.6 Monitor factory

To reduce the high number of dependencies between the main activity and the unknown number of monitors, we have implemented the Factory Method pattern [20]. Figure 7 illustrates how this is implemented.

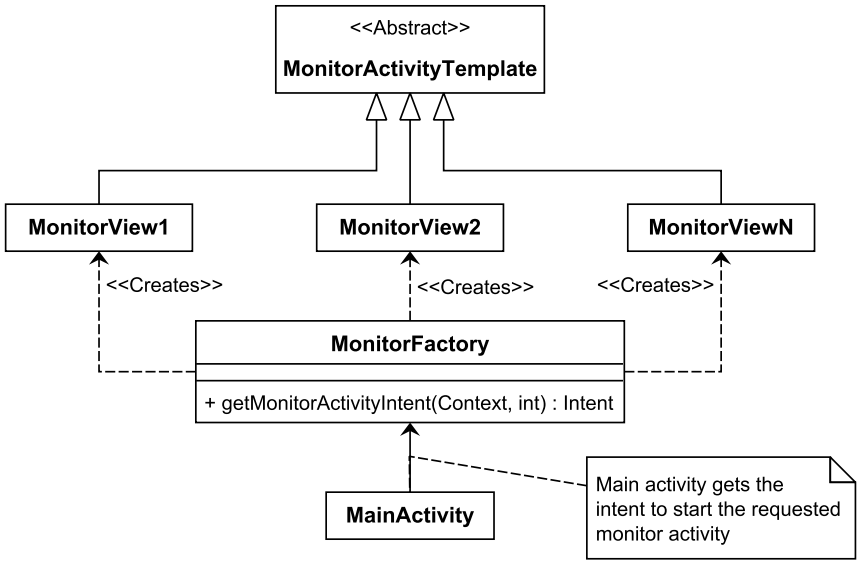


Figure 7: Illustrates the creation of monitor views.

To get an instance of a `MonitorView`, the `MainActivity` must use the static method to get an `Intent` to start the monitor activity. To create that intent, the `MainActivity` must specify the context to relate the intent and which monitor to start, represented as an `int`. When the intent is returned, an activity is started based on the intent.

5 Realisation and implementation

In this chapter we will explain how we realised and implemented the application. We will go into depth explaining the different algorithms, like the noise reduction and the quality rating system. In addition, this chapter will give the reader get a better understanding of the visual design and implementation.

5.1 Development environment

When deciding the program language of choice, we had two choices, either C++ or Java. Developing the application in the C++ programming language using the Native Development Kit (NDK), does according to Android "not result in an automatic performance increase, but always increases application complexity." [21]. Therefore we chose to develop the application in the Java programming language. The Software Development Kit (SDK) provided by Android seems to be better supported than the NDK. We have some experience using Java, so we should quickly adapt to this SDK. Read more about the SDK specification in Section 2.2.1.

As we are developing for Android using Java, there are two major IDE's to consider; Eclipse and IntelliJ IDEA. The trend seems to be using Eclipse in Android development, however all of us have good experience using IntelliJ IDEA. This made the selection easy, and we choose IntelliJ IDEA [22] as our development environment.

To test and debug our application, we used two HTC Desires and one Desire HD. We will also be able to borrow a HTC Wildfire and a Google Nexus S. It may be a bit problematic that we mainly have HTC devices. Even though DriSMo works perfectly on our test devices, it is not guaranteed to work on devices from different manufactors.

5.1.1 Version control

For version control, we were going to use the TortoiseSVN client [23]. We are familiar with this Subversion client, due to previous projects. The Subversion server is hosted by Gjøvik University College. When we created the project in IDEA, we found out that they had integrated Subversion

support. This makes the source control even easier, and we can work with a better flow. The only drawback with the integrated client is the window for merging code. It is painful to use on a small screen, so some of us used TortoiseSVN to resolve conflicts.

5.2 Tools

In addition to IntelliJ IDEA, we have been using some other tools. To structure the report and generate the pdf, we have been using LaTeX with a template from Gjøvik University College.

We have used the Android phone emulator, provided with the Android SDK, to simulate phones with *different* hardware specifications and Android versions.

Photoshop have been used to create the application GUI layout, illustrations, flyers and posters.

We also used Google Docs for collaborative writing, and minor spell checking. Docs is a great tool, since everybody can view and edit the same document in realtime. This is a good way to quickly review each others writing.

We used GNUplot to visualise the collected accelerometer data. This made it easier to analyse collected data. At first we tried using Google Docs to create the graphs, but it was way to slow and could not cope with the large amount of data.

5.3 Noise reduction

It was necessary to implement noise reduction, to get the most optimal quality evaluation (according to the analysis in Chapter 3). To remove excessive noise, we decided to use a moving average [24] algorithm. We had the choice between using a weighted (WMA) or an exponential (EMA) moving average. We tested both WMA and EMA filtering with different number of elements (N), from 9 up to 61. Through testing we found that the ideal N-value is 31. Values less than this were too weak and gave excessive noise, and values above 31 took too long to process (considering the limited processing power of the phone). After analysing both methods in relation to our data, we came to the conclusion of using the EMA. We found the WMA method to be too smoothing, flattening values more than necessary (due to linear weighting). The EMA on the other hand, removed enough noise to make the signal worth processing, while not flatting it out too much.

The mathmatical calculation of both methods (WMA and EMA) are shown in Figure 8, with $N = 5$ elements:

$$\frac{1x_{i-2} + 2x_{i-1} + 3x_i + 2x_{i+1} + 1x_{i+2}}{9}$$

(a) Weighted Moving Average

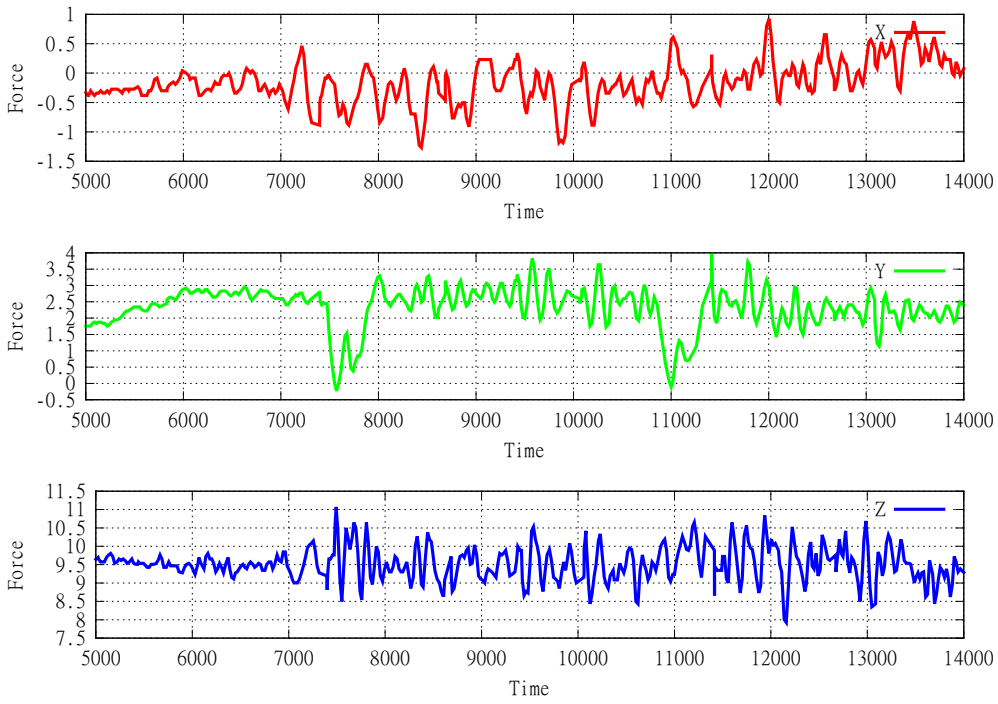
$$\frac{1x_{i-2} + 2x_{i-1} + 5x_i + 2x_{i+1} + 1x_{i+2}}{11}$$

(b) Exponential Moving Average

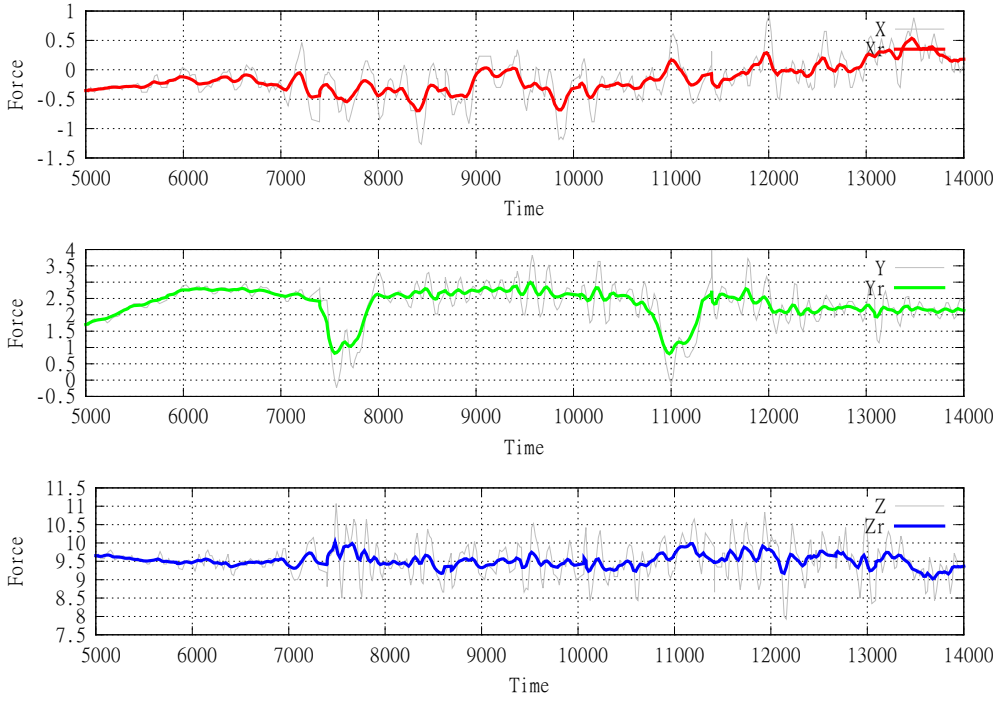
Figure 8: Examples of calculating moving average. WMA uses a linear weight while the EMA uses an exponential weight, but each methods focus their weights on the center element (i).

Figure 9 shows some original logged accelerometer values, and the aftermath of the noise reduction. The Figures 9a and 9b is an excerpt showing a car in motion over a time period of 9000ms, changing gears at 7500ms and 11000ms. Based on these two figures we can see that the noise is significantly reduced, and the critical data is not compromised.

The final code for both WMA (AccelerationHandler.java, line 206) and EMA (AccelerationHandler.java, line 241) is attached in Appendix F on page 109.



(a) Acceleration values before noise reduction



(b) Acceleration values after noise reduction

Figure 9: Original and noise reduced acceleration values.

5.4 Calibration

In order to get accurate measurements to evaluate the driving quality, we came to the conclusion of calibrating the device relative to the vehicle. This means virtually rotating the three-axis accelerometer to represent the vehicles behaviour (see Figure 3 in Chapter 3). To do this we need to know all the angles of rotation, presented in Figure 10.

The calibration is always initiated before monitoring, and is carried out in three steps. The first step of the calibration is to calculate the rotation angles according to the vehicle's level. The second step is to calculate the angle of the driving direction of the vehicle. When these two steps are completed, the final step of the calibration is executed, updating the rotation angles. The detailed calibration flow is illustrated in Figure 11.

To calculate the angles of rotation, we use the `atan2` function [25] to measure the angle between two given coordinates. To calculate the angle between two axes, we see the accelerometer values as polar coordinates provided to the `atan2` function.

The first step, explained above, is done while the vehicle is standing still. In relation to Figure 10, the angles calculated in the first step is both the roll- and pitch angles. The XY magnitude offset is calculated next. This is done by calculating the average magnitude between the X and Y axis. This offset is used to check if the vehicle is in motion. If the offset is varying widely, this means that the vehicle is in motion. If the vehicle is in motion at this stage, the first step is restarted and the offset is reset.

When the XY magnitude offset is calculated, we can proceed to the second step; calculating the yaw angle. First we check if the vehicle is in motion by checking if the magnitude of the XY axis (subtracted by the XY offset) is above a certain value. If this is true, we calculate the yaw angle based on this magnitude, and add the angle in an average buffer. This process is repeated until we have enough angles to calculate the average yaw angle. We use the average yaw angle to be certain that it is accurate and valid. If we just set the first calculated angle as the yaw angle, this has got the potential of being invalid.

When we have all the angles, the rotation angles in the acceleration handler is updated. The acceleration handler can then rotate the given acceleration values. The rotation around the multiple axes is mathematical done as following:

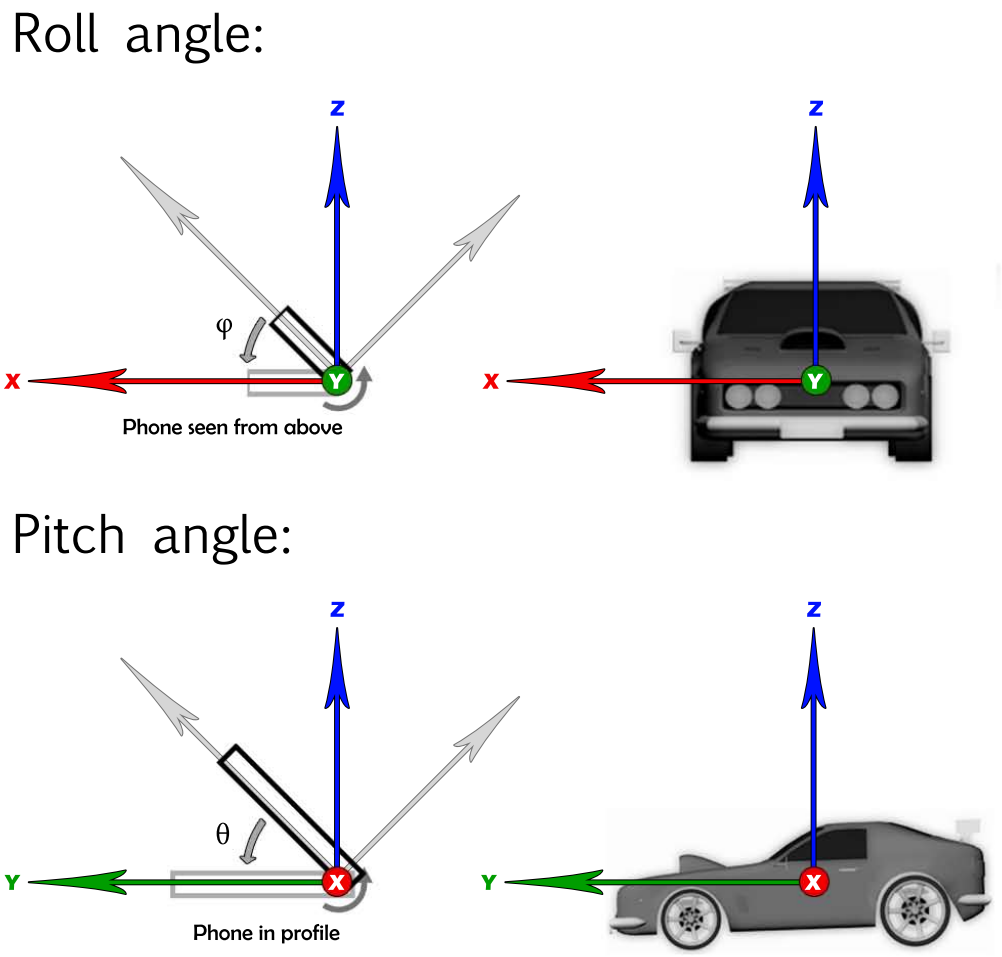


Figure 10: Illustration of the calibration angles.

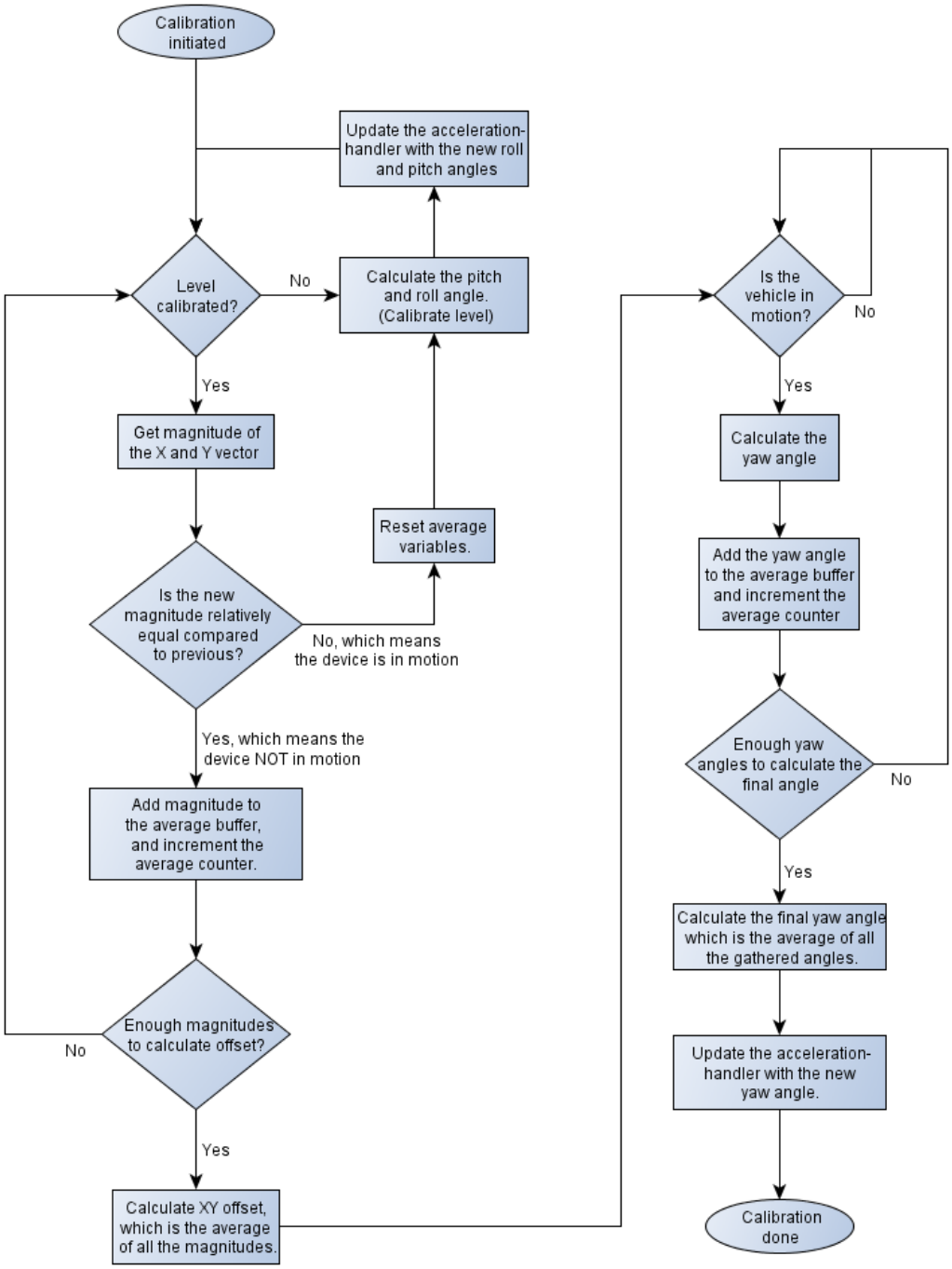


Figure 11: The figure explains the flow of the calibration process.

Rotating around the y axis (roll), using $\angle\varphi$:

$$x = x\cos\varphi + z\sin\varphi$$

$$z = x\cos\varphi - z\sin\varphi$$

Rotating around the x axis (pitch), using $\angle\theta$:

$$y = y\cos\theta + z\sin\theta$$

$$z = y\cos\theta - z\sin\theta$$

Rotating around the z axis (yaw), using $\angle\psi$:

$$x = x\cos\psi + y\sin\psi$$

$$y = x\cos\psi - y\sin\psi$$

Both the calibration code (Calibration.java, page 107) and rotation code (AccelerationHandler.java line 163, page 109) is present in Appendix F.

5.5 Quality Rating

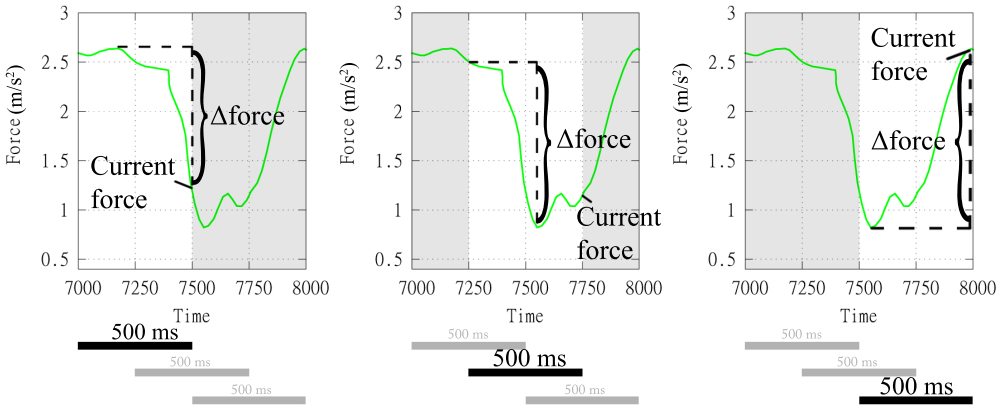
Based on the analysis done in Chapter 3, we need to provide a method for evaluating the driving quality. When discussing how we were going to do this, we thought of giving the user a quality score. This score will decrease if the user drives badly, and increase if the user drives well. One easy way to do this would be to always add the same amount of points, but by doing this the rating would be linear. If we had used the linear rating it would have been hard to give realtime feedback on the driving quality. Since we would like it to be difficult to get an excellent rating but still be able to instantly drop down to bad, we needed to have some sort of logistical rating [26]. We discovered that chess players use a rating system called ELO [27], which we could adapt. The main idea is to give the user a harder punishment if he is driving well and makes a mistake, than if he is already driving badly and makes a mistake.

The new quality rating is based of the change from the current rating:

$$R_{i+1} = R_i + \Delta R$$

The change in rating, defined as ΔR, is either negative or positive. The ΔR is based on 6 different calculations. These calculations are defined as a Δscore. There are 2 Δscore's for each of the 3 acceleration directions:

- One based on the current acceleration. Shown as `Current force` in Figure 12.
- One based on the max difference in acceleration over 500ms. Shown as Δ`force` in Figure 12.



(a) the first sample (b) second sample (c) third sample

Figure 12: Shows how the accelerometer values are sampled

Each of the 6 Δscores is categorized as either good, bad or ugly. If the value of the Δscore is high then the category will be ugly and when it is low, then it will be good. A moderate score will be categorized as bad. The thresholds used for this categorization are dependant on the different directions, but the general categorization is as follows

$$O(a) = \begin{Bmatrix} 1 & \text{if a} < \text{bad} \\ 0.5 & \text{if bad} \leq \text{a} < \text{ugly} \\ 0 & \text{if ugly} \leq \text{a} \end{Bmatrix}$$

Since the O states the quality outcome, we need to compare this to the expected quality. To do this we have to calculate the expected outcome and subtract it from the real outcome. This is done by calculating a score relative to the quality outcome: $f(O) = 1000 \times O + 500$. Then we take this relative score and subtract the current score. This value is put in a logistic function [26] and subtracted from the quality outcome O.

$$O - \frac{1}{1 + 10^{(f(O)-R)/100}}$$

The previous calculation gives us the delta score, however we want this value to differ based on the acceleration type, therefore we use a weight. The weight is scaled based on the acceleration type evaluated. Either 70% if the difference in force is evaluated or 30% if the constant force is evaluated. It is scaled this way, because the variation in force should weigh more than the constant force. The weight is initially 100, but this is divided among the 3 axes of the accelerometer.

$$weight_{delta} = \frac{100}{3} \times 0.7 = 23.3$$

$$weight_{current} = \frac{100}{3} \times 0.3 = 10$$

This means that the Δscore based on delta acceleration must be scaled by 23.3, and the Δscore based on current acceleration by 10. We then use these constants to calculate the Δscores individual weighting based on the same thresholds we used to find O, like this:

$$W = \begin{Bmatrix} weight_x/2 & \text{if a} < \text{bad} \\ weight_x \times 2 & \text{if bad} \leq \text{a} < \text{ugly} \\ weight_x \times 3 & \text{if ugly} \leq \text{a} \end{Bmatrix}$$

By doing this, the weight is scaled based on the quality outcome. This means when driving badly, the rating will rapidly decrease. When driving in an ugly manner, the rating will decrease even more rapidly. By driving well, the rating will incline slowly and steadily. Now we can define the Δscore:

$$\Delta score = W\left(O - \frac{1}{1 + 10^{(f(O)-R)/100}}\right)$$

Now that we have defined the Δscore, we can define the ΔR which is the sum of all the Δscores:

$$\Delta R = \sum_{j=1}^{6} \Delta score_j$$

This makes the complete rating equation:

$$R_{i+1} = R_i + \Delta R = R_i + \sum_{j=1}^{6} \Delta score_j = R_i + \sum_{j=1}^{6} W\left(O_j - \frac{1}{1 + 10^{(f(O_j) - R_i)/100}}\right)$$

In essence we are just checking good, bad and ugly driving, and setting a rating based on this. If the user is driving well, the rating will keep rising and ultimately lead to an excellent rating. Bad driving affects the rating faster than good driving because we want the rating to be realtime. If you brake too hard we want the rating to instantly drop. If you make an even bigger mistake, e.g. braking while cornering hard, the rating will drop even faster. The reason why we choose to weight bad driving more than good is because even one mistake lowers the driving quality.

The properties of this rating algorithm makes the rating peak at 1670. This is the maximum rating and is defined as excellent driving quality. As seen in the graph in Figure 13, the rating flattens out before peaking at 1670, meaning the user will have to drive flawless for some time to reach the excellent rating. The lowest possible rating of the algorithm is 330, but we stop the rating at 1250. By stopping the rating at 1250, the user will get a better rating the instant he starts to drive well. This will lead to the rating being realtime, making the user perceive the feedback as accurate, according to the current conditions.

Figure 13 shows the rating at different stages, affected by different events. Three thresholds are present to interpret the driving quality. The lowest threshold is the BAD-threshold set at 1250, explained above as the cut-off point. This is followed by the OK-threshold at 1400. Ratings below the OK-threshold is considered as bad driving quality. The next quality change is the GOOD-threshold at 1550, so ratings below this is considered as OK driving quality. The highest threshold is the EXCELLENT-threshold at 1650. Ratings below this threshold is considered as good driving quality, and ratings above as excellent driving quality.

It takes approximately 8 seconds of flawless driving to get excellent rating from the point

you got a good rating. To reach the max rating it takes about 13 seconds with excellent driving quality, from the point you got the good rating. Since the rating is logistical it takes 16 seconds from the lowest score (1250) to the highest (1670). This means it only takes 3 seconds longer from the worst possible score to the best, than from a good score (1550) to the best (1670).

Another thing the graph clearly illustrates is how much the bad/ugly values on one, two and all the three axis affects the rating. The number of bad axes determines the steepness of the rating slope.

The complete code for assessing the quality is attached in Appendix F (QualityRater.java, page 103). The `QualityRater` can be classified as a model-based reflex agent [28], in relation to artificial intelligence. It has a model of "how the world works" (and how to assess quality), and it acts on incoming acceleration values. It stores all the incoming acceleration values in a list, spanning over 500ms. These acceleration values are checked each 250ms, which means the quality assessment overlaps (as seen in Figure 12).

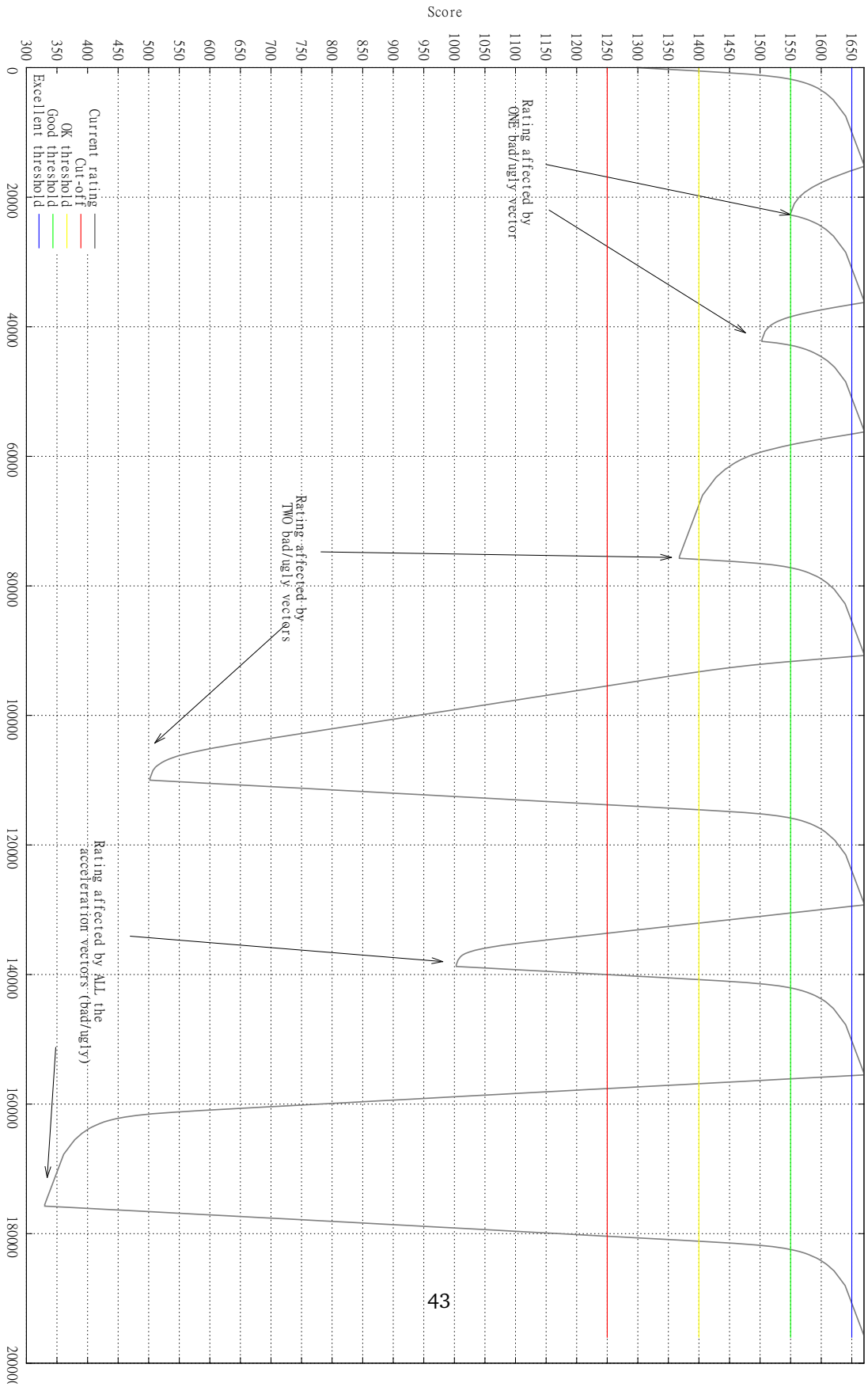


Figure 13: The figure visualise the rating algorithm.

5.6 Quality Monitoring

As specified early in Chapter 2, DriSMo need to give the user realtime feedback based on the current driving quality. We also implemented an archive function that lets the user browse and view previous trips.

We have chosen different colors to represent the driving quality: if the quality is low, the color is red; if the quality is good, it is green; and if the driving is excellent, the color is blue. The colors fade into each other, so between bad and good the color is yellow. The reason for choosing different colors to represent the quality, is because it makes it easier for the driver to see. We think the colors should be self explanatory, but some people might have a problem with understanding that blue is even better than green. However, they should be able to learn it quickly when they start using DriSMo.

Figure 14: Colors relative to driving quality.

5.6.1 Realtime implementation

Primitive monitor

The first monitor screen we would like to introduce is the primitive monitor, as seen in Figure 15. This is the most basic monitor, and it fills the screen with the current quality color. In addition to this, it also displays the quality in text. This monitor is very simple and easy to understand. It is clean and have no distracting elements.

Meter monitor

The meter monitor gives the user realtime feedback in the form of a quality meter, as seen in Figure 16. The reason for this design was to make it look like a speed-o-meter. If the arrow is low on the left side, the current quality is low. As the quality rise, the arrow moves to the right. When the arrow is at 12 o'clock the quality is green and good. If the driving is even better, the arrow continues to the right and the blue color. This means that the driving have been good for a while, and the quality is excellent.

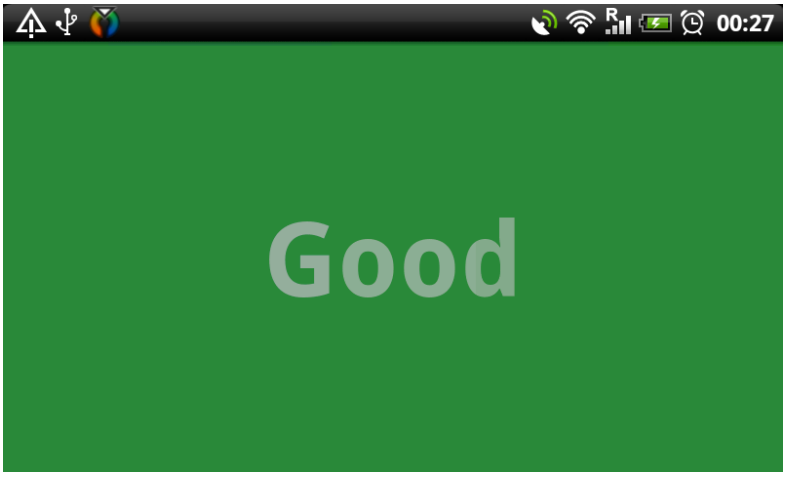


Figure 15: The primitive monitor displaying good quality.

Figure 16: The Quality meter monitor displaying nearly excellent quality.

Map monitor

As illustrated in Figure 17, the map monitor displays the last minute of driving on the map, with colors matching the quality. To display the map, the user have to either have an Internet connection or have the maps cached. Of course the user have to enable the GPS on the phone, and have GPS logging enabled in DriSMo. The map is automatically centered to the latest position, and the user can zoom in or out to get a better view.

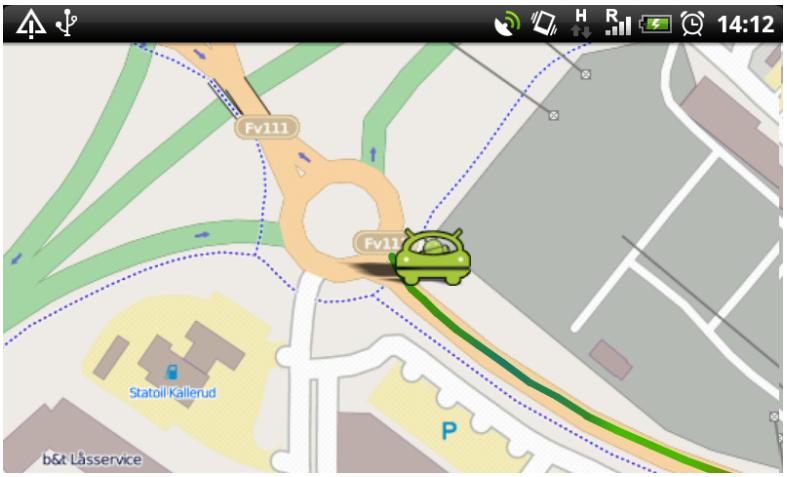


Figure 17: The map monitor displaying the latest quality.

Debug monitor

With help from the debug monitor, the user can make sure the device is calibrated right. The debug monitor draws a realtime graph of the acceleration values. If the vehicle stands still two of the lines should be in the middle of the screen, and one should be near the top. This means that there are no acceleration on the axes, and only the gravity is affecting the Z-axis (blue line). When the vehicle accelerates/decelerates the green line should be affected, and during cornering the red line should change.

5.6.2 Archive implementation

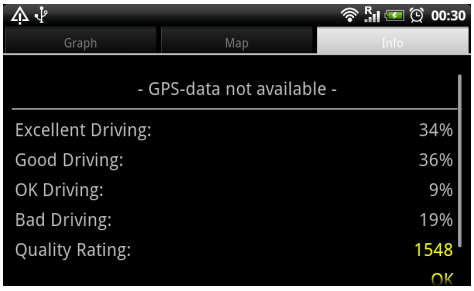
To let the user browse previous trips, we implemented an archive function. When the user opens the archive he is presented with all of the recorded trips, in a scrollable list. Each trip is represented with a name, duration and date of the recording. If the user press and hold a trip-item, a menu is displayed. This menu contains options to view, rename, delete, share or export the trip.

The view trip function is described further in Section 5.6.3. The share to Facebook option allows the user to share a text summary of the trip. The exporting option lets the user export a CSV-file[1] of the trip, so it can be uploaded by other applications, e.g. dropbox, bluetooth or e-mail.

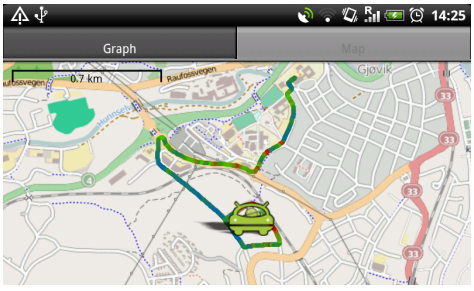
5.6.3 View trip

The archive function lets the user browse existing trips. When the user touches on a trip, the trip opens and displays a graph generated by that trip. The graph is only one in a total of three different tabs; graph, map and info. If the user have logged the GPS, he can select the map tab to get the trip drawn on a map, with colors matching the quality. When browsing the map, the user can zoom in/out and move around the map. The zoom-function is supported by multi-touch, as well as double tap and a set of buttons that are displayed if the map is touched. Both the graph and map view have a share to Facebook option. If the user presses the menu button and selects share, the current view is uploaded as a picture. The picture is generated based on the "what you see is what you get" principle, so the users can zoom in/out and move the map around to show specific details.

(a) Graph view



(b) Info view



(c) Map view

Figure 18: Shows different ways to view a trip.

[1] Comma separated values

The last way of displaying the trip is the info view. This view is just a text presentation of the trip, showing the percentage of excellent, good, ok and bad driving. It also shows the average quality rating of the trip, and the duration. If GPS data is stored with the trip, it also shows the max and average speed, trip distance and start/end location.

5.7 Optimization

Since the application is developed to run on smartphones, there are some things we have to take into consideration, as mentioned in Chapter 4 under best practices. The main challenge with smart phones is the different hardware, and the limited processing power.

When we first made the code for viewing a trip on the map, we did everything wrong. We had a recorded bus trip with over 4000 entries, and when we tried to view it everything was way too unresponsive. We did some testing, and found out that the phone used between 900 - 1700ms to draw the trip. We realised that some of the problem was that even the points outside the screen was drawn. So when we only drew the points on the screen, the new time was 500 - 1000ms. This was still way to slow and unresponsive.

The next thing to do was to rewrite the structure of a trip, because everything was currently stored in a linked list. When we changed from linked list to arrays there was a big improvement. The new time to draw the trip were 150 - 350ms. Finally, browsing a trip was possible, and not just irritating. Since we already had made such good improvements, there was no point in stopping. We then refactored the for loops, by getting the array length before the loop (i.e: `i < size` not `i < array.length`). The new results were between 63 - 250ms. With these improvements we were satisfied and the map was responsive even on a trip with 4000 GPS locations. This means we decreased the drawing time by 1450ms. That is between 7 and 14 times faster than originally.

The next thing to get optimised was the reading from file. We used the same bus trip to measure the reading time. When we timed the reading we found out that it took 1800ms, when using the methods `String.split` and `Float.parseFloat`. We changed the `split` method to a `StringTokenizer`, and the average reading time decreased to 850ms. This is still a long time to wait, so we changed the `ParseFloat` method to a custom parsing method and the result ended in 537ms on average. To try to make it even faster we created a primitive method

to read and split. The average was now 508ms. Although the new method was a little bit faster it would be much more complicated to adapt it later, so we went back to the `StringTokenizer` method. When we sum up all the changes we see that we optimised the reading from file with nearly 1300ms, in other words over three times faster.

As a result of these improvements DriSMo runs efficiently on most devices, even phones with low hardware specifications, such as the HTC Wildfire [5].

Battery efficiency

When we tested the nearly finished application, we found that it reduces the battery by 20% over 30 minutes (HTC Desire [29]). This means that DriSMo will drain the phone completely in 2.5 hours. Some users might percive this battery draining as too fast, so we decided that we needed to offer some kind of power saving options. As a result, it is now possible to dim the screen and decrease the GPS refresh rate. This lowered the battery usage to 15%. To further extend the battery time, the user can disable the GPS and then it only uses 10%. The complete tests can be viewed in Table 5.

Regular w/GPS	20%
Regular w/o GPS	18%
PowerSave mode w/GPS	15%
PowerSave mode w/o GPS	10%
Idle awake (not running DriSMo and not dimmed)	13%

Table 5: Battery consumption over a period of 30 minutes on HTC Desire [29].

5.8 Visual design and customization

In this section we will go through some key elements of the visual application design. DriSMo consists of four main sections, that have direct access from the main menu. The main menu layout is dependant on the orientation of the device, to avoid having a scrollable navigation, even on low resolution devices. To make it as easy as possible, there are no option- or context menus on the main screen.

Color theme

From the project description, we knew that at least some feedback from the application should be colored. The typical colors for feedback would in our case be red, yellow and green. We

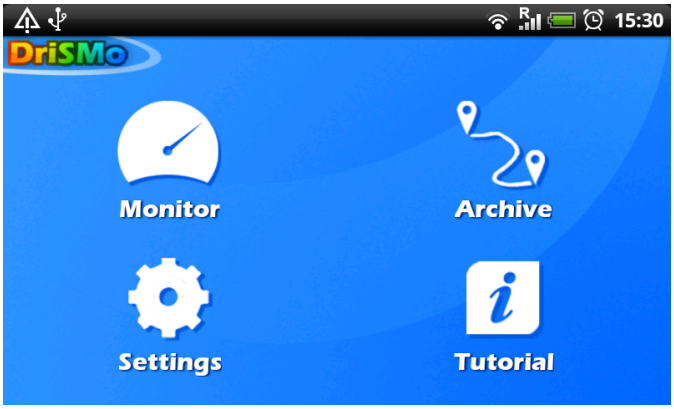


Figure 19: The main menu, with landscape orientation.

decided to use these colors in varying degree throughout the application. However, we were not quite pleased with letting these colors dominate our application, nor did we want to just go with black and white. After some discussions and sampling, we decided that the DriSMo color would be blue. This matched well with the other colors we already had to deal with, so we actually integrated it into our feedback scale, as seen in Section 5.6. If your feedback is in the "DriSMo color", you know that it is even better than good.

Apart from the blue theme, we also keep some sections of our GUI in black. This has been done to differentiate certain sections from the others. The black theme comes into play when a trip is monitored, or a trip recording is viewed. To avoid breaking with the Android standards, we also deviated from adding our custom style to the preferences, so these remain with a black background.

Guidance

It is well known that a lot of users do not read tutorials before they start to use something new. For our application to work as intended, it is vital that the user knows about the calibration process. We therefore decided to add an extra screen (see Figure 20) in the monitor activity, to remind users about the calibration. This screen will *always* show up before the calibration is initiated. We have also had experienced users in mind when implementing the pre-calibration screen. It is nice to just have a break at that screen, while doing all necessary measures to start the trip. When the user is all set, or just want to get on with it anyway, he/she can just tap the screen to initiate the calibration in no-time. In this respect, that extra step actually serves a

dual purpose. When the calibration is done, the standard monitor is initiated. You can read more about monitoring in Section 5.6.1.

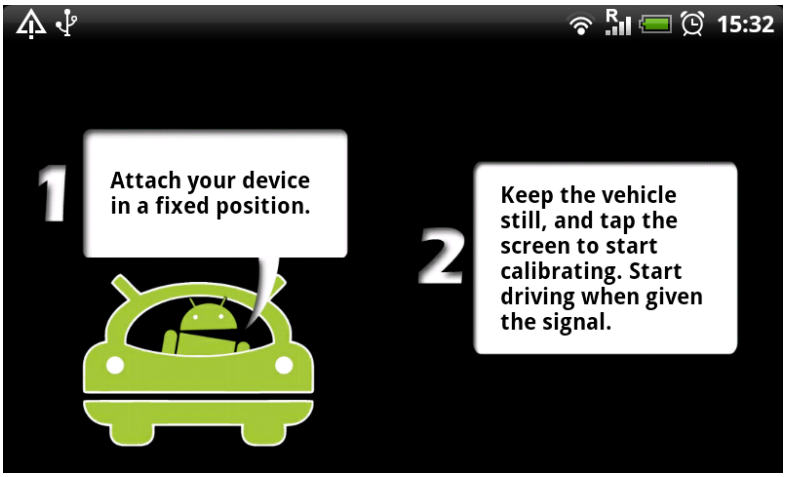
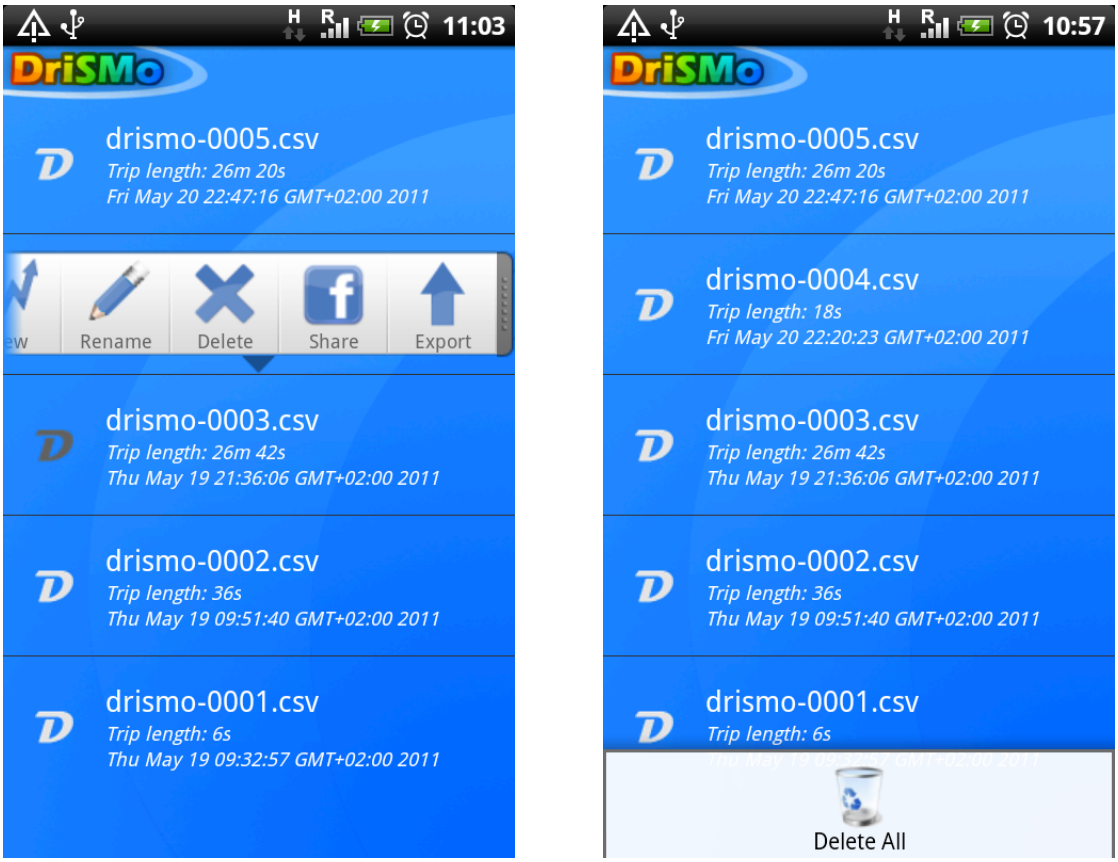


Figure 20: The pre-calibration screen.

Menus and lists

When entering the archive, all stored trips are presented in a list. This sub-section of the app. is the most complex, in terms of navigation and menus. We have taken an approach where the most common operations are easily accessed, while the ones used more rarely require an extra step or two. We have implemented our menus with the Android guidelines [30] in mind, to make sure we design it in a way our users are familiar with.

Normally, the user do not need to do operations directly with the trip files (i.e. export, delete etc.). We therefore have designed the archive with one-tap access to each trip's sub-section, to make a good flow in the application. If the user wants to do additional operations, the quick action context menu is invoked by pressing and holding on a trip item in the archive list. If the user wants to do an operation for all trips, these will be found in the options menu by clicking on the device menu button. We currently only offer the option to delete all files. See Figure 21 for illustrations of these menus.



(a) The quick action bar.

(b) The options menu (delete all files).

Figure 21: The above examples illustrates the different menus in the trip archive.

Different views

DriSMo is designed in a way that allows the user to chose between different ways of viewing the collected data. When monitoring, we have implemented three different ways of following the trip, based on the same data collection. The user can set a preferred monitor in the application preferences, but is still able to switch between views via the options menu while monitoring.

The idea of viewing the same data in different ways is also implemented in the sub-section for each trip. We have created a `android.app.TabActivity`, to let the user switch between tabs and get the trip presented as a graph, map or just a written summary. An example of this can be seen in Figure 18.

Settings

As previously stated, we want to let the users customize DriSMo. To achieve this, a set of preferences is needed. We have used standard `android.content.SharedPreferences` for this purpose. As a result, our preference screen is fully in line with Androids standards, and in a well known format for our users. Both in terms of layout, and functionality.

Special layout arrangements

To achieve our goals in regards to compatibility on many different devices and setups, it was necessary to take some extra measures in special cases. In Android development, we are able to make custom resources based on the device setup (e.g. screen size or language). We have among other specified a different font size for the Norwegian language, in our main menu. This had to be done because the Norwegian equivalents contained more letters. We also used the multiple resource option to supply different graphics for different screen resolution and sizes. Doing this, we in some cases provide higher resolution on graphics for high resolution screens, but also scale some graphics differently for screens with aspect ratios outside the norm.

5.9 Reused Code

For some of our features, we used existing solutions instead of "reinventing the wheel". Some of the code needed modifications to work the way we wanted it to. By reusing the code we found, we saved some time and was able to implement even more features to DriSMo. All of the external code we have implemented in DriSMo is compatible with the GNU GPLv3 [8] licence, and all of the sources has been rightfully credited for their work.

Graph view

A good example of building on an existing solution is the "graph view". In the graph view (Section 5.6.3) we used some code written by Arno Den Hond [31] as a base, and modified the parts we could use. This was faster than writing it all from scratch, since his code was already tested, and we knew it would work.

Osmdroid

To generate and draw the map, we chose to use Osmdroid [32]. Osmdroid is a open source library to replace the original `com.google.android.maps.MapView` class. This library makes it possible to use OpenStreetMap, as one would use the regular Google Maps API. We planned to use OSM from the start, so we needed a way to implement it, and Osmdroid was the best solution we could find.

Facebook SDK

During our beta period, we decided we wanted to make our application social, with hopes of reaching more potential users. We looked for a way to integrate our application with Facebook, and quickly found the Facebook SDK for Android [33]. This enables us to interact with the Facebook developer API. We are now able to log in users, make requests using the REST and Graph APIs, and start user interface interactions with the API (such as pop-ups promoting for credentials, permissions, stream posts, etc.). DriSMo is currently using this SDK to share stream posts with links and images, to the account which is logged on to Facebook via DriSMo.

Quick action

To provide additional options for each of the recorded trips, we decided to implement a context menu for the archive list items. Instead of going with the basic `android.view.ContextMenu`, we wanted to use a quick action bar, as described in Android UI Design Patterns [34]. This was more fitting for the direction we wanted to take with our GUI design. Since we already had seen such menu used in other applications, we went searching for code we might be able to reuse. We soon found a basic quick action menu, released open source by a programmer known as Qberticus. After a bit more searching, we found an extended version of this, created by Lorensius W.L.T. He had release the code under the new BSD licence, which means we were able to tweak and reuse it in DriSMo.

AppiRater

To get more user feedback, we integrated an application rating dialog. This is a popup dialog which is displayed after the user have used DriSMo for a set number of times. We used the AppiRater [35] code to generate the popup, but had to modify it to work on devices without

Android Market installed.

6 Testing and quality assurance

6.1 Whitebox testing

Whitebox testing is defined as a method of testing the applications internal structure, with the knowledge of the application's code and internal structure[36]. Since we are the ones with the knowledge of DriSMo's code and internal structure, it is obvious that we have been doing this kind of testing ourselves.

Due to the lack of knowledge regarding unit-testing on the Android platform, this kind of testing has been rather primitive. By using system prints to screen, we have been able to follow a desired test path, followed by determining if it was working based on the output. This has been sufficient, according to our needs.

6.2 Blackbox testing

Blackbox testing is defined as a method of testing functionality, without the knowledge of the application's internal structure. Programming knowledge in general is not required[37].

Since we are supporting multiple devices, it is important that the application is tested properly on different devices. To make this possible, we need a lot of testers with different Android devices, versions and settings. We decided to release a beta version of the application on the Android Market[7]. Android Market gives us statistics of which devices (including the Android version) that are using DriSMo. Crash reports are also handled by Android market, and can be seen in the back-end developer console (illustrated in Figure 22). This feature gives detailed stack-traces of the errors produced, which comes in handy when debugging the application.

Figure 22: Print screen of the Android market developer console.

User feedback is crucial, therefore we encourage users of DriSMo to help us improve the

application by giving us feedback. Because of this, we have managed to fix a number of bugs related to other devices than the ones *we* have tested the application on.

6.3 Beta test results

Other than finding crucial bugs, we wanted to find out if drivers agreed with the provided real-time quality feedback. Based on the feedback we have received, most of the users have been in agreement with the quality feedback. We have not received any feedback on the quality evaluation being too kind. On the other hand, we have received feedback that the evaluation is too strict regarding driving on a bad road. We chose not to change the quality evaluation accordingly because of the low number of entries with this feedback. In addition, we feel that the evaluation is well balanced with regards to driving quality.

We also got some response regarding faulty quality feedback. This was mainly when the driver drove over a speed bump. When driving normally over a speed bump, DriSMo will mark the current quality as bad. This is due to the fact that the accelerometer is slightly tilted back and forth, spreading the force on multiple axes. We calculated the maximum tilt angle to be $9.18^\circ \times \mathrm{threshold}_{\mathrm{bad}} = 9.18^\circ \times 1.0 = 9.18^\circ$ because of the angle distribution $\frac{90^\circ}{9.81} = 9.18^\circ$ per $\mathrm{m/s^2}$. This means that if the device/vehicle is tilted 9.18°, the quality will be labeled as bad. This principle is also true if the vehicle is climbing a slope of 9.18°.

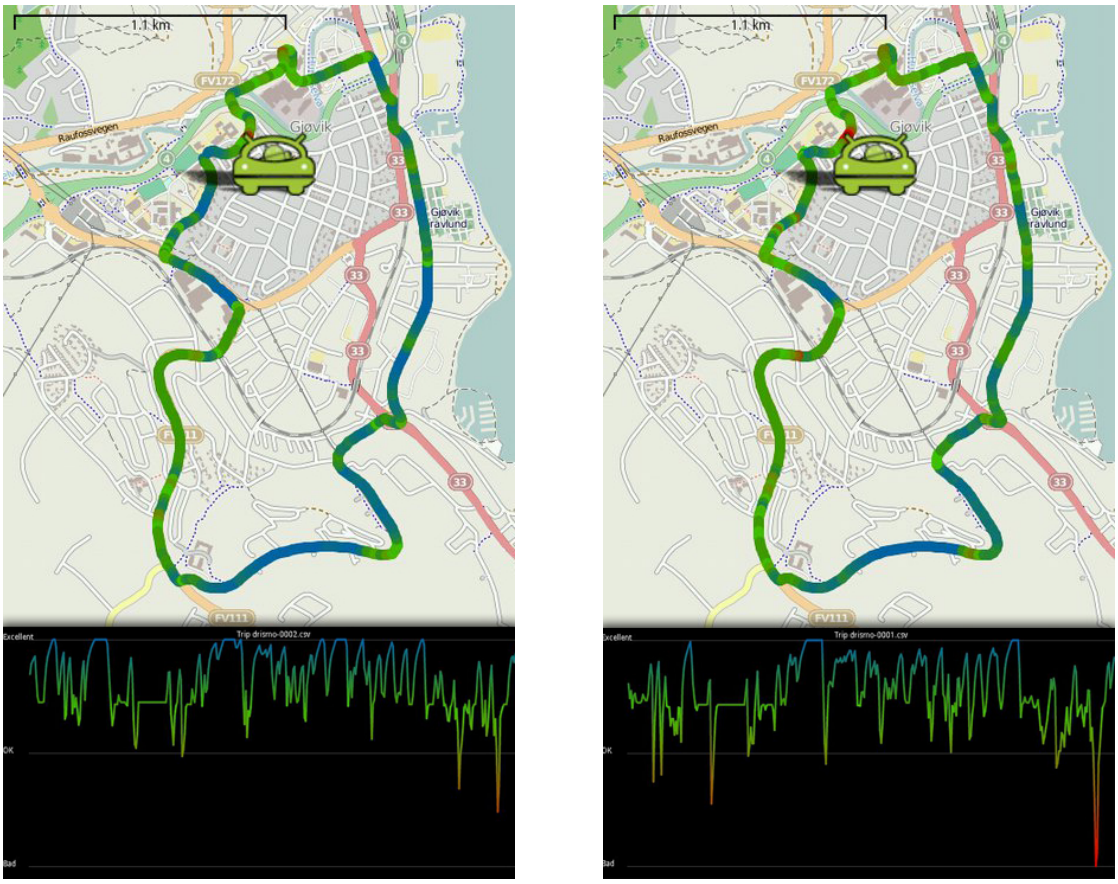
Unfortunately, this is something we can not fix, because we lack the ability to keep DriSMo leveled according to the road at all times. Most of the time, the vehicle and device is aligned to the road, but a speed bump distorts this alignment. However, it can be debated if driving over a speed bump is regarded as good comfort.

6.4 Closed and supervised testing

To test DriSMo, we decided to compare two drivers against each other. We told the drivers to drive normal[1] along a given route, using the same vehicle. The device used as a host for DriSMo, was a HTC Desire. One of the group members were monitoring the test from the passenger seat of the car. The test results is illustrated in Figure 23.

After the test, we asked both drivers if they agreed with the driving quality feedback pro-

[1] No exaggerated acceleration/deceleration or exaggerated turns.



(a) Driver 1 (Avg. Rating: 1593)

(b) Driver 2 (Avg. Rating: 1572)

Figure 23: Test results of the supervised testing, showing the quality map and graph for both drivers.

duced by DriSMo. Both drivers agreed that the feedback was valid. We also asked if they tried to adapt their driving style to get the best result. The driver with the highest average quality rating answered yes, while the other driver said that he partially did. This suggests that the application has an impact on the user's driving.

Based on the result of the test, the difference between the two ratings was exactly 21 points. As seen in Figure 23, the graph displaying the quality rating is relatively even between the two drivers. This was expected, because the conditions (same vehicle, environment, phone) was equal for both of the drivers. The results are slightly different because of the drivers ability to affect the driving quality.

This test proves the theory of using DriSMo to compare driving skills between two drivers. The test suggests that it is a valid way to measure driving skill. The test also suggest that measuring road quality is a possibility. The accuracy of this is difficult to conclude, due to the limited scope of the test and considering the margin of error.

7 Conclusion

7.1 Discussion

7.1.1 Results

Initially, our project consisted of creating a mobile application which could determine realtime driving quality. At the end of the project, we are left with a result which exceeds the initial project description by a long shot. We expanded the scope of the project by adding multiple functional requirements. All of these requirements (specified in Chapter 2) are met in today's version of DriSMo. By the end of the project, we also managed to implement extra features, such as social sharing and text to speech.

As of today, we have released a stable version of DriSMo. During the beta phase, the application grew to over 100 active users, and exceeded 250 downloads globally. We are quite pleased with the final result of DriSMo, and plan to keep on the development through releasing the application as open source.

The application might not seem as much work based on the functionality it presents, but the underlying work is quite extensive. All the underlying functionality that you can not see, is well planned and implemented, such as the calibration and quality rating system.

The plans and milestones set early in the project phase has been kept, and we are generally happy with the outcome of this project. The beta development phase made us get in contact with several users, all across the world. They contributed with some constructive feedback regarding application improvements.

As mentioned in Section 6.3, the majority of the beta testers agree with the quality assessment done by DriSMo. However, we did not consider this as proof enough. Therefore we planned to get DriSMo validated by a driving instructor. This turned out to be difficult, since the ones we got hold of was too busy to test it.

7.1.2 Alternatives

We initially thought of using the camera to take pictures along the route, when monitoring the driving quality. This way we could tag the picture with the current quality. This proved to be a challenge, due to the camera implementation in API level 7 of Android. We could not use the camera to take pictures, without presenting a live view on the screen of the phone. Because of the limited space on the phone screen, we could not sacrifice space for a camera-view, therefore we chose to abandon this feature.

We had to choose between two methods to assess driving quality; to evaluate each axis independently or evaluate the magnitude of all the axes (the total G-force). We quickly concluded that we should evaluate each axis independently, because evaluation of the acceleration magnitude would be too inaccurate. For this to be practical, we must rotate the accelerometer's axes according to the vehicle's axes, as illustrated in Figure 3 , Chapter 3.

7.2 Future development

When developing an Android application, there is always new features to add. In this section we describe some features we could have implemented, if we had more time.

Ask the user if he/she is ok with us uploading the driving statistics anonymous to a database, and generate statistics from this. This could be interesting for NAF [1], NPRA [2] and perhaps insurance companies. With the collected data, we could generate a map and see if there are some areas where most drivers report bad quality ratings. For this to work as intended, we would need many people to use DriSMo.

Another thing we could do with a database is for the users to upload their score, and generate a scoreboard. It could be a scoreboard for specific routes or areas. We could also make a user ranking system, to display each user's overall ranking and make it easier for friends to compare each other.

We though about implementing a realtime multiplayer mode, to make DriSMo a bit more fun. In this mode all of the user's friends could be drawn on the map, and the user could invite other users. If the user's friend accepted the invitation, they could get realtime comparison. This

[1] Norwegian Automobile Federation
[2] Norwegian Public Roads Administration (Statens Vegvesen)

would be nice for people driving together, in different cars. The drivers could then compete about getting the best driving quality. This would be a *good* way to make good driving becoming more fun.

Another nice extra feature would be to make it possible to compare different trips on the phone. E.g. if someone drives the same way to work every day, it might be interesting to compare how they drove on different days.

A camera feature could also be implemented. It could take pictures based on distance or time. When the user views a previous trip, the pictures could be linked to the map. If there where some locations with bad quality, the user could view the associated image to better understand the quality evaluation. There could also be a record video feature that filmed the whole trip. These features would of course require the user to mount the phone in a way that guaranteed the camera free sight.

There could also be a feature to give the user the opportunity to automatically record a trip between a preset start and end location. This could be done by letting the user save a location point from the GPS. If you drive over this point later on, a trip recording is started.

7.3 Group work evaluation

7.3.1 Introduction

The potential problems regarding this project (stated in Section 1.2), did not affect our work at all. At the early stage of the project, one of the group members bought a device, therefore we requested to borrow only one device of Gjøvik University College. This way, all group members had a personal phone to test the application on.

We also stated that testing was very important, and that we eventually could not do all the testing ourselves. This worked out as planned, by launching a beta application. This way we got the application tested on several devices by different users, which gave us some feedback to work with.

7.3.2 Organisation

When dividing responsibilities among the group members, we wanted to let everyone have their shot at being the project leader, for a duration of four weeks at a time. As we approach the end of

the project, we see that it did not really matter much who was the leader at any given time. We have managed to make all decisions jointly, and actually forgot who was the leader from time to time. We look at this oversight as a positive, and we always knew that we had a signed document that stated who the leader was at any time. So, if a leader decision was needed, we could always have dug up our contract.

Having a dedicated contact person also worked well, at least for the first part of the project. For some issues, it was just easier for the person it concerned to handle it himself. Also, when we got feedback from beta testers, it was natural that this contact was handled by the developer most related to the issue.

At the beginning of the project, we stated that we would like to be agile in terms of the development, and opened for the possibility to implement new features as they were requested. We are very pleased to say that this worked out very well for us, and did in our opinion lead to our end product being of higher quality than we would have achieved otherwise. Facebook integration is a good example of a feature we added, without it being even mentioned until mid beta.

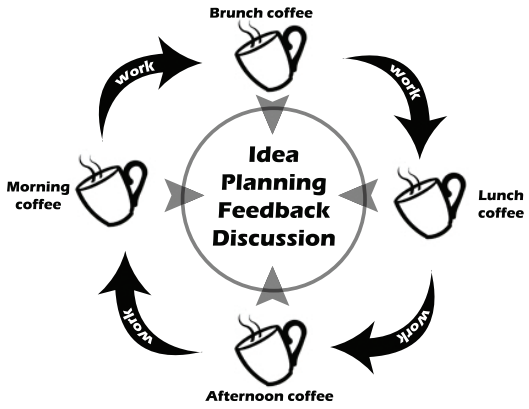


Figure 24: Typical work flow of a day developing DriSMo. The coffee cups resembles daily meetings.

Our coffee meetings was also a successful implementation to our work structure. During these meetings we could discuss current issues, and give each other advice on how to get past it. In some cases we came up with feature request. Some that has been successfully added, while others was discarded after being up for discussion. All features that was requested after we produced

the product backlog, was requested internally or by beta testers. Ergo, our employer probably would have settled for less. As you can see in Figure 24, we had at least four break points during a typical work day. Each of these meetings had a time frame of 5-15 minutes. Of course, we did not give each other "the silent treatment" the rest of the day, but the coffee break meetings was intended as an arena for more structured communication.

We are for the most part pleased with how the development process for this thesis was organised. We have been very agile, and due to some key meetings with our employer, we have always been confident that we would achieve our goals within the scope of the project. We actually were ahead of schedule on our beta milestone, and released it in the Android Market a week early.

7.3.3 Work distribution

The distribution of responsibilities between the group members have been kept as mentioned in Chapter 1. This has provided a relatively even distribution of labor, through the project. In the development phase we divided the workload with focus on the actual monitoring and the quality rating behind the scenes, without sacrificing other important features. This means that Fredrik Hørtvedt and Jørn André Myrland worked on the key features of DriSMo, while Fredrik Kvitvik worked with setting up the GUI and developing peripheral features of DriSMo (such as the messenger). This distribution of labor has proven to be very effective.

7.3.4 Subjective views of the project

Fredrik Kvitvik

I am for the most part very pleased with how the DriSMo project has progressed. Due to working on separate issues than the rest of the team for a while, my knowledge about core functionality could have been better at times. However, this steadily improved, and never became an obstacle for the overall project.

Even though we had to increase our working hours towards the end of the project, we always managed keep a good working environment. I am very pleased with our progression, and think that our end product is of high quality.

Developing an application for Android has been inspiring. At the beginning of the project, my experience with smartphones was non-existent. After borrowing a developer phone from GUC, I

have now realised the huge potential in this technology.

In fact, it has affected my career decisions at the end of my studies, as I will continue working within mobile technology in the near future.

Jørn André Myrland

My personal experience of this project has been very positive. As mentioned, the most difficult part of the project is what you can not see in the application; the mathematics and algorithms. Considering the fact that our degree only have one dedicated math course (and that we had it two and a half years ago), I think the outcome is pretty good. Working with Android and learning this new platform has been very rewarding, not to mention fun. All in all, I am quite happy with the outcome of this project.

Due to a misunderstanding (of dividing the supervisor role between employer and supervisor), our supervisor was only partially dedicated to our project. He helped us a lot in the beginning with some math problems we had, but later on he became more distant. For instance, we used over two weeks to get a hold of test devices from our supervisor, and he never replied on any emails I sent him. Fortunatly, this is the only downside I have to say about this project.

Fredrik Hørtvedt

I have learned a lot from this project. This was the first Android application I have worked on, so it was very interesting and different. The assignment was fun and gave us the opportunity to be creative. This made it easier to be motivated throughout the project. The most difficult part of the project for me, was the mathematical parts. Both trying to figure out how we could rotate the accelerometer values, and how to make the quality rating. Both required large amounts of planning and testing. I also had some minor issues when implementing the map functionality, but with enough googling and testing I figured it out.

One of the best parts with this project was when we released our own application to the whole world, via Android Market. It was nice to see that people all over the world downloaded and tried it. It was also fun trying to optimise the code for the map viewing, because the improvements made a huge difference in terms of usability.

7.4 Conclusion

In the beginning of the project, we were afraid that the scope of the assignment was too small, relative to other bachelor projects. However, this turned out to be in our favour, since we were able to expand the scope in our own direction. By using an agile SDM, we were able to expand the functionality of the application as we progressed through the project.

DriSMo has shown to be a challenging project in terms of design, algorithms and programming. The most difficult part was not to adapt to the Android way of programming an application, but all the mathematical aspects around the project. We used a lot of time and effort to ensure that the mathematics were correct in the calibration and quality rating. The importance of mathematics became obvious when applied to real life problems. This project have illustrated the relationship between mathematics, programming and hardware. We learned this through designing and developing features such as calibrating the accelerometer according to the vehicle, and designing the quality rating algorithm to evaluate driving quality.

We have learned to develop well structured applications on the Android platform, as well as key principles when developing an application on a mobile platform. This project has enhanced our ability to realise and implement a mobile application. We have also learned that there are many constraints related to developing a mobile application, in contrast to developing a program for a computer.

The final conclusion for this project, must be that the time we spent writing/realising the bachelor thesis has been very educational, and not to mention, very amusing. It has been a very good end to our studies at Gjøvik University College.

Bibliography

[1] Naseer, N. 2010. How to derive better gas mileage. http://www.suite101.com/content/how-to-derive-better-gas-mileage-a221446. [Online; accessed 5-May-2011].

[2] The Nielsen Company. 2011. U.s. smartphone market: Who's the most wanted? http://blog.nielsen.com/nielsenwire/?p=27418. [Online; accessed 9-May-2011].

[3] Haselton, T. 2011. Android to grab nearly 50% of global smartphone market by end of 2012. http://www.bgr.com/2011/04/07/android-to-account-for-nearly-50-of-smartphone-market-by-end-of-2012/. [Online; accessed 9-May-2011].

[4] Android. 2011. Platform versions — Android Developer Website. http://developer.android.com/resources/dashboard/platform-versions.html. [Online; accessed 9-May-2011].

[5] HTC. 2011. HTC Wildfire specifications. http://www.htc.com/no/product/wildfire/specification.html. [Online; accessed 10-May-2011].

[6] Android. 2011. Android accessibility practices. http://developer.android.com/guide/practices/design/accessibility.html. [Online; accessed 21-May-2011].

[7] Android. 2011. Android market. https://market.android.com/. [Online; accessed 10-May-2011].

[8] GNU. 2011. Quick guide GPL version 3. http://www.gnu.org/licenses/quick-guide-gplv3.html. [Online; accessed 18-May-2011].

[9] Leaf, B. 1998. What makes a good driver? — letstalkdriving website. http://www.letstalkdriving.co.uk/WHAT%20MAKES%20A%20GOOD%20DRIVER.htm#skil. [Online; accessed 2-May-2011].

[10] Australian Gov. — Department of Sustainability, Environment, Water, Population and Communities. 2010. Driving smoothly leads to fuel efficiency. http://www.environment.gov.au/settlements/transport/fuelguide/tips.html. [Online; accessed 10-May-2011].

[11] Android. 2011. Designing for preformance — Android Developer Website. http://developer.android.com/guide/practices/design/performance.html. [Online; accessed 11-May-2011].

[12] Android. 2011. Designing for responsiveness — Android Developer Website. http://developer.android.com/guide/practices/design/responsiveness.html. [Online; accessed 11-May-2011].

[13] Android. 2011. Compatibility — Android Developer Website. http://developer.android.com/guide/practices/compatibility.html. [Online; accessed 11-May-2011].

[14] Android. 2011. Activity task design — Android Developer Website. http://developer.android.com/guide/practices/ui_guidelines/activity_task_design.html. [Online; accessed 11-May-2011].

[15] Android. 2011. Javadoc: Activity — Android Developer Website. http://developer.android.com/reference/android/app/Activity.html. [Online; accessed 3-May-2011].

[16] Android. 2011. Javadoc: View — Android Developer Website. http://developer.android.com/reference/android/view/View.html. [Online; accessed 3-May-2011].

[17] Wikipedia. 2011. Template method pattern — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Template_method_pattern&oldid=416812236. [Online; accessed 3-May-2011].

[18] Wikipedia. 2011. Observer pattern — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Observer_pattern&oldid=425487525. [Online; accessed 3-May-2011].

[19] Wikipedia. 2011. Singleton pattern — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Singleton_pattern&oldid=425175556. [Online; accessed 3-May-2011].

[20] Wikipedia. 2011. Factory method pattern — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Factory_method_pattern&oldid=427190671. [Online; accessed 3-May-2011].

[21] Android. 2011. Android NDK overview. http://developer.android.com/sdk/ndk/overview.html. [Online; accessed 18-May-2011].

[22] Jetbrains. 2011. IntelliJ IDEA. http://www.jetbrains.com/idea/whatsnew/index.html. [Online; accessed 10-May-2011].

[23] Tigris. 2011. Tortoisesvn. http://tortoisesvn.tigris.org. [Online; accessed 18-May-2011].

[24] Wikipedia. 2011. Moving average — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Moving_average&oldid=426206384. [Online; accessed 3-May-2011].

[25] Wikipedia. 2011. Atan2 — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Atan2&oldid=427293302. [Online; accessed 18-May-2011].

[26] Wikipedia. 2011. Logistic function — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Logistic_function&oldid=427220499. [Online; accessed 4-May-2011].

[27] Wikipedia. 2011. ELO rating theory — wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Elo_rating_system#Theory. [Online; accessed 3-May-2011].

[28] Wikipedia. 2011. Intelligent agents — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Intelligent_agent&oldid=422999101. [Online; accessed 25-May-2011].

[29] HTC. 2011. HTC Desire specifications. http://www.htc.com/no/product/desire/specification.html. [Online; accessed 10-May-2011].

[30] Android. 2011. Menu design guidelines — Android Developer Website. http://developer.android.com/guide/practices/ui_guidelines/menu_design.html. [Online; accessed 21-May-2011].

[31] Hond, A. D. 2011. Graphview. http://android.arnodenhond.com/components/graphview. [Online; accessed 18-May-2011].

[32] OSMdroid. 2011. Osmdroid. http://code.google.com/p/osmdroid/. [Online; accessed 18-May-2011].

[33] Facebook. 2011. Facebook sdk. https://github.com/facebook/facebook-android-sdk/. [Online; accessed 18-May-2011].

[34] Android UI patterns. 2011. Android ui patterns. http://www.androiduipatterns.com/p/android-ui-pattern-collection.html. [Online; accessed 18-May-2011].

[35] Sissi. 2011. Appirater. http://www.androidsnippets.com/prompt-engaged-users-to-rate-your-app-in-the-android-market-appirater. [Online; accessed 18-May-2011].

[36] Wikipedia. 2011. White-box testing — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=White-box_testing&oldid=416219349. [Online; accessed 10-May-2011].

[37] Wikipedia. 2011. Black-box testing — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Black-box_testing&oldid=423671360. [Online; accessed 10-May-2011].

Appendices

A Work log

The project work log for all of us:

10.01 - 23.01: Meetings and initial work with the pre-project report.
Jørn M: 31 hours, Fredrik K: 35 hours, Fredrik H: 35 hours.

24.01 - 06.02: Pre-project plan, setting up the development environment and getting to know android and OSM.
Jørn M: 47 hours, Fredrik K: 50 hours, Fredrik H: 50 hours.

07.02 - 20.02: Writing the code for the logger, collecting test data.
Jørn M: 54 hours, Fredrik K: 47 hours, Fredrik H: 50 hours.

21.02 - 06.03: Analysing test data, meetings with Simon and starting on the rating algorithm.
Jørn M: 45 hours, Fredrik K: 45 hours, Fredrik H: 46 hours.

07.03 - 20.03: Realising the algorithm and implementing it.
Jørn M: 71 hours, Fredrik K: 72 hours, Fredrik H: 71 hours.

21.03 - 03.04: Coding
Jørn M: 65 hours, Fredrik K: 73 hours, Fredrik H: 75 hours.

04.04 - 17.04: Coding, finishing up the first beta release for Market and starting on the report.
Jørn M: 71 hours, Fredrik K: 70 hours, Fredrik H: 71 hours.

25.04 - 08.05: Fixing bugs based on beta feedback, implementing new features, report writing.
Jørn M: 106 hours , Fredrik K: 95 hours, Fredrik H: 106 hours.

09.05 - 22.05: Fixing more bugs and working for the final release. Writing and finishing up the report.
Jørn M: 125 hours, Fredrik K: 125 hours, Fredrik H: 125 hours.

Jørn M. total: 615 hours
Fredrik K. total: 612 hours
Fredrik H. total: 623 hours.

B Meetings

We have included all of the meeting protocols.

Meeting protocol

Meeting #: 1
Subject: Description of the project

Time: 18 January 2011
Place: NISlab, Høgskolen i Gjøvik
Duration: 30 minutes

Participants: Patrick Bours, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

First we need to write a program for the phone, which can log acceleration data. We should also look at the possibilities to store data from the GPS and/or camera. This may lead to capacity problems, so that is something we need to be aware of and look into.

We will also need to create a computer-program to store and synchronize data from all the devices we use to collect data, including multiple smartphones and a camera. To get help collecting data, we could contact a driving school and check the possibility to collect data from their driving lessons. The collected data will then be analyzed, so that we learn what characteristics the data of good, mediocre or bad driving has. To get a proper analysis of the data, we will use the ambulance simulator at the HOS institute. Feeding the simulator with our data, we can let more people experience the ride, and help us determine if the driving was good or bad. We can then use this human input to better our data analysis.

When we have a good understanding of how data for good or bad driving looks like, we can focus on programming the mobile app to present this to the user, both during and after the driving takes place. To give the user a better experience, we can look in to the possibility of using video or images, in addition to maps and other GUI-elements. If we decide to use images from the smartphone camera, we should maybe limit ourselves to 1 picture per second (or something like that) to avoid possible capacity problems.

Meeting protocol

Meeting #: 2
Subject: Clarification of tasks at hand

Time: 19 January 2011
Place: Structure-A, Høgskolen i Gjøvik
Duration: 50 minutes

Participants: Simon McCallum, Trond Stokkeland, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

The purpose of this meeting was to clarify the tasks at hand, and look at the possibilities to collaborate with Trond Stokkeland in some areas, since he is currently working on a master thesis that has some similarities to our project. During the meeting, we agreed that Trond was going to look at driving quality in a more scientific way. He will be working with the simulator, so that is no longer part of our project. We will however keep in touch with Trond, as we could share experiences, and he may be interested in using the test data we collect.

Our project is at the moment limited to creating the single phone app, logging accelerometer data to measure driving quality. We will also capture video, and include GPS-tracking and OpenStreetMap. If we later on decide to expand the application, we're thinking of the possibility of making it into a game. The game would give points to the user according to how well he drives.

We also talked about the development environment we are going to use. We got some input that the android emulator is very slow, so we should keep our phone plugged in when we are testing. Currently we only have one android phone in our project group, but we talked about borrowing another HTC Desire during our work on this project.

Meeting protocol

Meeting #: 3
Subject: Discussing our priorities for the application

Time: 16 February 2011
Place: NISlab, Høgskolen i Gjøvik
Duration: 40 minutes

Partisipants: Patrick Bours, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

In the beginning of the meeting, we looked at some output from our logger demo. While looking at the collected data, we discussed different ideas on how to analyze data to look for good or bad driving quality, and got some ideas on how to reduce noise in the data. We also talked a bit about battery usage, in regards to data storage.

The rest of the meeting we discussed the different ideas/features we had put in our project backlog. In short, we agreed with Patrick for the most part, and got some importance ratings from him to add to our product backlog stories. The feature of having a live map on screen while driving was discussed a bit. Patrick concluded that it would be a good feature to add, as long as it doesn't take too much time. This is something the group has to discuss, and make an estimate before we make the decision to go for it.

In general, we should focus on features that either improves the application purpose or is going to attract users. In example, features like weather data probably don't fit that description.

Meeting protocol

Meeting #: 4
Subject: Demo of our test data

Time: 25 February 2011
Place: K202, Høgskolen i Gjøvik
Duration: 20 minutes

Partisipants: Patrick Bours, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

We met with Patrick to show him our findings from the current test data graphs, and discuss how to do the quality measurements. We also showed him some different samples of how we plan to remove noise from our data, and got his opinion of what was the better suited technique.

To implement all the sensor values into a quality score can be done a number of different ways. An easy way of looking at it would be to just immediately sum up all the axes into one, and give each reading a total sum. With this approach, calibration would not be needed, because it wouldn't matter which of the axis we get the impact from.

Another way of doing it would be to look at pre-defined events from each of the axis, and see them in relation to each other as well. We haven't explored how events on different axis relate to each other enough yet, so this is absolutely something we should look more into. Working with calibrated values will facilitate a better quality score, and it will be easier to further develop as well.

We agreed that the calibration was something we should speak with Simon about, so we arranged a meeting with him as well. A summary of that meeting is available as well.

Meeting protocol

Meeting #: 5
Subject: Tool demos and calibration talk

Time: 25 February 2011
Place: A114, Høgskolen i Gjøvik
Duration: 60 minutes

Partisipants: Simon McCallum, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

Since we had some issues regarding the calibration of the device, we requested this meeting with Simon. We started out with some general talk about how we would like it to be calibrated etc. without going too much into details on the mathematics behind it.

As the meeting progressed, we moved a bit away from the subject, and Simon did some demos for us on LaTex and Gnuplot. Gnuplot has already been very useful for us, as it is much more effective than Google spreadsheets when it comes to drawing graphs, especially when they contain a lot of data. This had been an issue for us, so we are very glad that we now have found a better tool to handle our data. We haven't looked into LaTex enough at our own time yet, but it looked like a good tool for building our project report.

With regards to the intended subject of the meeting, we still didn't quite get the "eureka-moment", because we apparently need some refreshing of our math-skills. This is why we had to schedule another meeting the following week.

Meeting protocol

Meeting #: 6
Subject: Mathematical difficulties

Time: 28 February 2011
Place: A114, Høgskolen i Gjøvik
Duration: 45 minutes

Partisipants: Simon McCallum, Fredrik Kvitvik, Fredrik Hørtvedt and Jørn-André Myrland.

After falling behind schedule, we needed some guidance to get us moving again. We struggled a bit with the calibration, so Simon had to give us a short lecture on trigonometry and vectors. We talked about how to calculate angles and rotate the vectors to do the needed calibration.

In the end, Simon told us to read up on polar coordinates, and go from there.

C Status reports

STATUS REPORT

Week 14 – 2011

Focus and deadlines

According to our planned timeline, the beta release is set to Tuesday 12th of April. As it looks now, we will release according to plan. We are actually closer to the final release, than we initially planned for this beta. This is because we've had full focus on the application development towards this point in time.

As we failed to include time for report writing in our timeline, there has been little or no progress in that area. This is something we've been aware of the whole time, and should not be a concern. However, we will focus mostly on the report from here on in. This means that as far as new features concern, we don't expect many additions from our beta till the final product.

Beta testing

We've not been looking very widely for beta testers to this project. We're rather counting on a core of people that we know will give us good constructive feedback. As part of our testing, we will be following these testers passively the first time they use the application. This way we will be able to see for ourselves if the application is as intuitive as we are hoping. We will be writing a check list for ourselves, so each of us know what to look for when we observe the testers.

After this initial testing, we will let the users (probably more than the above mentioned core) try the application for a while, throughout Easter and beyond. There hopefully won't be any huge issues coming our way, although some smaller bugs are expected.

Problem areas

The application is now doing mostly what we expected to complete during this project. There is however one feature we talked about beforehand, that it looks like we can't implement. We've been trying to implement a feature to take geo-tagged still pictures at a given interval, as we thought this could be interesting for the user when looking at the trip date afterwards. The problem with this is that

we wanted it to be a task running in the background, and thus not take focus away from the other GUI-elements. This has proven to be very difficult to hack, and is probably not permitted in Android development. So, basically it looks as though this feature will not be included. At least there won't be any photo feature in the beta release, that's for certain. We did however manage to finish many other features that were of importance to us, and our employer. A list of application features can be found below. This list will be written in more detail in a beta release note coming next week.

Features

- Advanced quality monitor
    - Multiple monitor views
- Retrospective trip analysis
    - Quality graph
    - Quality rating and general trip information
    - Map view, including quality track
- Map and GPS-tracking
- SMS auto-reply
- Power-saving option
- Export quality data

STATUS REPORT

Week 18 – 2011

Recently added features

We have been at the end of our application development for a while now, but is seems there is always one last patch to apply. Lately we've implemented a Facebook SDK (development kit), and given the user some different ways to tell his/her friends about the result in DriSMo. They can either share only a text summary, or attach a picture of the graph or map to the summary. For the first option, we will attach a link to download the app. in Android Market.

We are very happy to present the Facebook integration to our current users, and hope that Facebook integration will generate some extra downloads as well. DriSMo have always been about bragging to a certain degree. What better way to brag than post the result to the arena where you've gathered all your friends?

Community

During our beta, we've experienced that it can be a bit difficult to get feedback from "random" users. It is therefore important to get a good amount of test users, to increase the chance that at least some of them will give us some feedback. We've been writing about our project in a lot of Android related communities, and hope that this will generate some fuzz and hopefully some proper feedback from experienced users.

We've also decided to make DriSMo an open source project, and release it under the GNU GPLv3 license when we graduate. We have already set up our account at SourceForge, and hope for some contributions in the future. We think that the application has a lot of potential, not only in the private market. One example of this would be to collect the trip files (with geo-data) to a server, and use the data to create some kind of heat map for areas where bad driving quality is exposed. This could help public road administration to see where they probably should improve the roads. This is something we believe could be a success, but at this point in time, we can't fit it into our project scope.

Writing the report

We've been making some good progress on our report lately, but there is still a long way to go. We're writing it in LaTeX, using the template supplied by the school. This way we make sure we get a nice and clean layout on our report, in addition to a good tool for adding mathematics and graphs to the document.

We've recently tried to make contact with some driving instructors, to take a test run with our app. and discuss the feedback it is giving. This could and should probably have been done a bit sooner, to allow time for us to improve on potential issues. However, that kind of feedback would still be valuable to us, and a good addition to the report, even if it doesn't turn out to be all positive.

Still some problems

We're not very far from justifying going out of beta, and release a stable version of the application. We do of course aim to do this before the project report deadline. Our biggest problem at the moment is our limited access to test devices. In our group we have two HTC Desires and one HTC Desire HD. These are very similar phones, and we can't really say that we have been doing in-house testing on a wide range of devices.

Our beta testers have been using some other devices as well, but we can't really trust that everything worked as intended, when users don't bother giving feedback when shit hits the fan. Luckily we have been speaking to some testers first hand as well. This actually revealed our biggest problem yet. One we haven't managed to resolve. The calibration in DriSMo has been reported to fail on 4 different Samsung phones. All phones were of the same model, the i9000 Galaxy S, and our app. is practically useless on that model.

We hope to resolve this ASAP, but don't have high hopes since we don't have that model available as a dev. phone.

D Pre-planning

In this appendix you can find the pre-planning report for DriSMo.

This is a pre planning report for the DriSMo bachelor project. In this document we'll among other things discuss the project organization, goals, restrictions, risks and development methodology.

Project Plan

DriSMo

Jørn-André Myrland, Fredrik Kvitvik and Fredrik Hørtvedt

Content

1. GOALS AND RESTRICTIONS

1.1. Background

Today's technology allows car drivers to be guided by a GPS. To take this a step further, we could provide a system to monitor and analyze the driving quality. This will help us determine if the driving is good or bad, and use this information to help the driver improve him-/herself. Smart phones are becoming more and more common, therefore we can utilize this source of technology to take measurements and monitor the quality of driving. Smart phones are packed with useful sensors like the accelerometer, electrical compass and of course the GPS. These sensors makes it possible to measure the force that the car is exposed to. When we analyse this data, we can say something about how smooth and good the driving is.

Patrick Bours (NISlab, HiG) presented us with the idea of making an application for mobile phones, to let people review and/or monitor the quality of driving, with the use of only their smart phone. After meeting with Patrick, we agreed that this application would be our employment for the last bachelor semester.

The project was originally named "Driving Quality App for Mobile Devices", but we soon decided to rename it, as that made it easier for us to create a catchy acronym. After playing a bit with words, we came up with "Driving Skill Monitor" and the short hand name "DriSMo". A possible issue with this name is the use of the word "skill" instead of "quality", as these represent two different things. However, a certain amount of skill is needed to achieve good driving quality, so our name still holds. The issue is that driving skills is much more than what can be measured by our application, so in that way it is a bit misleading. DriSMo will be the project working title, but the application name could still be changed before the release.

1.2. Project goals

DriSMo's goal is to emphasize the attention towards good driving quality and make the drivers aware of their own driving quality. By raising the awareness of driving quality, the drivers can increase the quality of driving. Also the fuel efficiency will be improved, as a result.

A stable version of the application should be finished and released in the middle of May 2011. This application should include the driving skill/quality monitor, GPS tracking, video feed and the ability to record (and play back) this data.

1.3. Platform selection

The assignment does not set any restrictions regarding what platform this application should be developed on. We want to reach out to as many users as possible, but at the same time we have to limit ourselves due to limited time on the project. We have therefore decided to choose only one platform in the scope of the project, but the application has the potential to be supported on several platforms at a later stage.

The idea of prioritizing development on the iPhone platform was soon discarded. Due to requirements of having to develop the application in an Apple environment, and having the iPhone for testing. Since we don't have any Apple computers (or phones), we would depend on having access to multiple computers in the "mac-lab" at all times. This could cripple us, since we should be spending most of our time programming. It would also rule out working from home.

We chose to develop the application on the Android platform. This is because we had all the necessary tools available right from the start.

1.4. Restrictions

- We will be limited to using only the sensors of the phone when assessing the driving quality. An exception to this is the camera. We will not be using captured footage as part of our quality analysis.
- The application will be developed for Android smart phones. Therefore we are restricted to the API given in the Android software development kit (SDK).
- When developing the application we also have to be aware of the power usage and limited processing capacity. We desire an optimal solution which is effective and power efficient.
- The amount of sensors (accelerometer, orientation, GPS, etc.) may differ from phone to phone, therefore we must not depend on every sensor to monitor the driving quality, but a minimum demand for the application to work is that the accelerometer is available.
- Economic boundaries must also be taken in consideration. I.e. some users might not use the GPS as a sensor because of the cost.
- Since the application is focused on helping the driver, it is essential that we don't distract him while he is driving.

2. SCOPE

2.1. Project description/appraisal

The assignment is to make a smart phone application, that can monitor and analyse driving quality. The user will use this application when he/she is driving. The application will gather data from the smart phone running it, and analyse this data to calculate the driving quality. The application will have 2 different modes. One mode monitoring the quality while driving, and the second mode will let the user playback (view a previous trip) where he/she has driven and the quality of driving at different locations. Both modes will use open street map, and color the trip according to driving quality (green, yellow and red). If the user want to record/see the location, they will have to enable the

GPS, so the app can log the trip. The amount of trips which can be recorded is only limited by the free space available on the phone's memory. To playback a trip, the user selects the trip of choice from a list of recorded trips. He/she can of course also delete selected trips, and choose if he/she will have new trips recorded.

To be able to determinate what good driving quality is, we first need to make a logger. Then we can collect data from what we think is good and bad driving, and analyze it to spot differences. To validate our analysis we can try to get a driving school to record some data for us, to see if this matches our view. We may also be able to validate the application in the ambulance simulator (at the HOS institute).

3. PROJECT ORGANISATION

3.1. Responsibilities and Roles

Employer and supervisor

Our employer is Patrick Bours (Associate professor, NISlab - HiG). He will also be a part time supervisor, and share this role with Simon McCallum (Associate professor, HiG). Simon will be our main resource regarding the technical side of the development. Patrick will assist us on theoretical questions.

Project leader

This role will be shared (among the members of the group) throughout the project, according to §4 in the DriSMo project rules (appendix A). At any given time we will have a specified leader, but who it is may vary. The leader's responsibilities is described in our rules, as well as in point 3.2 of this document.

Contact person

Our contact person is Fredrik Kvitvik. All formal contact between the group and external sources should go through him. He will keep in touch with the employer and supervisors, and try to arrange meetings when the group (leader) sees a need for this.

Webmaster

Jørn-André Myrland is our webmaster. It is his responsibility to get the project web up and running. He will also be responsible for the design, and further development during the project. The whole group will have access to update the site content, but the webmaster has the final say when it comes to how we structure and publish things.

Shared responsibilities

All areas can't be covered *specifically* by the above roles, but the group has a common understanding on how we share the smaller responsibilities among us. For example who is going to write the protocol from this weeks meeting, and similar tasks. If things don't work as intended, then of course the project leader will need to step in.

3.2. Procedures and rules in the group

Each group member have signed a contract where we agree to work at least 28 hours per week on the DriSMo project. Normally we will be working in our assigned room, 09.00-16.00 on weekdays except Monday. Deviations from this will be done by vocal agreement.

If any problems or conflicts arise, it is the project leader's responsibility to assess and solve them. All important topics should first be discussed in the group, and hopefully the majority will come to an agreement. In cases where no agreement can be made, the project leader will have a double vote. If the decision is directly concerning one of the group member, this member will have his voting rights revoked.

As the rules state, we will share the leader title on a four week basis. When the four week period has passed, the next person on the list will take over this responsibility.

If the group have any costs during the project, these will be shared between all group members. If one of the members pays on behalf of the group, a receipt will need to be presented.

The full (Norwegian) rule document is available in appendix A.

3.3. Tools

We have decided to use Google Docs when collaborating on documents during this project. In addition to this, we will use a subversion repository (and TortoiseSVN) to keep version control during the program development. For easy file sharing and additional backup we have created a shared online Dropbox folder. Our website is running on Joomla 1.5 CMS.

We will be writing the application with the java programming language, since we are developing on the Android platform. IntelliJ IDEA 10.X will be our programming environment, because it's optimized for Google Android programming, and has the latest SDK support.

4. PLANNING, MONITORING AND REPORTING

4.1. Software Development Methodology

When we discussed "agile vs traditional" software development methodology's (SDM), we came to a conclusion of using an agile SDM. We came to this conclusion because we are in need of agility, since the scope may change at a later stage. Since this is our first big project, we don't feel as confident in our estimates either. This is a disadvantage

when using a traditional SDM. The agile methods gives us the advantage of not having to plan everything to the last detail, early in the project.

We have discussed several agile methodologies of development relevant to this project, including evolutionary, incremental, XP (eXtreme Programming) and Scrum.

We considered using the evolutionary SDM, because it would give us an early visual overview of the service we will develop. Our belief is that XP is a more structured SDM, but the main problem of XP is that it's evolved around pair programming and this is a big drawback since we are only three programmers.

The incremental SDM would be very beneficial to use, since it gives us an early and complete list of specifications, and because we place emphasis on developing module by module. Incremental and iterative development is the way to go, but we are not convinced that these SDM's are structured enough for this project.

In the mist of the last section, we have decided to use Scrum as the SDM for this project. We chose Scrum because it's a structured and relevant SDM for this project and we also like the artifacts presented by Scrum. Scrum is also very open for changes, so this methodology would help us deal with modifications of the project description along the development process.

SDM modifications

As we were discussing the different methodologies, we agreed that pair programming would be a benefit to us, especially since we don't have a lot of experience. However, as mentioned when discussing XP, this would be difficult since we are three developers. We then looked at other solutions to gain some of the same advantages that pair programming offers, and has come up with our custom code validation routine. We will be using this as a part of our modified Scrum model.

Code validation means that when one developer is finished with a module or a relatively large chunk of code, at least one other person from the team will have a read-through before we move on to new tasks. The code creator will do a quick demo if necessary, but this process should be as effective as possible. Doing this on a regular basis, we will have a certain level of knowledge regarding the other developers work. This could prove very useful if someone is absent due to illness. It would also make it easier to help each other if we are stuck. There are several other benefits as well, but the important thing is that we get some sort of joint ownership of our code. The drawback of doing this is off course that it takes additional time. It is therefore important that we do this process efficient, so that what we gain will be worth it.

Scrum practice is against working overtime, which would have been nice if we could work a full week. But in reality, we only have 3-4 working days available. This means that we *may* have to expand the work hour limit, to satisfy the goals/demands of the current sprint. However, we do not consider overtime when constructing the sprint backlog.

Scrum spesification

Patrick Bours and Simon McCallum are involved as product owners. The person who is acting as the project leader, will also be acting as the Scrum master.

The sprint period will be 2 weeks, because we think anything longer than this will create too much “overhead”. We also need frequent feedback on the work that’s done, and the short sprints will help us. This gives us less chance to “stray the path”. Anything shorter than 2 weeks would be useless, because you haven't got enough time to get in a good working flow, and the goals would be too small for each sprint.

4.2. Plan for status meeting and decision points

Since we will be using Scrum, we are going to have sprint planning meetings. These meetings will be at the start of each sprint, and it is here we will choose what to do in the new sprint (construct the sprint backlog).

After each sprint we will have a “sprint retrospective” meeting, focusing on what we can do better next sprint. This will help us improve on our weaknesses, and avoid doing the same mistakes twice. We also plan on having a “sprint review” meeting with our employer and supervisor after each sprint, to make sure we are headed in the right direction.

5 ORGANIZATION OF QUALITY ASSURANCE

5.1. Documentation, standard applications and source code

We will be using JavaDoc and in-line comments to document the source code. The reason why we choose to use JavaDoc is because it’s fully integrated with IDEA, and gives a good overview of the whole project.

We will be commenting the source code along the way, and make sure the JavaDoc documentation is up to date each Friday. If we wait any longer, it would only lead to us having to catch up on large chunks of code at some point. This could possibly screw us when it comes to keeping up with our deadlines.

Even if we do the JavaDoc update on Fridays, it is still very important that we comment our code in a normal fashion as soon as possible after we write it. If we don’t, we could end up wasting our Fridays with trying to figure out what we actually wrote on Tuesday.

5.2. Risk assessment

Risk is defined as the product of probability and consequence. In the table below we have made a assessment of problems that can occur along the way. Detailed description of the relevant issues will be described below.

Description	Probability	Consequence	Risk assessment
Long term illness	2	7	14%
Lack of competence	4	8	32%
Too high technological requirements	4	8	32%
Bad time estimates	4	7	28%
Loss of data/equipment	1	8	8%
Fatal changes	2	9	18%

Preventive solutions for critical risks

As the table above illustrates, the lack of competence among the developers (us) could be a huge problem for the project. We have no experience of developing on the Android platform, therefore it will be a challenge. To avoid this problem we will be consulting with our supervisor Simon. He teaches a mobile programming course, as well as having experience in many other relevant fields. We will also be using code validation, as discussed in point 4.1.

The risk of the technologic requirements being too high is also something we have to take into account. To lower the performance requirements of the phone, we want to let the users choose what instruments to use. The accelerometer is the exception to this, as we need a minimum of data to evaluate the driving. Allowing the user to deselect sensors, they can also avoid problems concerning the cost of using the GPS system. This will hopefully lead to the application being tested by a larger amount of people. Of

course we would recommend using as many sensors and measurements as possible, to generate the best feedback possible.

The last of our three most critical risks, is the possibility that we have made some bad estimates in the planning phase of the project. This isn't unlikely considering our inexperience with this kind of developing platform. To make sure the consequence of this is as low as possible, we need to be extra aware when we prioritize what stories to pick first from the product backlog. Then we at least make sure that the most crucial parts are done according to schedule.

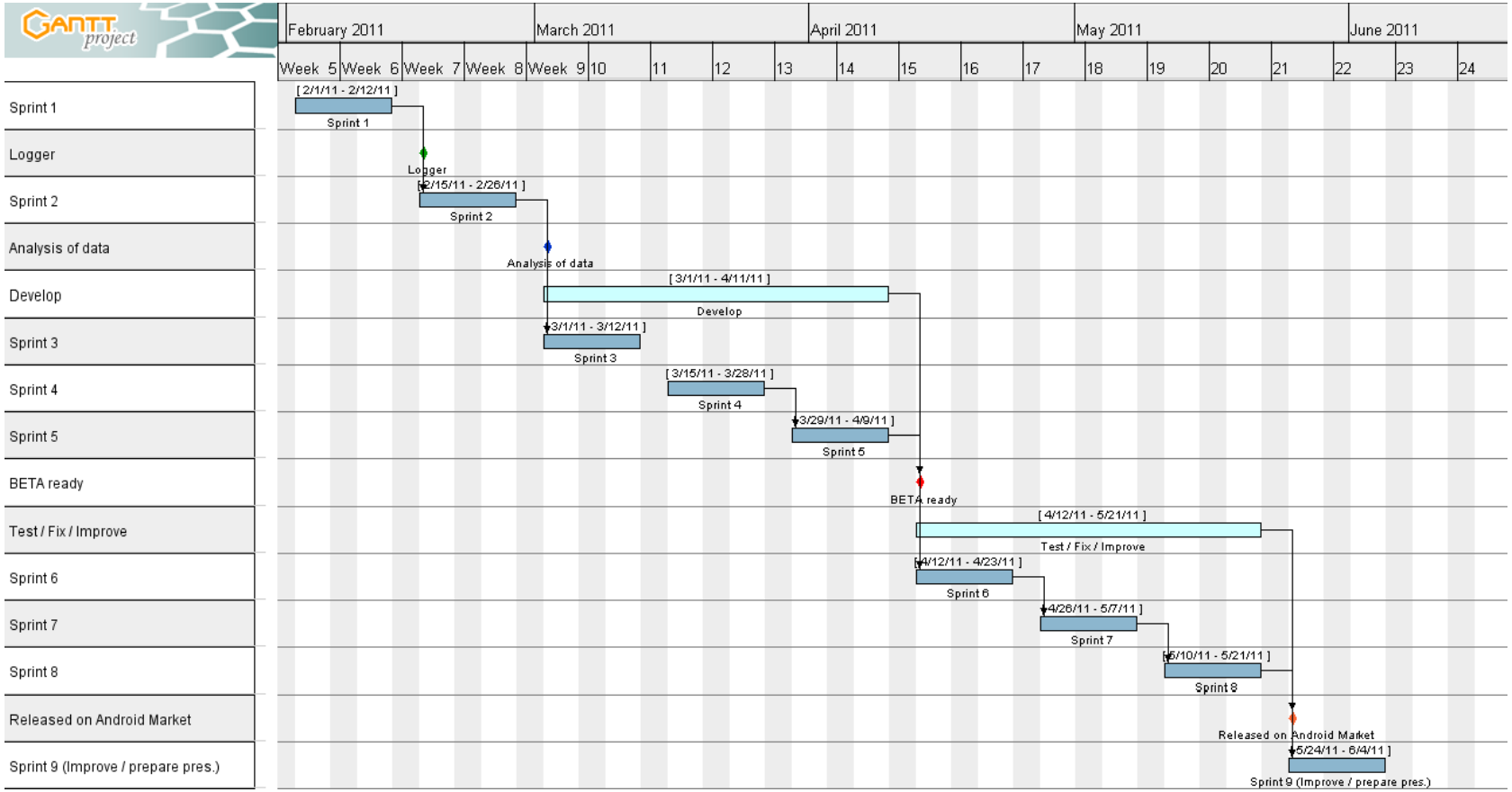
Critical success factors

For this application to be successful, it is essential that the users agree with the way his/her driving is evaluated, and that the evaluation is as proper as possible. At the end of the day, this is largely depending on the competence of the development team. In the table above, you can also see that this issue needs to be our top priority. For preventive measures regarding this, see the above section.

To prevent a mismatch between the quality algorithm of the application and the user's expectations, we need to look at scientific data for driving quality measurements, related to the sensors we'll be using. We should also gather a good amount of human input on the subject.

6. PLAN FOR IMPLEMENTATION

As we will be using iterative development methodology, the gantt chart will be very interval based with many "sprints". In the gantt chart below, we have put in a more describing parent node for some sprints and some milestones, marked as diamonds in different colors. The specifics of each sprint will be determined in the sprint planning meeting.


GANTT project
February 2011
March 2011
April 2011
May 2011
June 2011
Week 5
Week 6
Week 7
Week 8
Week 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Sprint 1
Logger
Sprint 2
Analysis of data
Develop
Sprint 3
Sprint 4
Sprint 5
BETA ready
Test / Fix / Improve
Sprint 6
Sprint 7
Sprint 8
Released on Android Market
Sprint 9 (Improve / prepare pres.)
[2/1/11 - 2/12/11]
[2/15/11 - 2/26/11]
[3/1/11 - 4/11/11]
[3/1/11 - 3/12/11]
[3/15/11 - 3/28/11]
[3/29/11 - 4/9/11]
[4/12/11 - 5/21/11]
[4/12/11 - 4/23/11]
[4/26/11 - 5/7/11]
[5/10/11 - 5/21/11]
[5/24/11 - 6/4/11]

7. APPENDIX A - Regler for bachelorprosjektet DriSMo

§1 - Mandager er i utgangspunktet en prosjektfri dag. Arbeid ut over forhåndsavtalt arbeidstid (9-16 øvrige hverdager) må bestemmes av flertall i gruppa. Det kan søkes fritak, men gruppemedlemmet er da pliktig til å jobbe inn timene ved en senere anledning. Prosjektlederen har anledning til å avslå et slikt fritak, hvis det vurderes som uforsvarlig i forhold til prosjektets tidsfrister.

§2 - Når beslutninger skal tas, løses uenigheter ved hjelp av avstemning. I tilfeller der flertall ikke oppnås, skal det søkes ekstern rådføring med relevant personell, før ny avstemning utføres. Hvis dette ikke fører frem etter gjentatte forsøk, vil fungerende prosjektleder bli gitt en dobbelstemme. Hvis avstemningen handler om problemer rundt et bestemt gruppemedlem, vil ikke denne personen ha stemmerett i den aktuelle saken.

§3 - Alle skal møte presis til avtalte tidspunkter. Gruppemedlemmet med høyest ukentlig ugyldig fravær, har ansvar for innkjøp av kaffekaker til ukeslutt-møtet på fredager. Hvis dette ikke oppleves som tilstrekkelig sanksjonering, kontaktes høyere makter.

§4 - Rollen som prosjektleder vil gå på rundgang blant gruppemedlemmene. Ny prosjektleder inntrer hver 4. uke, f.o.m. uke 3. Prosjektlederens maktposisjon er som beskrevet i dette dokumentet. I forhold som ikke nevnes i dette dokumentet, er gruppemedlemmene likestilt. Ved særlig misnøye med prosjektlederen, kan de øvrige gruppemedlemmene ved 100% enighet inndra prosjektlederens tittel, å gå videre i rulleringen.

§5 - Kostnader som må dekkes av prosjektgruppa skal fordeles likt blant gruppemedlemmene. Hvis noen lagt ut for slike kostnader, skal det foreligge kvittering for dette. Når de andre på gruppa har betalt sin andel, blir dette markert i regnskapet.

Signatur

_____
Jørn André Myrland

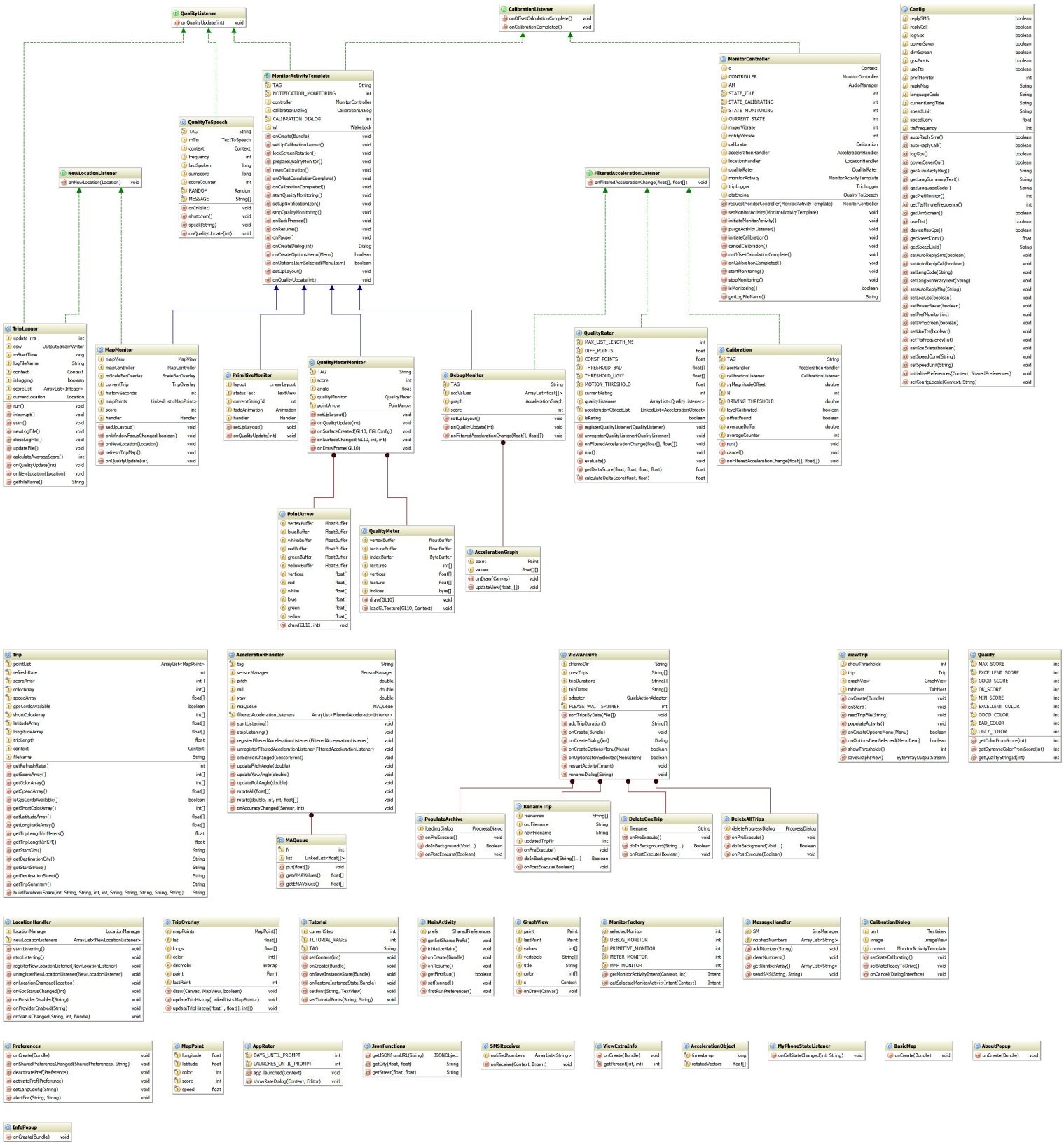
_____
Fredrik Kvitvik

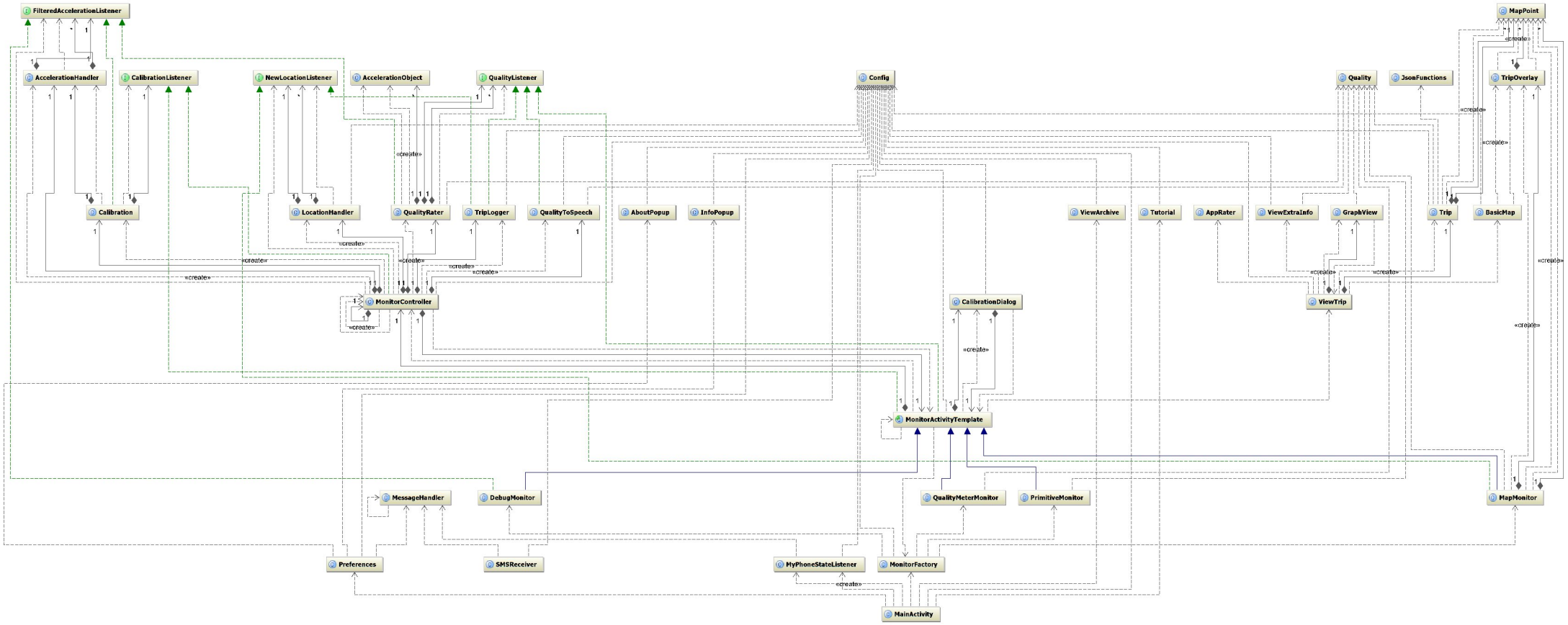
_____
Fredrik Hørtvedt

E Design Schemas

Hint: All schemas are vector based, therefore exploit your PDF client's zoom ability.

The last UML diagram is auto-generated, and therefore contains a lot of unnecessary dependencies.





F Source code

The source code of this project is provided with this document. It is located in the `source` folder.

QualityRater.java

```
1  package com.drismo.logic;
2
3  import android.hardware.SensorManager;
4  import android.os.SystemClock;
5  import android.util.Log;
6  import com.drismo.model.AccelerationObject;
7  import com.drismo.model.Quality;
8
9  import java.util.ArrayList;
10 import java.util.LinkedList;
11
12 /**
13  * Handles quality rating and distributing updates to all the registered listeners.
14  */
15 public class QualityRater extends Thread implements FilteredAccelerationListener {
16
17     private static final int MAX_LIST_LENGTH_MS = 500;
18
19     /**
20      * Used to calculate a delta score based on changes in the force the vehicle
21      * is exposed to, over a short period of time (MAX_LIST_LENGTH_MS).
22      */
23     private static final float DIFF_POINTS    = ((100f/3f) * 0.7f);
24
25     /**
26      * Used to calculate a delta score based on the constant force the vehicle
27      * is exposed to, over a short period of time (MAX_LIST_LENGTH_MS).
28      */
29     private static final float CONST_POINTS = ((100f/3f) * 0.3f);
30
31                                     //Representing vectors {  X   ,  Y  ,  Z }
32     private static final float THRESHOLD_BAD[]= new float[]  { 2.0f, 1.0f, 0.8f};
33     private static final float THRESHOLD_UGLY[]= new float[] { 3.0f, 2.2f, 1.6f};
34     private static final float MOTION_THRESHOLD = 0.1f;
35
36     private int currentRating = 0;
37
38     private volatile ArrayList<QualityListener> qualityListeners =
39                                                 new ArrayList<QualityListener>();
40
41     private final LinkedList<AccelerationObject> accelerationObjectList;
42
43     private boolean isRating;
44
45     /**
46      * Constructs the accelerationObjectList.
47      */
48     public QualityRater(){
49         accelerationObjectList = new LinkedList<AccelerationObject>();
50     }
```

```
51
52      /**
53       * Adds a quality listener and starts the thread if it is the first listener
54       * added to the list.
55       * @param listener The listener to add.
56       */
57      public synchronized void registerQualityListener(QualityListener listener){
58
59          if(qualityListeners.size() == 0){
60              Log.d("qualityListeners", "STARTED!");
61              isRating = true;
62              start();
63          }
64
65          if(!qualityListeners.contains(listener))
66              qualityListeners.add(listener);
67      }
68
69      /**
70       * Removes the specified quality listener and stops the thread if the
71       * listener list is empty.
72       * @param listener The given listener to stopMonitoring.
73       */
74      public synchronized void unregisterQualityListener(QualityListener listener){
75          qualityListeners.remove(listener);
76
77          if(qualityListeners.size() == 0) {
78              isRating = false;
79              interrupt();
80              accelerationObjectList.clear();
81          }
82      }
83
84      /**
85       * On each entry an AccelerationObject (containing the vectors and timestamp)
86       * is added to the <code>accelerationObjectList</code>, and deletes all the
87       * objects that are older than <code>MAX_LIST_LENGTH_MS</code>.
88       * @param filteredVectors Not used here.
89       * @param RotatedVectors This iadded as the vectors in the
90       * <code>AccelerationObject</code>
91       */
92      public void onFilteredAccelerationChange(float[] filteredVectors,
93                                               float[] RotatedVectors){
94
95          synchronized (accelerationObjectList){
96              AccelerationObject current =
97                      new AccelerationObject(RotatedVectors.clone(),
98                                             SystemClock.uptimeMillis());
99                  if(accelerationObjectList.size() > 0){
100                     AccelerationObject last;
101
102                     do{                  //Remove all objects which is timed out:
103                        last = accelerationObjectList.removeLast();
104                     }while(accelerationObjectList.size() > 0 &&
105                           current.timestamp - last.timestamp>MAX_LIST_LENGTH_MS);
106
107                     if(last != null)        //put back the object that was valid
108                         accelerationObjectList.addLast(last);
109                 }
110                 accelerationObjectList.addFirst(current);
111             }
112
```

```
113     }
114
115     /**
116      * Evaluates the driving quality and updates all the listeners.
117      */
118     public void run() {
119         currentRating = 1600;
120
121         synchronized (this){
122
123             while(isRating){                //while the rating isn't stopped
124
125                 evaluate();                   //Evaluate and update current score
126
127                 for(QualityListener listener : qualityListeners )// Notify
128                     listener.onQualityUpdate(currentRating);     // listeners
129
130                 try {                   //Wait half of the list length, to overlap
131                     wait(MAX_LIST_LENGTH_MS/2);
132                 } catch (InterruptedException e) {
133                     e.printStackTrace();
134                 }
135             }
136         }
137     }
138
139     /**
140      * Evaluate the new quality rating by adding the sum of each delta
141      * score (calculated by evaluating changes and constant force
142      * measured by each vector).
143      */
144     private void evaluate(){
145         if(accelerationObjectList.size() >0){
146             synchronized (accelerationObjectList){
147
148                 final AccelerationObject first =
149                                     accelerationObjectList.getFirst();
150
151                 final float currentValues[] = first.rotatedVectors;
152
153                 if(currentValues[1] > MOTION_THRESHOLD ||
154                    currentValues[1] < -MOTION_THRESHOLD){
155
156                     float minValues[] = currentValues.clone();
157                     float maxValues[] = currentValues.clone();
158
159                     float diffValues[] = new float[3];
160                     float deltaScore=0;
161                         //get the max/min values, used to calculate max difference
162                     for(AccelerationObject previous : accelerationObjectList){
163                         float previousValues[] = previous.rotatedVectors;
164                         for(int i = 0; i < 3; i++){
165                             if(minValues[i] > previousValues[i]){
166                                 minValues[i] = previousValues[i];
167                             }else if(maxValues[i] < previousValues[i]){
168                                 maxValues[i] = previousValues[i];
169                             }
170                         }
171                     }
172                                     //calculate the delta score for each axis
173                     for(int i = 0; i < 3; i++){
174                         diffValues[i] = maxValues[i] - minValues[i];
```

```
175                             //calculate the delta score based on the max diff.
176                         deltaScore += getDeltaScore(diffValues[i],
177                                 THRESHOLD_BAD[i], THRESHOLD_UGLY[i], DIFF_POINTS);
178
179                             //calculate the delta score based on the constant force
180                         if(i == 2)
181                             deltaScore += getDeltaScore(currentValues[i] -
182                                     SensorManager.STANDARD_GRAVITY,
183                                     THRESHOLD_BAD[i], THRESHOLD_UGLY[i],
184                                     CONST_POINTS);
185                         else
186                             deltaScore += getDeltaScore(currentValues[i],
187                                                 THRESHOLD_BAD[i],
188                                                 THRESHOLD_UGLY[i],
189                                                 CONST_POINTS);
190                     }
191                                                  //set the current score
192                     currentRating += deltaScore;
193
194                            //if the new score is less than min, new score == min
195                     if(deltaScore < 0 && currentRating < Quality.MIN_SCORE)
196                         currentRating = Quality.MIN_SCORE;
197                 }
198             }
199         }
200     }
201
202     /**
203      * Get delta score based on the current state given.
204      * @param vector Current acceleration vector, can be either Good, the bad
205      * or the ugly.
206      * @param badThres Threshold relative to the given vector, representing
207      * bad sectors.
208      * @param uglyThres Threshold relative to the given vector, representing
209      * ugly sectors.
210      * @param points unit used to calculate the delta currentRating.
211      * @return returns the delta score to update the current score.
212      */
213     public float getDeltaScore(float vector, float badThres, float uglyThres,
214                                                                 float points){
215                                             // Calculate the delta currentRating:
216         if(vector > uglyThres || vector < -uglyThres ) {  //given a ugly vector
217             return calculateDeltaScore(0f, points * 3);
218         }
219
220         else if(vector > badThres || vector < -badThres ) {// given a bad vector
221             return calculateDeltaScore(0.5f, points * 2 );
222         }
223
224         else {                                              // given a good vector
225             return calculateDeltaScore(1.0f, points / 2 );
226         }
227     }
228
229     /**
230      * This is used to calculate a score relative to the current
231      * score and quality.
232      * @param outcome this is the outcome of the current driving quality.
233      * @param points unit used to calculate the delta score.
234      * @return Returns the new delta score
235      */
236     public final float calculateDeltaScore(final float outcome,
```

```
                                        final float points){

        final float relativeScore = 1000*outcome + 500;
                                                //calculate the expected score
        final float expectedOutcome = (float) (1 / ( 1 + Math.pow(10.0,
                                    (relativeScore - currentRating)/100.0)));

                                                //return the new delta score
        return points * (outcome - expectedOutcome);
    }
}
```

Calibration.java

```
package com.drismo.logic;

/**
 * Calibrates the (roll, pitch and yaw) angles of the device <b>RELATIVE</b> to
 * the cars position and driving direction. By calibrating we can rotate the
 * acceleration vectors relative to the car, which means we can use these vectors
 * to analyze and rate the quality of driving.
 */
public class Calibration extends Thread implements FilteredAccelerationListener {

    private static final String TAG = "Calibration";

    private AccelerationHandler accHandler;
    private CalibrationListener calibrationListener;

    private double xyMagnitudeOffset = 0;

    private static final int N = 25;                        // The average factor.

    private static final double DRIVING_THRESHOLD = 0.5;    // Used to determine
                                                            // if we are driving.
    private boolean levelCalibrated = false;
    private boolean offsetFound = false;

    private double averageBuffer = 0;               // Some buffer variables
    private int averageCounter = 0;                 // used to calculate average.

    /**
     * Stores the acceleration handler and the calibration listener.
     * Starts the calibration thread.
     * @param ah The acceleration handler.
     * @param cl The object that listens for changes in calibration.
     */
    public Calibration(AccelerationHandler ah, CalibrationListener cl) {
        accHandler = ah;
        calibrationListener = cl;
        this.start();
    }

    /**
     * Waits 1 second to reduce touch vibration/noise,
     * then initiates the calibration.
     */
    public synchronized void run() {
        try {
                                                // Waits 1 second to remove
            wait(1000);                         //excessive touch vibration/noise.

```

```
49              // Registers this object to listen for filtered acceleration values.
50              accHandler.registerFilteredAccelerationListener(this);
51
52          } catch (InterruptedException e) {
53              e.printStackTrace();
54          }
55      }
56
57      /**
58       * Cancels the calibration.
59       */
60      public void cancel() {                          // Unregister this object and
61          accHandler.unregisterFilteredAccelerationListener(this);
62          this.interrupt();                           //interrupt the running thread.
63      }
64
65
66      /**
67       * Computes the roll, pitch and yaw angles relative to the car. This means
68       * that (by rotating the acceleration vectors) we can detect turns in the
69       * X axis, acceleration in the Y axis and bumps in the Z axis.
70       * Noise reduced acceleration values is given from the AccelerationHandler
71       * object, on change.
72       *
73       * @param filteredVectors The original weighted moving average
74       * acceleration vectors (X,Y,Z).
75       * @param rotatedVectors The rotated weighted moving average
76       * acceleration vectors (X,Y,Z).
77       * @see AccelerationHandler#onSensorChanged(android.hardware.SensorEvent)
78       */
79      public synchronized void onFilteredAccelerationChange(float[] filteredVectors,
80                                                            float[] rotatedVectors){
81
82          if(!offsetFound){      // If driving offset not found, we need to find it.
83
84              if(!levelCalibrated ){              // If we just started calibrating
85                  // We need to find the ROLL and PITCH and rotate the given values.
86                  double roll = StrictMath.atan2(filteredVectors[0],
87                                                 filteredVectors[2]);
88                  AccelerationHandler.rotate(roll, 0, 2, filteredVectors);
89
90                  double pitch = StrictMath.atan2(filteredVectors[1],
91                                                  filteredVectors[2]);
92                  AccelerationHandler.rotate(pitch, 1, 2, filteredVectors);
93
94                  accHandler.updateRollAngle(roll);          // Update tilt angles:
95                  accHandler.updatePitchAngle(pitch);
96                  accHandler.updateYawAngle(0);
97
98                  levelCalibrated = true;
99              }else{                    // Level is calibrated, find the XY offset:
100
101                 double magnitude = Math.hypot(rotatedVectors[0],
102                                               rotatedVectors[1]);
103                  //If this is the first entry OR if the new magnitude is relatively
104                 if(averageCounter == 0 ||        // the same value as the last one:
105                         ( magnitude < (averageBuffer/averageCounter) + 0.05  &&
106                           magnitude > (averageBuffer/averageCounter) - 0.05 ) ){
107
108                     averageBuffer += magnitude;        // Sum the magnitude of XY:
109                     averageCounter++;
110                                     // If we have enough values to get the avg:
```

```
111                     if(averageCounter >= N){            // Get the average offset:
112                         xyMagnitudeOffset = averageBuffer / averageCounter;
113                         averageBuffer = 0;          // Reset the average variables:
114                         averageCounter = 0;
115                         offsetFound = true;          // We found the offset!
116                                                             // Update the listener.
117                         calibrationListener.onOffsetCalculationComplete();
118 
119                     }            // If the new magnitude was not relatively like,
120                 }else{           //  the vehicle is in motion!
121                     levelCalibrated = false;       // Reset the level calibration.
122                     averageBuffer = 0;             // Reset the average variables:
123                     averageCounter = 0;
124                 }
125             }
126 
127         } else { //If we got the XY magnitude, we can detect the driving direction
128 
129                                 // Get the magnitude of X and Y minus the offset.
130             double drivingDirectionMagnitude = Math.hypot(rotatedVectors[0],
131                                                 rotatedVectors[1]) - xyMagnitudeOffset;
132 
133                                                     // If true, we are in motion:
134             if( drivingDirectionMagnitude > DRIVING_THRESHOLD){
135 
136                                     // Add the new yaw angle in the avg buffer:
137                 averageBuffer += Math.atan2(rotatedVectors[0], rotatedVectors[1]);
138                 averageCounter++;
139                           // If we have enough angles to calculate the direction:
140                 if(averageCounter > N){ // Get the average yaw angle, and update
141                     double yaw = averageBuffer / averageCounter;
142                     accHandler.updateYawAngle(yaw); // the acceleration handler.
143 
144                                       // We are done calibrating, so we unregister.
145                     accHandler.unregisterFilteredAccelerationListener(this);
146                                                         // Update the listener.
147                     calibrationListener.onCalibrationCompleted();
148                 }
149 
150             }
151         }
152     }
153 
154 }
```

AccelerationHandler.java

```
1  package com.drismo.logic;
2 
3  import android.hardware.Sensor;
4  import android.hardware.SensorEvent;
5  import android.hardware.SensorEventListener;
6  import android.hardware.SensorManager;
7  import android.util.Log;
8  import java.util.*;
9 
10 /**
11  * This object handles everything regarding collecting and filtering acceleration
12  * data. To listen for filtered acceleration data, implement the
13  * <code>FilteredAccelerationListener</code> interface.
14  * @see com.drismo.logic.FilteredAccelerationListener
15  */
```

```
16 public class AccelerationHandler implements SensorEventListener {
17     private static final String tag = "AccelereationHandler";
18 
19     private SensorManager sensorManager = null;
20 
21     private double pitch;
22     private double roll;
23     private double yaw;
24 
25     private MAQueue maQueue;
26     private final ArrayList<FilteredAccelerationListener>
27     filteredAccelerationListeners = new ArrayList<FilteredAccelerationListener>();
28 
29     /**
30      * Set up the sensor manager and the moving average queue.
31      * @param sm The sensor manager used to register a accelerometer data listener
32      */
33     public AccelerationHandler(SensorManager sm){
34         sensorManager = sm;
35         maQueue = new MAQueue();
36     }
37 
38 
39     /**
40      * Start listening for acceleration data.
41      */
42     public void startListening() {
43         sensorManager.registerListener(this,
44                 sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
45                                     SensorManager.SENSOR_DELAY_FASTEST);
46     }
47 
48     /**
49      * Stop listening for acceleration data.
50      */
51     public void stopListening() {
52         sensorManager.unregisterListener(this);
53     }
54 
55     /**
56      * Add an object to listen for filtered acceleration data.
57      * @param listener The listener to add.
58      */
59     public void registerFilteredAccelerationListener(
60                                     FilteredAccelerationListener listener){
61         if(filteredAccelerationListeners.size() == 0){
62             startListening();
63         }
64 
65         if(!filteredAccelerationListeners.contains(listener))
66             filteredAccelerationListeners.add(listener);
67     }
68 
69     /**
70      * Remove an added listener from the list.
71      * @param listener The listener to remove.
72      */
73     public void unregisterFilteredAccelerationListener(
74                                     FilteredAccelerationListener listener){
75         filteredAccelerationListeners.remove(listener);
76 
77         if(filteredAccelerationListeners.size() == 0){
```

```
78              stopListening() ;
79          }
80      }
81
82      /**
83       * Filters the incoming acceleration data using the WMA or EMA algorithm,
84       * and fires the callback method of all the listeners.
85       * @param event Sensor event. Only using Sensor.TYPE_ACCELEROMETER.
86       * @see FilteredAccelerationListener#onFilteredAccelerationChange(float[],
87       *                                                                float[])
88       */
89      public void onSensorChanged(SensorEvent event) {
90
91          if(event.accuracy == SensorManager.SENSOR_STATUS_UNRELIABLE) {
92              return;
93          }
94
95          int sensorType = event.sensor.getType();
96  //        int sensorType = event.type;
97          if (sensorType == Sensor.TYPE_ACCELEROMETER) {
98              maQueue.put(event.values.clone());
99              float[] filteredAcceleration = maQueue.getEMAValues();
100
101             /**
102              * If vectors are noise reduced, and listeners is registered:
103              */
104             if(filteredAcceleration != null &&
105                                 filteredAccelerationListeners.size() > 0){
106
107                 float[] rotatedAcceleration = filteredAcceleration.clone();
108                 rotateAll(rotatedAcceleration);
109
110                 for(FilteredAccelerationListener listener :
111                                             filteredAccelerationListeners)
112                     listener.onFilteredAccelerationChange(filteredAcceleration,
113                                                           rotatedAcceleration);
114             }
115
116         }
117
118     }
119
120     /**
121      * Updates the pitch angle relative to the vehicle.
122      * @param YZ The new angle (radians)
123      */
124     public void updatePitchAngle(double YZ){
125         pitch = YZ;
126     }
127
128     /**
129      * Updates the yaw angle relative to the vehicle.
130      * @param XY The new angle (radians)
131      */
132     public void updateYawAngle(double XY){
133         yaw = XY;
134     }
135
136     /**
137      * Updates the roll angle relative to the vehicle.
138      * @param XZ The new angle (radians)
139      */
```

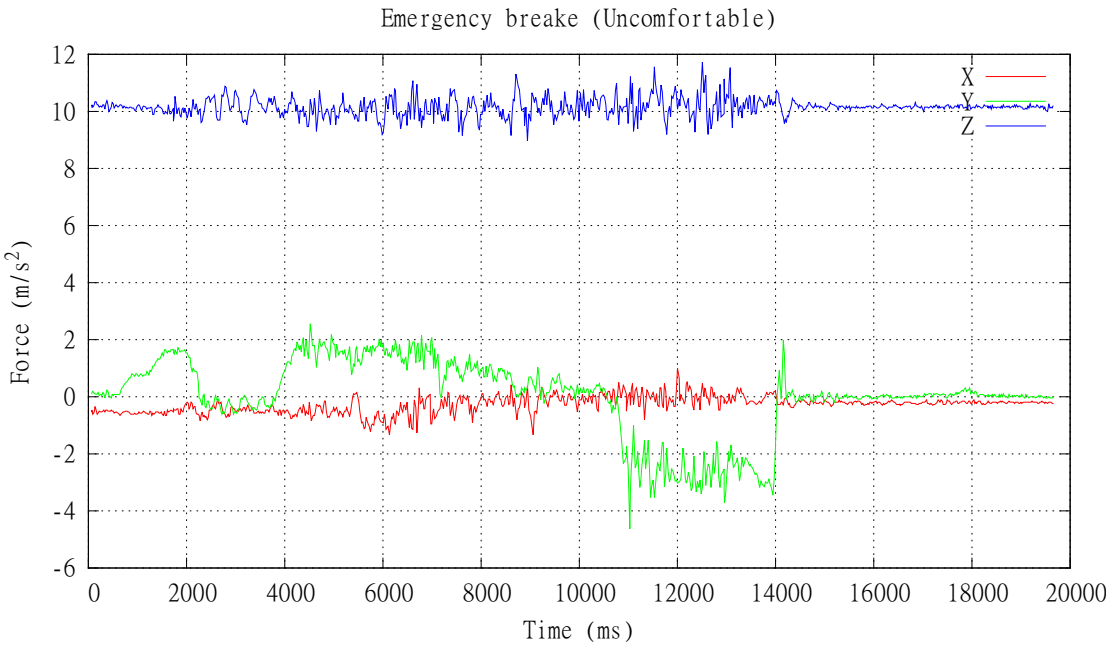
```
140     public void updateRollAngle(double XZ){
141         roll = XZ;
142     }
143 
144     /**
145      * Rotates all the acceleration vectors given as parameter, according to
146      * the <code>roll</code>, <code>pitch</code> and <code>yaw</code> angles.
147      * @param vectors The acceleration vectors
148      */
149     public synchronized void rotateAll(float[] vectors){
150         rotate(roll, 0, 2, vectors);
151         rotate(pitch, 1, 2, vectors);
152         rotate(yaw, 0, 1, vectors);
153     }
154 
155     /**
156      * Rotates two vectors using polar rotation, using each vector
157      * as a coordinate.
158      * @param radAngle The rotation angle in radians.
159      * @param indexX Index of the X coordinate
160      * @param indexY Index of the Y coordinate
161      * @param vectors Acceleration vector array.
162      */
163     public static void rotate(double radAngle, int indexX, int indexY,
164                                                            float[] vectors){
165         float tempY = vectors[indexY];
166         vectors[indexY] = (float) (vectors[indexX] * Math.sin(radAngle) +
167                                    vectors[indexY]  * Math.cos(radAngle));
168 
169         vectors[indexX] = (float) (vectors[indexX] * Math.cos(radAngle) -
170                                    tempY * Math.sin(radAngle));
171     }
172 
173     /**
174      * An inner static class used to remove noise from the acceleration values.
175      * Keeps a simple FIFO queue of N acceleration values, used to filter out
176      * noise using the Weighted/Exponential Moving Average algorithm.
177      */
178     private static final class MAQueue {
179 
180         /**
181          * N elements in the queue.
182          */
183         private final int N = 31;
184 
185         /**
186          * The queue of vectors.
187          */
188         private LinkedList<float[]> list = new LinkedList<float[]>();
189 
190         /**
191          * Puts the acceleration vector in the FIFO queue, and removes
192          * excessive vectors.
193          * @param v The XYZ acceleration vectors.
194          */
195         public synchronized void put(float v[]) {
196             list.addLast(v);
197             if(list.size() > N)
198                 list.removeFirst();
199         }
200 
201         /**
```

```
202          * Calculates the WMA value of the element i in the queue, based on
203          * the <b>Weighted Moving Average</b> algorithm.
204          * @return The weighted moving average value of the element i.
205          */
206         public float[] getWMAValues(){
207             float returnValue[] = new float[3];
208 
209             if(list.size() < N)    {       // If the list is not complete, we
210                 return null;               //  can't calculate the moving average.
211             }
212 
213             else{                       // If enough elements to get calculate WMA:
214                 int i = (N / 2) + (N % 2);   //  get the current element position
215                                              // For each acceleration vector:
216                 for(int vector = 0; vector < 3; vector++){
217 
218                     int denominator = 0;
219                     float avgSum = 0;
220                                         // Go from element i-(N/2) to i+(N/2):
221                     for(int j = -(N / 2); j < i; j++){ //Get the multiplier/weight
222                         int multiplier = (j < 0) ? i + j : i - j;
223                                           // Add multiplier to the denominator:
224                         denominator += multiplier;
225                                         // Add the weighted element to the avg sum:
226                         avgSum += multiplier * list.get((i+j)-1)[vector];
227                     }
228                                           // Calculate the WMA of this vector.
229                     returnValue[vector] = avgSum / denominator;
230                 }
231             }
232 
233             return returnValue;
234         }
235 
236         /**
237          * Calculates the EMA value of the element i in the queue, based on the
238          * <b>Exponential Moving Average</b> algorithm.
239          * @return The exponential moving average value of the element i.
240          */
241         private synchronized float[] getEMAValues(){
242             float returnValue[] = new float[3];
243 
244             if(list.size() < N)    {        // If the list is not complete, we
245                 return null;               //can't calculate the moving average.
246             }
247 
248             else{                      // If enough elements to get calculate EMA:
249                 int i = (N / 2) + (N % 2);  //  get the current element position
250                                             // For each acceleration vector:
251                 for(int vector = 0; vector < 3; vector++){
252 
253                     int denominator = 0;
254                     float avgSum = 0;
255                     int multiplier = 0;
256                                           // Go from element i-(N/2) to i+(N/2):
257                     for(int j = -(N / 2); j < i; j++){
258                                           // Calculate the exponential multiplier:
259                         multiplier = (j < 0) ? N/(1+-j) : N/(j+1) ;
260                                           // Add multiplier to the denominator:
261                         denominator += multiplier;
262                                           // Sum the exponential weighted elements:
263                         avgSum += multiplier *  list.get((i+j)-1)[vector];
```

```
264                     }
265                                         // Calculate the EMA of this vector:
266                     returnValue[vector] = avgSum / denominator;
267                 }
268             }
269             return returnValue;
270         }
271     }
272 
273     /** Never used */
274     public void onAccuracyChanged(Sensor sensor, int i) { }
275 }
```

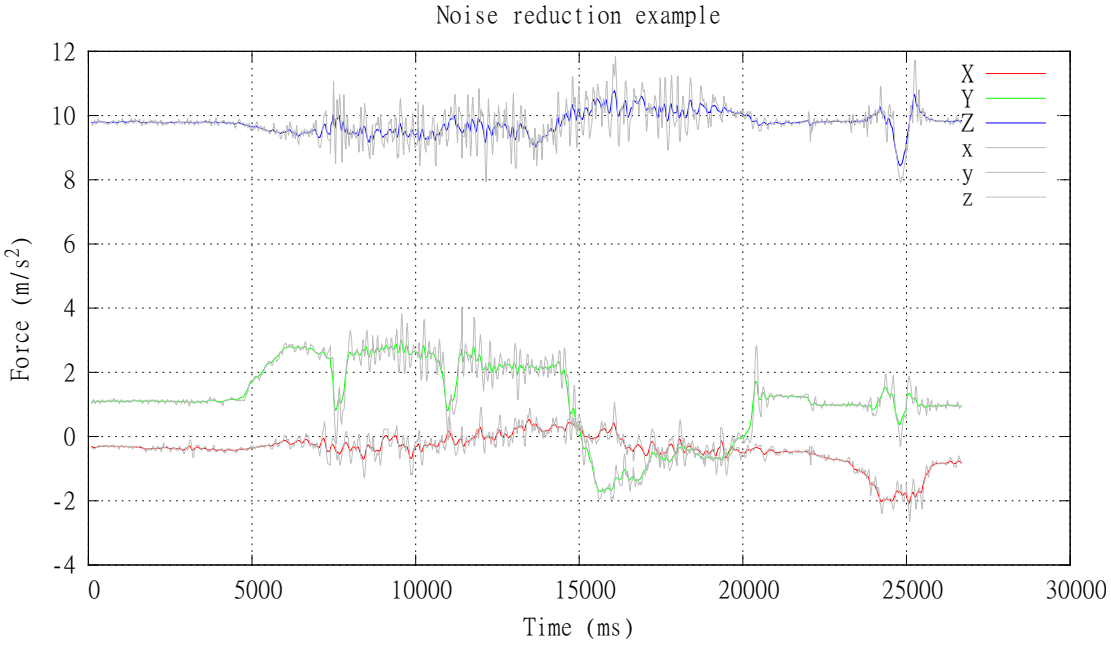
G Collected data

In this appendix you will find some of the collected data. The first graph is without noise reduction. The second graph shows the difference between a noise reduced and normal graph. The rest of the graphs are noise reduced.



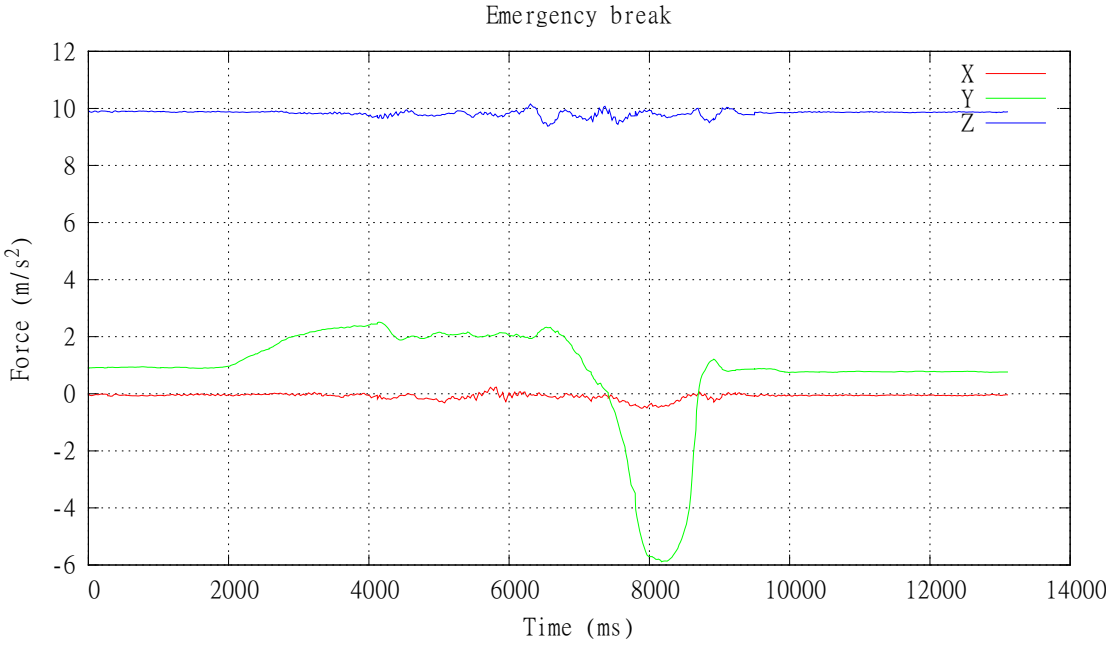
Test 1: Emergency brake.

We see from the graph that the vehicle accelerates rapidly, followed by an emergency brake at high speed. When braking there is a big change in acceleration (uncomfortable), first from 2m/s^2 to 0m/s^2 followed by 0m/s^2 down to -4m/s^2. This was executed in the winter, hence the long braking period. This is unfiltered acceleration values, with no form of noise reduction.



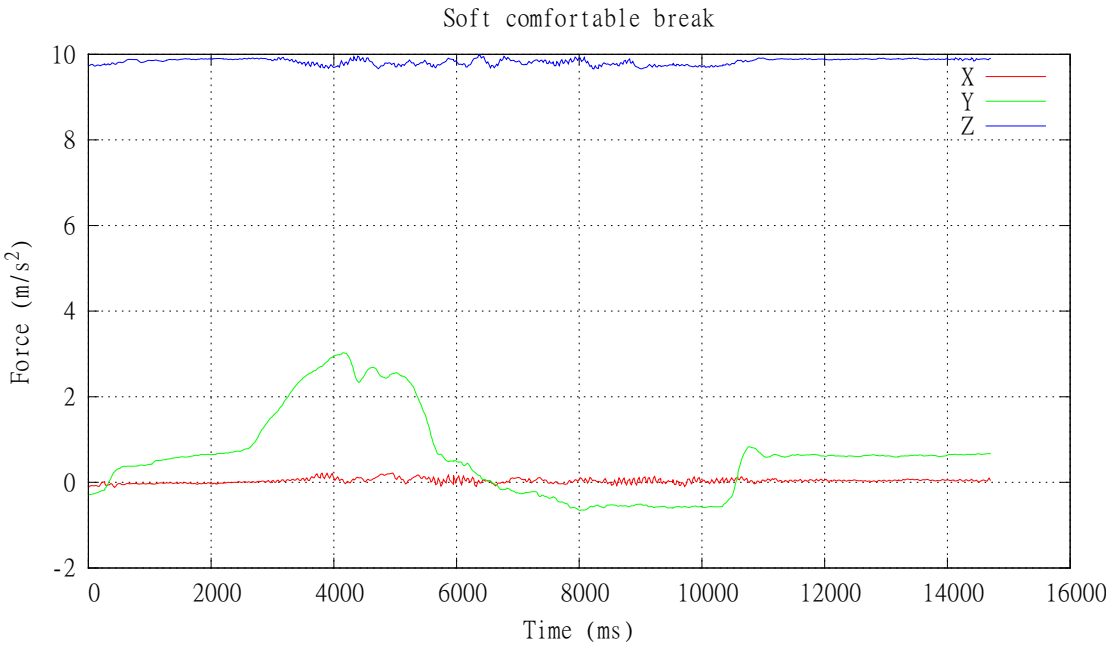
Example of original data (gray) and noise reduced data (colors).

The graph shows the difference between the original acceleration values, and the noise reduced acceleration values. At this test, we did not calibrate the device. We tried to manually align the device in relation to the vehicle. As the graph illustrates, the device was not entirely aligned with the vehicle. The initial y value is around 1m/s^2, which should have been 0m/s^2. The other axes is also slightly off. This test contains 2 gear shifts (at 7000ms and 11000ms), followed by an abrupt brake (from 15000ms to 20000ms). We clearly see that the noise reduction do not remove any critical data; it just smooths the acceleration values, removing excessive noise.



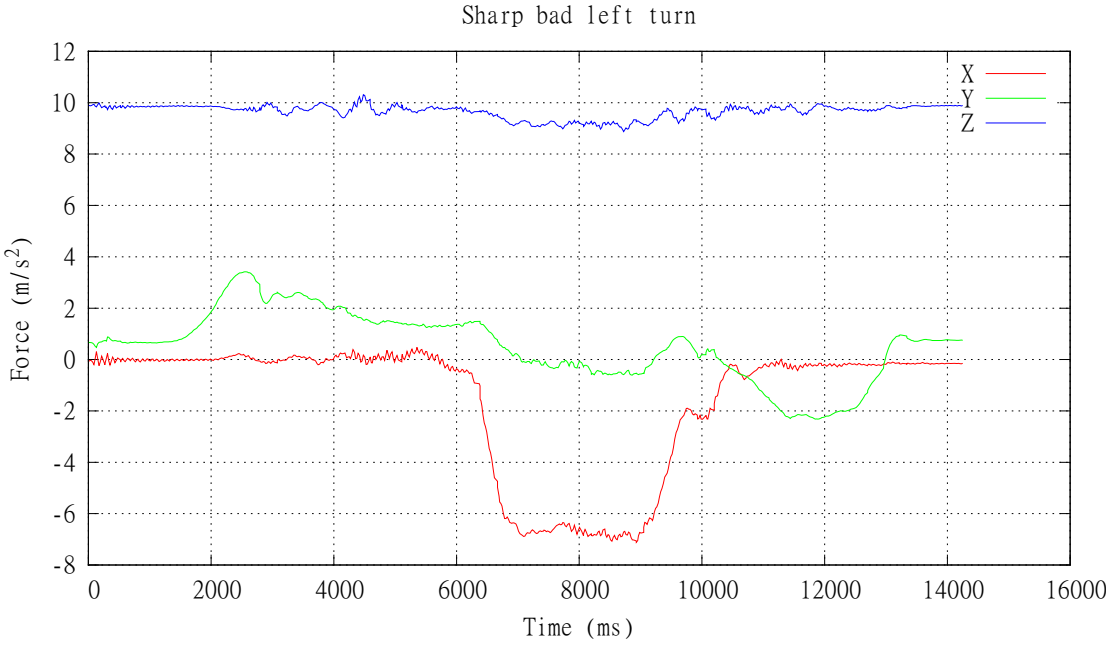
Test 1.5: Emergency brake. (Redone)

This is test is a replica of the first, testing an emergency brake. Excessive noise is remove, using the noise reduction method described in Section 5.3. This test was done on a clear road surface (no ice/snow), which enhanced the the changes in acceleration. At around 7000ms the brake is initiated, leading the values (of the Y axis) to fall from 2m/s^2 to about -6m/s^2, followed by climbing back to normal. All this occurred roughly over 2000ms, which was not comfortable at all. Ergo, rapid acceleration changes is not comfortable.



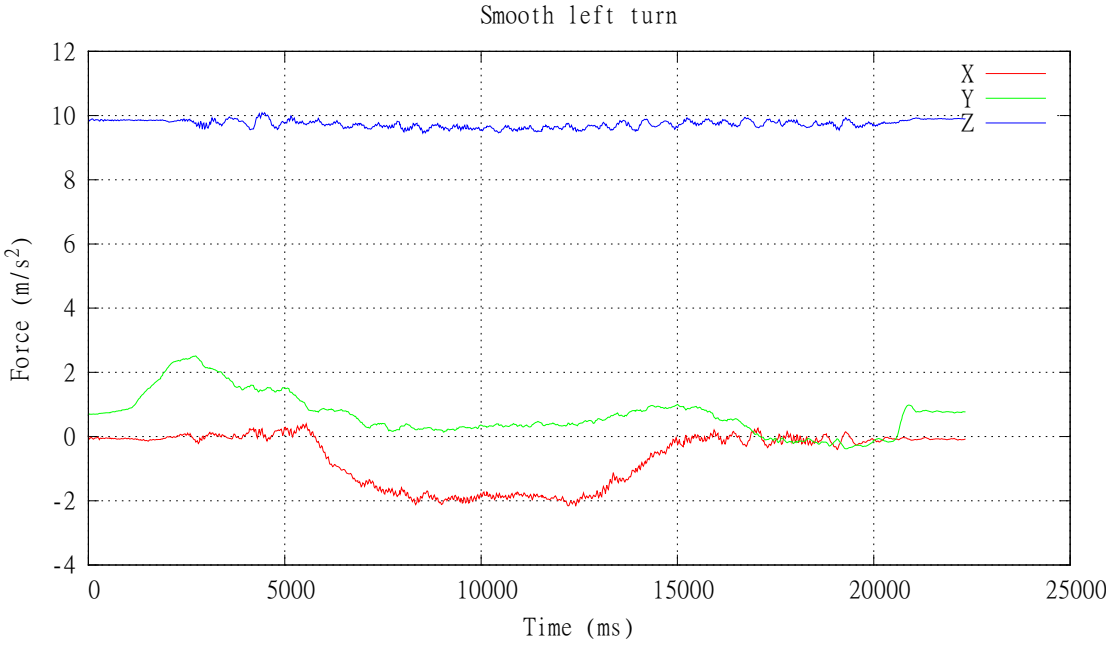
Test 2: Soft comfortable brake.

The graph illustrates the acceleration values of a test when trying to brake comfortably. The vehicle accelerates quick, followed by decelerating slowly. Compared to Test 1, we clearly see that 1) the acceleration changes is much smaller and 2) the period of time when braking is much smaller. Ergo, small acceleration changes over a long time period is comfortable.



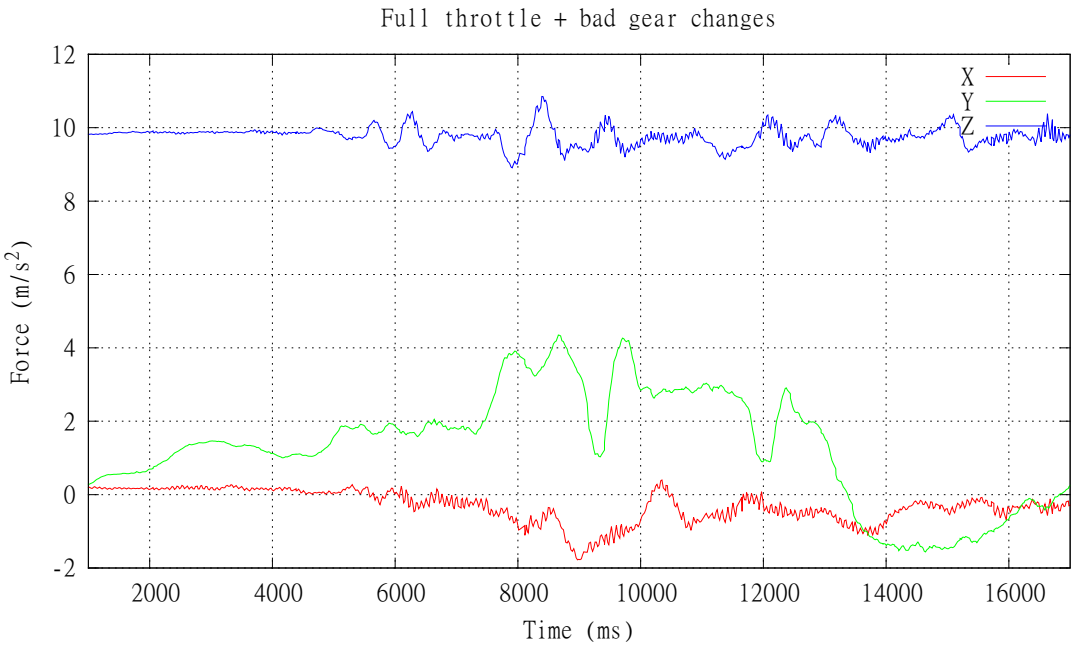
Test 3: Sharp bad left turn.

The graph illustrates the acceleration values of a vehicle in a sharp turn. When the turn is initiated, the acceleration values in the X-axis, falls from $0m/s^2$ to about $-6.5m/s^2$ over a time period of roughly 1000ms. The values stays at around $-6.5m/s^2$ for about 2250ms before returning to $0m/s^2$, which indicates that the turning lasted more than 2250ms. Being exposed to a constant force like this, over a relatively large time period, was not comfortable.



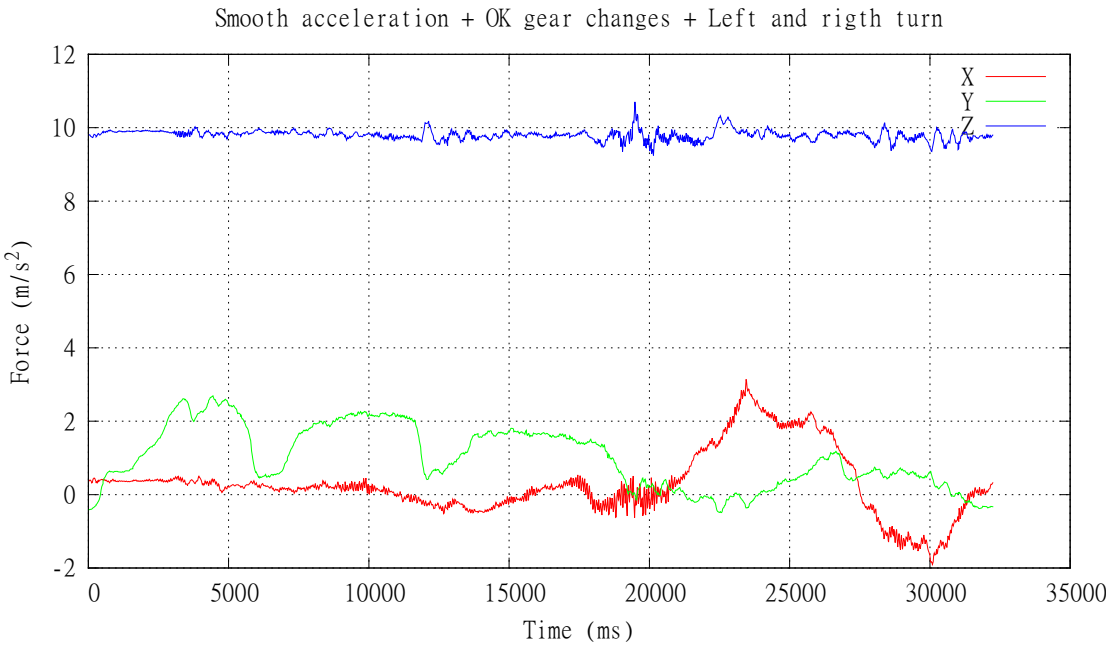
Test 4: Smooth left turn.

The graph illustrates the acceleration values of a vehicle in a smooth/soft turn. The time period (10000ms) is much bigger than Test 3, but the constant force applied in the actual turn is much lower (roughly $2m/s^2$). This test run was not so uncomfortable, in relation to Test 3. Ergo, the level of constant force makes a bigger difference, than the period of time exposed to the force.



Test 5: Full throttle and bad gear changes.

This test run illustrates gear changes executed badly. The first gear change is executed at around 9000ms, which we can see a large bounce (up and down) in the force exposed to the Y-axis. The second gear change executed at around 12000ms, is not as bad as the first, showing a lesser bounce in force. Ergo, big bounces in acceleration over a short period of time (500-1000ms) is bad.



Test 6: Smooth acceleration with OK gear changes, followed by a left and rigth turn.

This test was done to test normal gear changes followed turning both directions. The gear changes is not optimal, but better than the ones executed in Test 5. The right turn (positive X-acceleration value) is much harder (changed roughly $3m/s^2$) than the left turn(changed roughly $2m/s^2$).

H Project agreement

The signed project agreement.

HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Patrick Bours, NISlab

(oppdragsgiver), og

Jørn-André Myrland (080531)
Fredrik Kvitrud (080532)
Fredrik Hortvedt (080530)

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 10/1-11 til 7/6-11.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra HiG å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved HiG og/eller studenter har interesser.

   Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene.

   Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Simon McCallum

Oppdragsgivers kontaktperson (navn): Patrick Bours

Student(er) (signatur): Fredrik Hjelstuen dato 27/1 -2011

Jon-André Myrland dato 27/1-11

Fredrik Kvitvik dato 27/1-11

dato

Oppdragsgiver (signatur): dato 27/1/2011

IMT Dekan/prodekan (signatur): Rune Hjelsvold dato 02.02.2011