Drive-LaB: An Experimental Platform for Usage-Based Car Insurance

Preface

The project builds on the paper *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19]. The paper presents a scoringmodel and data warehouse for usage-based insurance. In the following paper, these components are used as a basis for an experimental system, except where specified otherwise. A description of changes are located in Section 1.1 labeled Prerequisites.

Casper Laustsen
Casper Holst Laustsen

[illegible]
Johan Leth Gregersen

Morten Møller Jakobsen
Morten Møller Jakobsen

Drive-LaB: An Experimental Platform for Usage-Based Car Insurance

Johan Leth Gregersen, Morten Møller Jakobsen and Casper Holst Laustsen*

Abstract

Usage-based insurance (UBI) is currently surfacing both in research and within insurance companies. There are a lack of actual described UBI products, and those that exist are experimental and limited to small customer segments. Insurance companies are showing a clear interest in entering the market, but UBI as a product is complex, and little research exists when it comes to completely implemented products. In this paper, the authors describe the design, implementation and experimentation on Drive-LaB, a fully functional UBI platform. Drive-LaB lets users collect spatio-temporal data with their smartphone. The system uses this data to identify driving style and environmental context, to allow risk assessment associated with car insurance. Drive-LaB is supported by a complex backend system featuring an advanced data warehouse and computational logic to identify driver styles. It also offers an easy-to-use Android application frontend, allowing users to log trips and see detailed statistics on completed trips. Drive-LaB has been used for experiments, collecting more than 13.000 kilometers worth of data in roughly one month. This data has been used to validate the platform and display how the system performs in a realistic setting.

1. INTRODUCTION

Usage-based car insurance (UBI) has been researched actively over the last decade. Insurance companies are showing interest in making UBI a reality, and some have even launched experimental products [3] [27] [28]. Achieving a fully functional UBI product is a complex task involving difficult design choices and numerous technical challenges. Several research papers attempts to address specific concerns such as data quality [25] or privacy [5]. UBI products are still sparse, and to the authors knowledge no one has offered UBI as a countrywide product, anywhere in the world. For usage-based insurance to compete with traditional car insurance, there are many potential problems and solutions, depending on the chosen design. Considering a simple UBI product, insurance companies could measure distance driven by a user, and bill for mileage accordingly. Implementing such a simple UBI system still results in several non-trivial choices. Examples could be:

- Whether to use dedicated devices or rely on user equipment such as smartphones
- Which technology to rely on for accurate positioning, and at what frequency
- How to retrieve and store data logged by individual users
- How to let users keep track of their insurance, understand and verify that they are being billed correctly

This paper presents the entire stack of a fully functional UBI system that anyone can use. The authors attempt to move UBI away from ideas and models and one step closer to a market implementation. Having a live UBI system allows for real-life experiments,

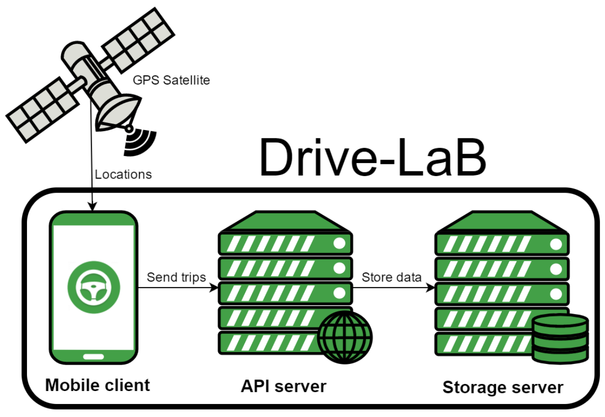


Figure 1: Composition of the system

*The Department of Computer S, Aalborg University and offers unique insight into problematiques associated with different approaches to UBI. In this paper the authors try to answer the following questions:

- How could a complete product look like?
- Is it possible to create a UBI system that supports a fair and understandable metrification of driving styles?
- Are modern smartphones adequate to support UBI?

The remainder of this paper describes the process leading towards answering these questions. In Section 2 the system design is explained. Section 3 describes the implementation of the system. Section 4 describes experiments made possible by releasing the system into a public domain. Finally, the authors provide answers to the problem statement based on the results of the experiments, followed by a conclusion in Section 6.

1.1. Prerequisites

Drive-LaB is based on the paper *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19]. The project features a metric-based scoringmodel for UBI, and focuses on being intuitive and understandable. Furthermore, it features an advanced data warehouse capable of storing all required data for supporting the described scoringmodel. Drive-LaB utilizes both the data warehouse and metric-based scoringmodel, although with certain improvements. The data warehouse implemented for Drive-LaB can be found in Appendices, Figure 17. Most notably, the `SubtTripFact` table has not been implemented in Drive-LaB. Instead, two new tables are introduced, namely `Competition Information` and `CompetingIn`.

The scoringmodel has not been altered, although its flexibility has allowed us to create a more fitting policy for the system. The policy used for experiments will be described in Section 4. Each metric delinquency is divided into 8 different intervals, with different weights for each interval. Each metric is scored by the following algorithm:

$$\left(\frac{\sum_i^n (interval_i * \quad weight_i)}{100}\right) \quad - \quad 1$$

This results in an aggregated weight. Roadtypes and Critical time period are scored linearly, calculated by multiplying the aggregated weight with the amount of delinquencies. Speeding, Accelerations, Brakes and Jerks are all evaluated polynomially. The aggregated weight is fed into a polynomial equation determined by the policy, resulting in a final aggregated weight which is multiplied with the amount of delinquencies.

$$\begin{aligned} &ax^y + bx + c \\ &\qquad where \quad x = AggregatedWeight \end{aligned}$$

The full description of the scoringmodel is described in [19] at Section 5.1.

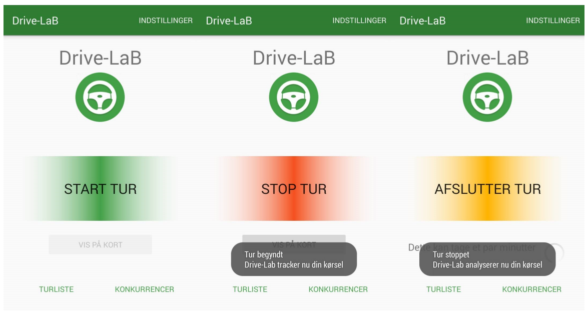
2. DESIGN

This section describes the design of a full-stack UBI system. Drive-LaB is designed to collect, process and store spatio-temporal data from its users. It is a complex system but is designed to be simple to understand and use. Complexity should not be an issue for neither the end user nor the insurance company. A goal of this paper is to answer whether smartphones are suitable devices for UBI. For this purpose, the frontend of Drive-LaB is designed as a mobile application. This choice eliminates the need for a dedicated tracking device, while increasing accessibility for anyone wanting to use Drive-LaB. The design of Drive-LaB as a complete system can be seen in Figure 1. It is composed of three overall components. On the right side of Figure 1 is the storage server. In itself, it contains no logic, and simply acts as storage for the data warehouse. Left of the storage server, is an API server that acts as interface for the frontend, and performs all required operations on incoming data. No data is stored on the API server permanently, but is instead sent to the storage server. Finally, Drive-LaB has a frontend application. It is responsible for location tracking, and visual presentation of the results calculated on the API server. As such the frontend allows users to evaluate trips after driving them. External services such as GPS satellites are used to provide location data for Drive-LaB.

2.1. Frontend

The frontend of Drive-LaB has two responsibilities in the overall system, namely collection and presentation of location data. As the interface for users, it is designed with ease of use and understandability in mind. Given the non-trivial responsibilities, these requirements pose significant design challenges. These are addressed in Sections 2.1.1 and .

2.1.1. Data Collection. Data collection is complicated by requiring user interaction. Without additional hardware, it is not possible to automatically detect the be-



Tur begyndt
Drive-Lab tracker nu din kørsel

Dette kan tage et par minutter

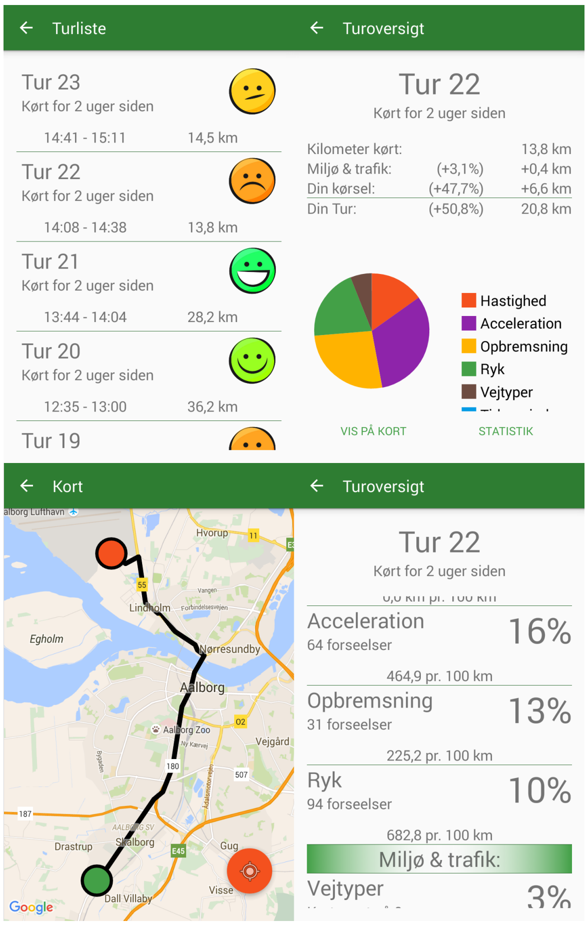
Tur stoppet
Drive-Lab analyserer nu din kørsel

Figure 2: Design of the Start/Stop/Finishing button

ginning or end of a trip. As such, users are required to manually control when their smartphone should start and stop tracking their movement. Upon ending a trip, a sequence of actions takes place. A series of locations has been logged, and is passed on to the API server. To achieve this, data is converted into classes readable by the API server. Furthermore, trips are packaged into JSON objects which can be received by the REST service hosted on the API server described in Section 2.2.1. If the API server is unavailable, trips are cached locally and bundled the next time a trip is ended. This sequence of events does not require user interaction, and is fully automated. Summarizing the process of collecting location data, one can gather three different working states for the application:

- Idle - Ready to track a new trip
- Tracking - Continually logging new positions
- Finishing up - Packaging and sending all logged data

These working states are made visible through a central button in the application, as seen in Figure 2. The progression of working states happens from left to right. ”START TUR” (START TRIP) means the application is ready to track a new trip, and invites the user to do so. ”STOP TUR” (STOP TRIP) means the application is currently tracking, and invites the user to end the trip when finished. Finally the user is presented with a message, ”AFSLUTTER TUR” (FINISHING TRIP), informing the user that the tracking has stopped, but the application is still working. Upon finishing the process of sending the trip to the API server, the application will once again be ready to start a new trip, displaying ”START TUR” for the user. While the logic behind the button is complex, usage is simple and easy to understand. The user only have to define the beginning and end of a trip; the application takes care of the rest.



Tur 23
Kørt for 2 uger siden
14:41 - 15:11 14,5 km

Tur 22
Kørt for 2 uger siden
14:08 - 14:38 13,8 km

Tur 21
Kørt for 2 uger siden
13:44 - 14:04 28,2 km

Tur 20
Kørt for 2 uger siden
12:35 - 13:00 36,2 km

Tur 19

Tur 22
Kørt for 2 uger siden

Kilometer kørt:		13,8 km
Miljø & trafik:	(+3,1%)	+0,4 km
Din kørsel:	(+47,7%)	+6,6 km
Din Tur:	(+50,8%)	20,8 km

🟥 Hastighed
🟪 Acceleration
🟨 Opbremsning
🟩 Ryk
🟫 Vejtyper

Tur 22
Kørt for 2 uger siden

Acceleration 16%
64 forseelser
464,9 pr. 100 km

Opbremsning 13%
31 forseelser
225,2 pr. 100 km

Ryk 10%
94 forseelser
682,8 pr. 100 km

Miljø & trafik:

Vejtyper 3%

Figure 3: Presentation of trips split across 4 screens

2.1.2. Data Presentation. Data presentation is complicated, both by the extensiveness of the data itself, but also the limited screen size of a smartphone. In other words, the application has to fit a lot of information into a small screen. With the goal of having an easily understandable system, the extensive data is simplified by presenting descriptive summaries, rather than raw data. The small screen is however still a challenge, and for this reason the application offers a multilevel description of trips, each presenting a certain level of data. Each level of description is presented on its own screen, and ranges from a broad description down to specific statistics. For an arbitrary trip, these screens can be seen in Figure 3.

On the top left screen, users are first presented with a list their trips ordered by date. For each trip, general statistics are displayed. These are useful mostly for identifying the trip. In key with keeping Drive-LaB simple to use and understand, the user can also see the trip-score summarized as a smiley. Smileys range between green/happy and red/sad. They allow for cursory trip evaluation, without the need to look at more detailed statistics. The top right screen is shown when clicking a trip, and displays a more extensive score summary. The screen uses a pie chart to visualize the relative influence of each metric, allowing the user to identify points of possible improvement. On Figure 3, the bottom screens both show evaluation specifics, letting the user see exactly what was registered during a trip. The left screen places the route on top of a map, showing the user exactly where locations were logged. The right screen shows exact metric counts, allowing the user to see why a metric score turned out the way it did. The different levels of information allow users to navigate only to the desired detail level. As such, some users might be satisfied with seeing the resulting smiley, and will not be forced to look at details. For those who want more detailed results however, these are also made available.

2.1.3. Competitions. Drive-LaB is designed to offer competitions as an additional service, both for the insurance company and end users. Competitions allow the insurance company to collect additional data about users. For users, bringing a competitive element into Drive-LaB can act as incentive to use it and perform to ones best ability. In the Drive-LaB application, users are able to browse active competitions and choose which to participate in. Section 4.5 further describes end user experiments supported by a competition hosted within Drive-LaB. When participating in a competition, the user gains access to live statistics for the duration of the competition. These include current ranking and a leaderboard meant to encourage and motivate users to perform better.

2.2. Backend

The backend of Drive-LaB is required to handle a large amount of incoming data. Whenever a user ends a trip, the logged data is packaged and sent to the backend for processing and storage. The backend is also required to respond to frontend requests for data used for visual presentation, as described in Section 2.1. These requirements are wrapped into a cloud-like architecture, where all data is stored on the Drive-LaB servers. Whenever a user attempts to access data on competitions or earlier trips, the application requests the data from the backend. This limits the amount of local storage required by the application and reduces resources needed for computation on the mobile device.

The backend of Drive-LaB is split across two servers. The less powerful of the two is exposed on the internet, hosting the communication layer, logical layer and data access layer. The other server is hosted only on the same local network as Server 1, with the trip-processing layer and physical storage layer, as seen by the standard UML component diagram in Figure 4. The two servers can be considered available resources, more than a profound design choice. The functionality design of these servers will now be explained, starting with Server 1, which will be called the API server. Server 2 will be called the storage server. No work has been put into security due to prioritization of resources and the commitment to complete the full range of functionality in the system.

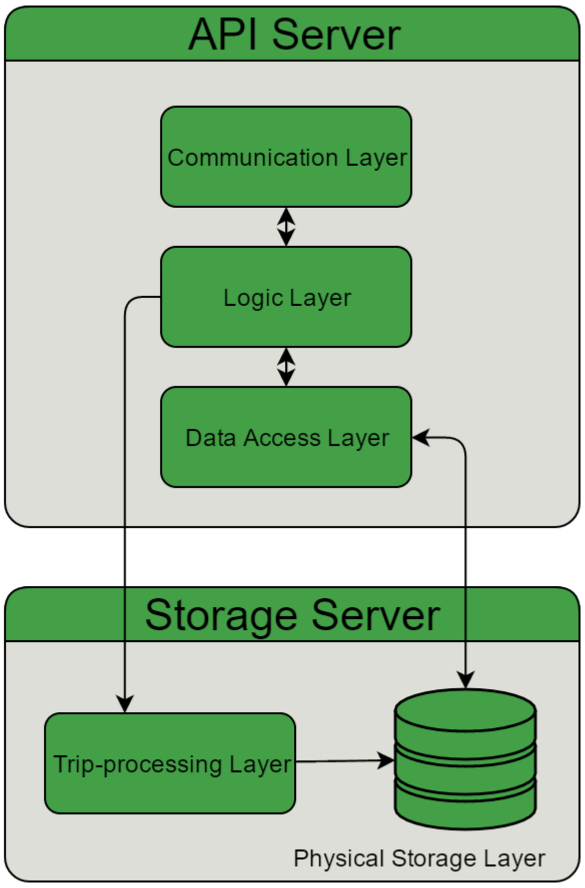


Figure 4: Design of the Drive-LaB backend

2.2.1. API Server. The API server contains three layers: A communication layer, logic layer, and data access layer.

The communication layer provides a uniform interface for both web- and smartphone clients, to communicate with the backend system. The communication layer is designed to be universal and service any device, be it Android, iOS, Windows Phone, Web, etc. This is achieved by hosting a RESTful Web Service on the API server, with a total of four service endpoints.

A service endpoint is an enclosed subset of functionality, placed under a URL-extension to the base address (the IP-address to the server). Each endpoint offers a set of HTTP methods to access functionality in the logic layer. The four service endpoints are named Fact, Trip, Car, and Competition. Each service endpoint offers a structured way of communicating with the functionality in these four categories. As example, the Fact service endpoint offers three HTTP methods. Two of these use the HTTP GET verb, namely `GetFacts`, and `GetFactsForMap`. Both require a `CarId` and a `TripId` as parameters and returns the set of Facts corresponding to the parameters. When using `GetFactsForMap` the set of facts is trimmed to consist only of GPS coordinates and timestamps, because this is the minimal requirement for map display. The third method in the Fact service endpoint is a POST-method. It accepts a stream of trip data sent by a client upon ending a trip. Upon receiving this data, a request is passed on to the logic layer to process the stream of data.

The other three service endpoints function the same way, offering structured communication and utilizing the logical layer for determining which data to either process, or return to the client. The Trip service endpoint contains three HTTP-Get methods for a client to request. These methods return a single trip, trips corresponding to a specific user and a customized list of trips to present in a list, respectively. The Car service endpoint contains two HTTP-Get methods, and one HTTP-Update method: One HTTP-Get method for returning the data stored for a `CarId`, and another to get-or-create a new car in the system. The latter requires an IMEI number as a unique combination of numbers to identify the client. The HTTP-Update method is used whenever a client wants to provide the system with a desired username. The Competition service endpoint has a total of seven HTTP-methods, and will not be described thoroughly. It contains methods to return lists of competitions, and support the ability to sign-up and sign-down from available competitions.

Data exchanged by clients and the communication layer is wrapped in JSON-format. JSON is a lightweight format supported in many environments and languages. Using XML was also considered, but has an unnecessary overhead compared to JSON. GPX (GPS exchange format [6]) was considered for when clients send raw trip data to the backend, but was not chosen due to poor support compared to JSON. It would furthermore require diversification in the formats used, which is considered unfavorable. To consider the use of GPX, a test should prove significant benefits in performance and data-overhead reduction, but such a test was not conducted due to prioritization of resources.

The logic layer is extensive, complex and handles a wide range of functionality in the backend. The logic layer controls the resulting action when a client accesses a HTTP-method in the communication layer. In some cases, the logic behind a HTTP-method is trivial, causing the logic layer to simply request data from the data access layer. The logic layer then parses the raw data into appropriate classes which can be serialized using JSON. It then returns the result in proper format to the client. This serialization is designed to be very simple, because the system makes use of modularized classes. As an example, the `Fact` class only contains three simple properties: `EntryId`, `CarId` and `TripId`. An instance of the `Fact` class contains instances of six additional classes called Spatial, Temporal, Measure, Flag, Segment, and Quality. Each of these classes are serializable on their own, because they implement their own `DataContract`. A `DataContract` defines how to serialize a C# object into JSON or XML format. Often a client only require a single module of the otherwise complex fact object. Having modular classes then allows for serializing simpler objects, which are to be sent using HTTP verbs.

The logical layer forwards the task of processing of new trips to the trip-processing layer. This layer is hosted on the more powerful of the two servers. This utilizes the computation power available on the two servers optimally, because trip-processing is the main consumer of computation power. Trip-processing is described in Section 2.2.2.

The data access layer (DAL) handles selecting, inserting, updating and removing data on the storage server. This layer has to work well with the choice of PostgreSQL as DBMS. A description of the physical data layer can be found in Section 2.2.2. The DAL retrieves data from the physical data layer and returns references to C# objects in the logic layer. This allows the DAL to provide simplified access to raw data. Queries in the DAL service can return both complete rows of data and customized selections. Conversion from raw data to class objects often includes extensive null checking in the logic layer, to eliminate null-reference exceptions, because the modularity of classes is not perfect.

2.2.2. Storage Server. The storage server contains two layers: Trip-processing and physical storage.

The trip-processing layer handles the processing of new trips when it receives a stream of raw GPS data. The purpose of this layer is immensely specific, but the functionality it encapsulates is comprehensive. This layer is intentionally removed from the logic layer and placed on the storage server to optimize the use of computational resources. It also ensures the fastest possible retrieval and updating of data in the database, as the trip-processing layer can utilize localhost querying, removing any network communication overhead. The trip-processing layer uses the scoringmodel described in Section 1.1 and other components referenced in *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19]. The layer completes seven steps to process a new trip, listed below:

Steps required to process a new trip

- Deserialize from raw GPS data to C# objects
- Insert a new trip into the TripFact table of the database, and obtain the automatically assigned `TripId`
- Insert the raw GPS coordinates and timestamp into the GPSFact table, setting the relevant `CarId` and `TripId` for each row
- Request map-matching from third party service, using the raw GPS coordinates [7]. This requires the parsing of GPS coordinates into CSV-format, which becomes part of the request sent to the service. The service returns a set of road segments used on this trip, and the collection of map-matched GPS coordinates. Entries in the GPSFact table is then updated with the map-matched GPS coordinates, road segment ids and speed limits (if available).
- Compute measures and flags for the entire trip in the GPSFact table. Measures are attributes like speed, acceleration, etc. Flags provides a true/false value to indicate whether the driver is speeding, accelerating, etc. If duplicate timestamps are found in the collection of GPS points for a trip, these are pruned. Using a 1 second resolution makes such points relatively common, but also useless. It is impossible to compute measures with a division of 0 seconds as time passed since the previous point. The updated GPSFact entries are then updated in the GPSFact table with the computed measures and flags.
- Compute the attributes in the TripFact entry: Length of trip, duration, optimal score, tripscore and the count of each delinquency, etc. The attributes in the TripFact entry are then updated in the database. The entire list of attributes can be seen in Figure 17 in Appendices.
- Check whether the car is enrolled in a competition, and if so, check whether the trip is valid for use in the competition. If it is, update the CompetingIn table, with the corresponding score and increased number of attempts.

The physical storage layer is where the database is hosted. The database is a modified version of the data-warehouse schema presented in *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19], hosted through the open source DBMS, PostgreSQL [26]. The modified data warehouse schema can be seen in Figure 17 in Appendices.

3. IMPLEMENTATION

In the following section the implementation of Drive-LaB is described.

The backend of Drive-LaB covers the 5 layers described in Section 2.2. It spans over 8.042 lines of code, written in C# in Visual Studio. The frontend application is implemented as an Android application. It spans over 19.192 lines of code, written in Java and XML, in Android Studio.

The system makes use of 2 different servers. The first server, being the API server, is a single core server with a Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz processing unit, and 6GB RAM. The second server is an 8-core server with an AMD Opteron(tm) Processor 6376 processing unit, and 16GB RAM.

3.1. Frontend

A goal for Drive-LaB is to be easily accessible. To fulfill this goal, Drive-LaB is developed for Android. As of 2015 Q2, over 80% of shipped smartphones use the Android OS [30]. Choosing Android as platform has allowed for the release of Drive-LaB on Google Play [12], making it accessible for the vast majority of smartphone users. Furthermore, the implementation of Drive-LaB supports Android 4.0 – 6.0.1, making it compatible with over 97% of current Android devices [4]. As mentioned in Section 2.1, the application has two responsibilities; data collection and data presentation. These responsibilities are mirrored in the structure of the application, seen in Figure 5. The application consists of two major components. One is the user interface, consisting of 10 different Android `Activity` [9]. The interface presents data for the user, and allows for interaction with `Location Service`, the second component. `Location Service` is an Android `Service` [17]. It runs separately from the rest of the application, in its own process. It is however entirely controlled by the user interface as seen in Section 3.1.1. The `Location Service` is responsible for all location logging. Running the service separately allows for

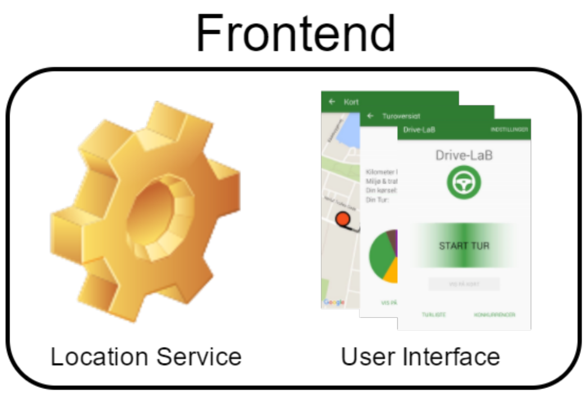


Figure 5: The frontend consists of two components, running separately from each other

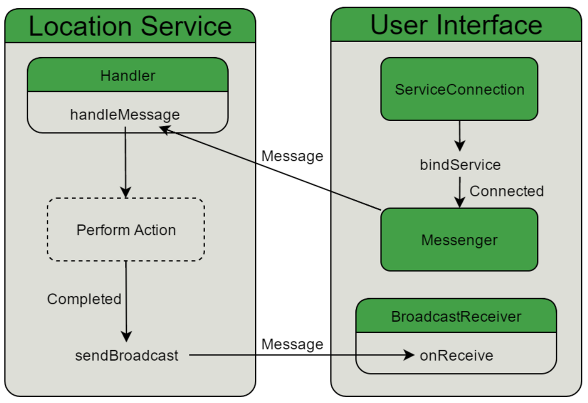


Figure 6: Two-way communication path between Location Service and User Interface

a lifecycle independent of any `Activity` bound to it. In effect, the Android device can still be used normally. The application can be closed, and the screen turned off, saving battery. This will not affect the `Location Service` in any way.

3.1.1. Service Communication. The downside of having `Location Service` running in a process separate from the rest of the application, is that communication becomes non-trivial. With the chosen setup, there is no support for synchronous communication, method invocation, or even two-way communication. The alternative method of communication is illustrated in Figure 6. `Location Service` and `User Interface` represents a running instance of a `Service` and an arbitrary `Activity` respectively. For these components to communicate, an implementation of the interface `ServiceConnection` is required [18]. The `ServiceConnection` is bound to the `Location Service` using `bindService` [10]. Upon a successful connection to the `Location Service`, the `ServiceConnection` instantiates a `Messenger` [15] which is able to send messages asynchronously. Looking at the `Location Service` in Figure 6, incoming messages are first caught by a `Handler` [14] class. By extending this class and overriding its `handleMessage` method, the `Location Service` is able to determine a course of action, depending on the message received.

Often, the `Location Service` is required, not only to perform an action, but also respond to incoming messages. The `Location Service` is however unable to reference a calling `Activity`. Instead, the `Location Service` utilizes `sendBroadcast` [16] which issues a message globally on the device. Returning to the `User Interface` side of Figure 6, the receiving `Activity` can then listen for the message using a `BroadcastReceiver` [11] and act accordingly to the content of the broadcasted message.

3.1.2. Location Logging. `Location Service` is responsible for the continuous retrieval of location updates. Retrieving location updates through Android is done using the Google Play services location APIs, specifically the `FusedLocationProviderApi` [13]. This API is able to automatically choose the best location provider, maximizing the possible precision and availability of location updates. Locations can therefore be based on both GPS, Cell-ID, or Wi-Fi. To achieve the desired quality and frequency of locations, these settings are used when requesting locations through the `FusedLocationProviderApi`:

- Desired interval: 1000ms
- Fastest interval: 1000ms
- Priority: High Accuracy

These settings enables Drive-LaB to receive locations exactly once every second, whenever possible. Locations are furthermore pinpointed as exact as possible, regardless of battery consumption. This will usually result in locations based on GPS positioning, as this is generally the more accurate option.

3.2. Backend implementation

Servers for the Drive-LaB backend are virtual machines running Ubuntu, a GNU/Linux operating system, and does not naturally run any Windows executables (.exe). Mono is an open source implementation of .NET, capable of running C# software on Ubuntu [22]. Because Mono does not implement the complete feature set of .NET, using Mono causes some issues when porting from initial use of Visual Studio to Mono. These issues are, among others, stated in the following implementation description.

3.2.1. API Server. As portrayed in Figure 4 the API server consists of 3 layers described in this section.

The communication layer is fully implemented in C# as a RESTful Web Service API, using the built-in .NET `ServiceModel` library. It is implemented following the design described in Section 2.2.1. The communication layer includes thorough error-handling along with an error-reporting system. It is critical that the communication layer does not crash, because the access to the Drive-LaB backend will crash with it. The error-handling ensures that corrupted data being processed in the logic layer, does not cast exceptions back to the communication layer. Also, by using the .NET RESTful library, a series of error-handling tasks is conducted automatically. As example, if a client targets a non-existent service endpoint, or if a client use a wrong HTTP verb, the API will return a corresponding HTTP error code.

The logic layer is implemented in C# using regular OOP-style programming. It contains many classes and methods to handle the variety of functionality handled by this layer. A majority of this functionality is to create appropriate C# objects, either based on JSON data received from the communication layer, or data received in `DataRow` format from the DAL. DataRow is a .NET specific data type, designed to hold a row of data received from a database, which is what the DAL returns to the logic layer.

The logic layer uses Json.NET, a popular JSON framework for .NET [23]. Using a third party framework to support JSON serialization and deserialization is necessary, because Mono does not contain Visual Studio libraries to support this task. Another library that is not implemented in Mono is `Device.Location`, which offers the type `GeoCoordinate` in C#. It is used to store spatio-temporal data, and offers functionality like computing distance between two coordinates. A third party library called GeoCoordinatePortable offers this functionality while also being Mono-compatible [8]. Therefore, this library is used as substitute.

Last is the DAL. This layer is implemented using a combination of SQL and C#. It makes use of `Npgsql` [24], a .NET data provider for PostgreSQL, which makes it very easy to write SQL statements and C# code in the same IDE.

3.2.2. Storage Server. In the implemented system, it was found to be simpler to implement the trip-processing layer as part of the logic layer. This does opposes the design decision to optimize the computational resources offered by the more powerful server. Monitoring the ongoing load on the API server, however shows that computation power is sufficient with a small set of users. The solution will however not scale well, and with more users this will have to be reworked to maximize the performance of the system.

The trip-processing layer contains two extensive computational schemes named `GPSFactUpdater` and `TripFactUpdater`. The former computes all measures and flags between every GPS coordinate logged during the trip. This is the foundation for the entries inserted in the `TripFact` table and therefore has to be accurate. The modular design of classes is valuable in the context of processing trips, because appropriate objects can be chosen and forwarded to the `MeasureCalculator`. The `MeasureCalculator` contains mathematical formulas for calculating measures, for example how to compute speed. Only the appropriate sub classes are sent as parameters, and the complete object is not thrown around between formulas.

`TripFactUpdater` computes the required attributes in the `TripFact`, which is statistical groupings of the information stored in the `GPSFact` table. It uses these statistical attributes to analyze driver performance, compute optimal- and actual tripscores. The scoringmodel, used to compute the actual tripscore, can be seen in Section 1.1.

4. EXPERIMENTS

The following section contains a collection of experiments. Section 4.1 is an introductory description of the data collection used as basis for the experiments, followed by an explanation of the policy used. Section 4.2 is a description of a system experiment, testing whether smartphones are suitable GPS devices for a usage-based insurance. Following is a test of the metrics used in Drive-LaB, to test whether they correlate with each other. It is described in Section 4.3. This is done by calculating the Pearson correlation between the metric scores, and analyzing the results. Section 4.4 contains an experiment on driver profiling. With the experiments follows a discussion of whether the metrics chosen are sufficient. Optimally, it should be possible to create a characterizing driver profile and differentiate among the drivers with risk assessment in mind. Lastly, Section 4.5 describes a user experiment involving 10 drivers who used Drive-LaB consecutively for a test period of roughly one month. This experiment was used to test the entire system, and the concepts of usage-based insurance from a business perspective with real users.

4.1. Data Collection

Releasing Drive-LaB into the public has allowed for the collection of a sizable dataset. For this report, 51 people participated, contributing a total of 581 trips spanning 13.075 kilometers. The trips are all logged between May 1st, 2016 and June 11th, 2016. All installations of Drive-LaB uses the same setup for logging locations, ensuring a uniform premise across all contributed trips. As mentioned in Section 3.1.2, the requested sample rate is set at 1Hz, but conditions such as hardware limitations and poor signal can block this from being possible. As such, the collected data varies in sample rate. The sample rate can be computed by comparing the total number of entries in the `GPSFact` table with total number of seconds driven among all trips.

$$\frac{566.659\ entries}{898.290\ seconds} = 0.63Hz \qquad (1)$$

The total average of the dataset is 0.63Hz, which is still high-frequency, but lower than the desired 1Hz.

4.1.1. Data. Trips logged through Android devices contain a set of latitudes, longitudes and timestamps with an optimal granularity of 1Hz. The Drive-LaB application does not store data permanently and therefore only holds this data in memory. As a trip is sent, all data is received by the API server and processed as described in Section 2.2.1. When the data reach the physical storage layer on the storage server, all data is stored in tables corresponding to the data warehouse schema described in Figure 17 in Appendices.

4.1.2. Policy. The trips have been evaluated based on a policy throughout the test period. The policy is described in Tables 1, 2, 3, 4 and 5. The base values for the delinquencies are described in Table 6. The most notable change is that the polynomial scoring mechanism presented in *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19] has been omitted, as it is uncertain how the equation directly affects the scoring of the system. It should be revised when more information about the effect is present.

4.2. System Experiments

An important aspect of the Drive-LaB experiments, is to validate if there is an impact on tripscores caused by the diversity of users' smartphones. In the context of usage-based insurance, users should be treated

Roadtype	Weight
Motorway	0.8
Trunk	0.9
Primary	0.95
Secondary	1.05
Tertiary	1.1
Unclassified	1.1
Residential	1.2
Service	1.2

Table 1: Roadtypes with weights

Active days	Start	End	Weight
Monday - Friday	07:00:00	09:00:00	1.16
Monday - Friday	15:00:00	17:00:00	1.12
Saturday - Sunday	09:00:00	13:00:00	1.02
Saturday - Sunday	20:00:00	23:59:59	1.12
Saturday - Sunday	00:00:00	00:04:00	1.325

Table 2: Critical time intervals with weights

Interval (%)	Weight
[0, 10[	1.3
[10, 20[	1.4
[20, 30[	1.5
[30, 40[	1.6
[40, 50[	1.7
[50, 60[	1.8
[60, 70[	1.9
[70, ∞]	2.0

Table 3: Speeding intervals with weights

Interval (m/s)	Weight
[6, 7[	1.05
[7, 8[	1.10
[8, 9[	1.175
[9, 10[	1.275
[10, 11[	1.40
[11, 12[	1.55
[12, 13[	1.725
[13, ∞]	2.0

Table 4: Acceleration with weights

Interval (m/s)	Brake Weight
[8, 9[	1.05
[9, 10[	1.10
[10, 11[	1.175
[11, 12[	1.275
[12, 13[	1.40
[13, 14[	1.55
[14, 15[	1.725
[15, ∞]	2.0

Table 5: Jerks and brakes with weights

Action	Base weight
Acceleration	40
Brake	65
Jerk	15

Table 6: Base weights for accelerations, brakes and jerks

equally, which relies entirely upon their smartphone and its GPS antenna. Under optimal conditions, two smartphones logging the same trip, should report identical tripscores. Such an analysis can be quite extensive, involving an entire market of smartphones and different versions of GPS antennas. Instead, a small scale test was performed. The test contained a selection of different smartphones to indicate whether Drive-LaB is vulnerable to GPS inaccuracy.

The test will be referred to as an applicability test. An applicability test will confirm whether or not smartphones are capable of scoring trips equally. It was conducted by setting up five different smartphones and two high quality GPS trackers [29] in the same car, and record the trip concurrently with all devices. For the system to be applicable for usage-based insurance, the smartphones needs to report identical tripscores, and the routes should to be near-identical to those recorded by the high quality GPS trackers.

With the seven devices, four trips were completed driving around in northern Jutland, in the vicinity of Aalborg. Raw data from the trips can be seen in Tables 10, 11, 12 and 13 in Appendices. A summary of the tripscores for each device, can be seen in Table 7.

When examining trip 1 from Table 7 which has a length of approximately 36.200 meters, two smartphones and one high quality GPS scores between 37.000-40.000. This corresponds to a good score. But the remaining four devices disagree with varying severity. The worst is the Huawei Y330, which scores 81.819. The Huawei Y330 however only logs 23% of the average GPS coordinates compared to the other devices. The Huawei continues this trend throughout the

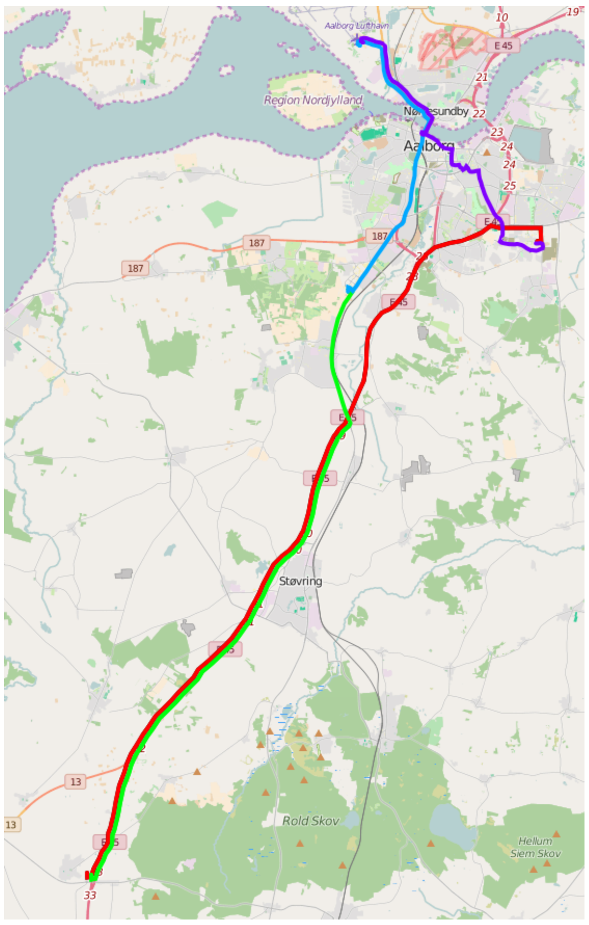


Figure 7: Routes traveled in the area of Aalborg during the experiment

test, and seems unfit for use in usage-based insurance. Another alarming result in trip 1 is the degree to which the two high quality GPS devices disagrees. One scores 37.910 while the other scores 69.956 - a 84% increase.

Trip 2 in Table 7 has a length of approximately 28.200 meters. On this trip, one of the high quality GPS devices failed to log the entire trip. For the remaining devices, two range from 27.750-28.750 in tripscore. The noteworthy result compared to trip 1 is, that the devices with a low tripscore are not the same as those in trip 1.

The same pattern occur with trip 3 from Table 7, which has a length of approximately 13.400 meters. This time, four devices gets a score ranging from 16.500-24.050, which are somewhat similar. The last three devices got a much higher tripscore, the highest being 85.139, approximately 535% above the triplength, and it was logged by one of the high quality GPS devices. To highlight the disagreement between the two high quality GPS devices, the other device got a score

Table 7: The tripscores from all seven recording devices, on all four trips used, during the first test

	Trip 1	Trip 2	Trip 3	Trip 4
OnePlus One	50.191	58.922	42.751	23.228
Samsung Galaxy S5	40.092	56.781	24.026	27.530
HTC One Mini 2	75.063	28.734	21.012	19.203
Huawei Y330	81.819	128.056	50.622	13.082
Samsung Galaxy S4	37.010	27.762	16.927	18.825
BT-Q1300ST (#1)	37.910	25.373	20.981	23.917
BT-Q1300ST (#2)	69.956	72.785	85.139	27.074

of 20.981.

The pattern is broken with trip 4 from Table 7, where all devices range between 13.000-27.550. While scores are still diverse, it is significantly better compared it to trip 3. For this trip, the worst device scored approximately 91% above the length from this trip, compared to 535% in trip 3. The high quality GPS devices also score similarly on trip 4 with a result of 23.917 and 27.075 respectively.

When analyzing the results from this test, the immediate thought may be to throw away the smartphone as usage-based insurance component. Concern about accuracy, integrity, availability and continuity of service in standalone GPS receivers is also raised in other articles [20] [21] [25] [31] [32]. But two other factors may have influenced the results in Table 7. The first factor is GPS interference, in which case the test setup could have changed the results. For the experiment, all GPS devices were in close proximity of each other. All devices were placed in a fabric container and placed in the front of the car near the windshield. This could affect the GPS receivers by interfering with each other [1] [2]. This position was decided upon, because a common reference point was valued in the applicability test. One article provides concrete results in terms of coverage from smartphone GPS receivers, and during a 1 hour and 15 minutes trip, 6 hard brakes were detected with an OBD device inside the car. A total of seven smartphones was brought on this trip, and they had a coverage in the interval of 60% to 99.7% when detecting these brakes [25]. This states that smartphones are indeed vulnerable to inaccuracy. Coverage means the degree to which the smartphones align with the control-unit, in this case the OBD. The test also included outliers, false positives, and indeterminable which causes the percentages to be skewed.

The second factor is the use of TrackMatching, a third party software for map-matching spatio-temporal trajectories [7].. TrackMatch attempts to map-match a series of points to the OpenStreetMap road network, and output the entire route by segments and map-adjusted GPS points. The algorithm TrackMatch uses to map-match coordinates are however unknown. No module was implemented to oversee this readjusting, so this influence cannot be changed.

It was decided to repeat the applicability test and eliminate the GPS interference as much as possible. For the second test, the GPS devices were arranged in the car with as much distance as possible between them. The possible margin of error by using a separated reference point was disregarded. The Huawei Y330 was also omitted.

The test results shown in Table 8 once again shows a considerable difference when comparing the two high quality GPS devices. But the difference is decreased substantially compared to the first test, which signifies that some external influence may have been affecting the devices. Looking at the raw data in Tables 10 through 17, an observation is that their behavior is consistent. When looking at the amount of accelerations, brakes and jerks, one device consistently counts more than others. BT-#1 counted a total of 1529 of these events, whereas BT-#2 counted a total of 3894. That is 191.13 events per trip on average for BT-#1, and 486.75 events per trip on average for BT-#2. This generalization of more events registered by BT-#2 are present in both tests. This signifies that it may not be able to conclude any comparison between these two devices, because they deliver such different results.

The results from Samsung Galaxy S4, Samsung Galaxy S5 and BT-#1 from the second test, actually compares well to the results they provided in the first test. There is an influence in the driving style, but both tests was performed by the same driver. The driver attempted to follow his personal driving style during both tests. Additionally, traffic may also influence the results in the tests, but the tests was performed in a similar time-period, both on a weekday.

The remaining devices disagrees with their results from the first test to the second test, for some of the trips. When looking at the overall result from both tests, the system is not currently applicable for usage-based insurance. This is due to the diversity in tripscores for similar trips, making it unfair for policyhold-

Table 8: The tripscores from all six recording devices, on all four trips used, during the second test

	Trip 1	Trip 2	Trip 3	Trip 4
OnePlus One	64.511	31.075	17.103	18.223
Samsung Galaxy S5	46.668	48.169	19.010	22.779
HTC One Mini 2	54.564	39.439	27.674	29.767
Samsung Galaxy S4	37.475	29.242	16.672	18.094
BT-Q1300ST (#1)	37.800	30.397	26.440	25.064
BT-Q1300ST (#2)	41.260	37.029	36.531	45.327

ers. While several articles have raised concern about the GPS receivers, some also suggested ways of supporting the raw data with model-based signal processing and an outlier rejection scheme [20] [25] [32]. This would make the trajectories more stringent, and possibly eliminate falsely positive delinquencies caused by a jumpy GPS coordinate. It is considered a good next step to design and implement such a scheme into the Drive-LaB system. This scheme could potentially make each GPS receiver more consistent. If it could be achieved that every GPS device produces reproducible results, a calibration mechanism could eliminate the diversity caused by different GPS devices. This would ultimately cleanse the instability about accuracy, integrity, availability and continuity of service from standalone GPS receivers, and make the system robust enough to apply entirely to usage-based insurance.

4.3. Pearson Correlation

Drive-LaB valuates trips based on the scoring-model described in Section 1.1. The scoringmodel is based on six metrics; roadtypes, critical time periods, speeding, accelerations, brakes and jerks, as described in *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19]. These six metrics cover a big part of the drivers performance during a trip, but it is essential to verify the importance of each individual metric when identifying driver style. Such an experiment can be conducted by computing a Pearson correlation matrix, in order to ensure that no metrics are directly correlating, meaning one of the metrics is negligible. This experiment is run in coherence with the chosen policy, as described in Section 4.1.2. The Pearson correlation is calculated on the scores of the metrics on the given trips.

The matrix of Pearson correlations between the metrics is shown in Table 9. The most notable result is the multicollinearity between accelerations and brakes, which seems rather odd as they are diametrical opposites and can per definition not exist at the same time. Another highly correlating metric is jerks, having a correlation of 0.828 with both brakes and accelerations.

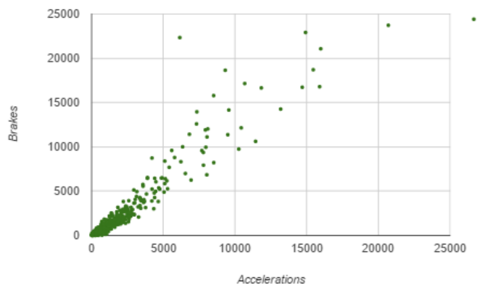


Figure 8: A scatterplot of the correlation between acceleration- and brake score

Looking closer at the multicollinearity between accelerations and brakes, there are a couple of different factors which affect the result. As mentioned, accelerating and braking are diametrically opposite, but they are both dependent on the speed of the vehicle (as they are calculated through the change in speed). Given a trip starts and ends at the same speed(0 km/t), the accumulated acceleration must be equal to the accumulated brake.

Figure 8 clearly illustrates the correlation between accelerations and brakes. Another reason why these metrics correlate can be the thresholds used in the policy to calculate the scores. As earlier mentioned, the threshold for brakes are 8 m/s^2 whereas accelerations are counted from 6 m/s^2. If there were no thresholds, and the delinquencies were scored the same, the correlation would be 1.0 as it only depended on the speed of the vehicle. Jerks was the metric that met the highest level of skepticism when the scoringmodel was designed, and the correlation shows that it was not completely unwarranted. It does show a lot of correlation with both brakes and accelerations. Figure 9 shows the correlation between accelerations and jerks. Jerks are, as mentioned, calculated as m/s^3, and a driver with many accelerations are almost bound to have a lot of jerks as it is hard to keep a constant acceleration.

Table 9: This is the Pearson correlation matrix between the metrics

	Roadtypes	Critical Time Periods	Speeding	Accelerations	Brakes	Jerks
Roadtypes	1					
Critical Time Periods	-0.250	1				
Speeding	-0,546	0.156	1			
Accelerations	-0.341	0.460	0.196	1		
Brakes	-0,348	0.428	0.195	0.971	1	
Jerks	-0,241	0.313	0.144	0.828	0.828	1

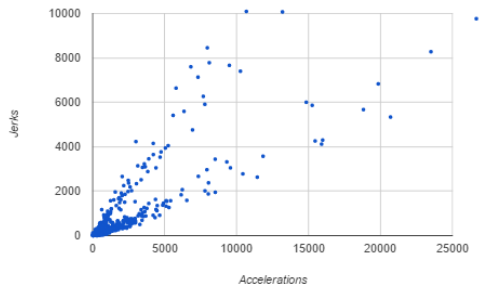


Figure 9: The correlation between acceleration- and jerkscore

4.4. Driver Profiling

Creating driver profiles is one of the strengths in Drive-LaB. With a descriptive set of metrics it is possible to differentiate between drivers, and create fairly accurate driver profiles without compromising the users privacy. The concept of a driver profile is relevant for this project due to the direct connection to the insurance industry. Naturally, it is possible to evaluate the drivers insurance costs more precisely, if the given driver profile is accurate. However driver profiles is demanded to be accurate and portray the full picture when you are dealing with paying customers [25] as they need an incentive to use the product. In the remainder of this section, two random driver profiles will be reviewed and used to illustrate the capabilities in Drive-LaB to differentiate between driving styles.

Score Percentages are a great way to differentiate between drivers. The two drivers will be referenced to as Driver 1 with an average tripscore percentage of 65,08%, and Driver 2 with an average tripscore percentage of 39,07%.

Beside the obvious difference in percentages, looking at where the drivers generate their scores, show clear differences. Figure 10 and Figure 11 shows a comparison between the two drivers, portrayed as a bar chart and a pie chart with the distribution of the tripscore based on metrics, respectively. Looking at Driver 1, a lot of the added score actually comes from accelerations at roughly 21%, brakes at roughly 28% and to some extent jerks at roughly 11%. It is also worth mentioning that roadtypes actually scored negative on average. Looking at the pie chart in Figure 11, brakes are easily recognizable as the biggest contributor to a higher score.

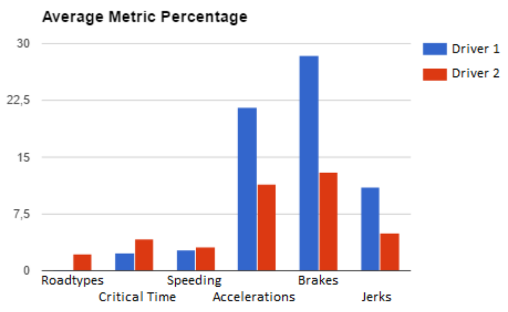


Figure 10: Bar charts of the distribution of tripscore percentage by metrics for Driver 1 and Driver 2

Driver 2 has quite a different distribution than Driver 1, aside from a lower tripscore percentage in general. It is clear that accelerations and brakes heavily influence the score in the system, however this driver has a significantly lower percentage in both of the metrics in the tripscore. This is noticeable in the pie chart in Figure 11 which shows far less disparity between the metrics than Driver 1.

Normalized Metrics are the average metrics on a certain distance driven. For easy comparison the distance chosen is 1.000 meters. Looking at Driver 1 in Figure 12, Driver 1 has 6.84 points with jerks flagged given the chosen distance. Comparing Driver 2 to Driver 1, the former almost halves the amount of accelerations, brakes and jerks per 1.000 meters. The only metric Driver 2 has more of, given the chosen distance, is speeding.

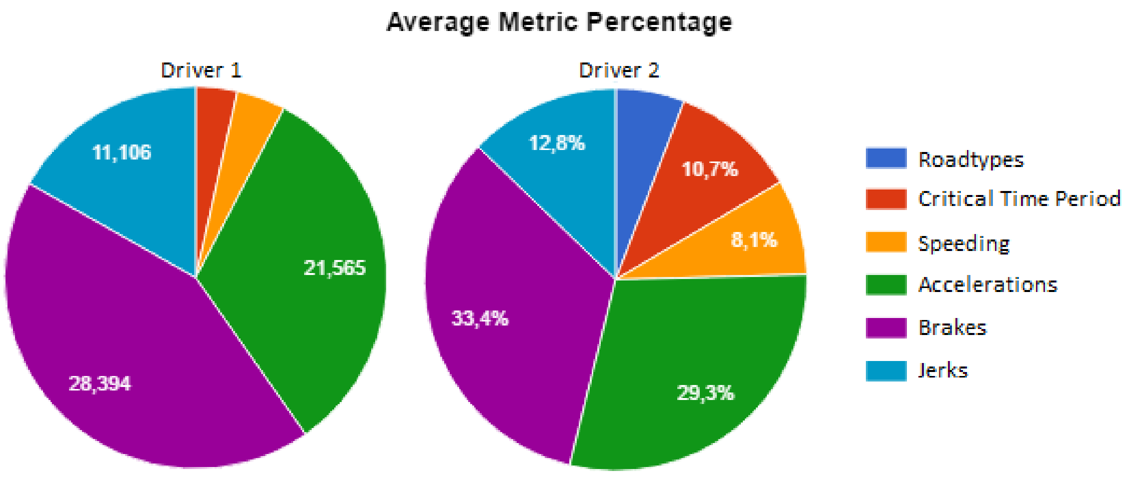


Figure 11: Pie charts showing the distribution of tripscore percentage by metrics for Driver 1 and Driver 2

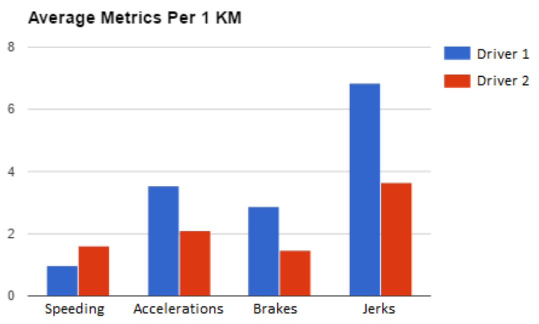


Figure 12: A bar chart of the metrics per 1.000 meters for Driver 1

Severity of Delinquencies is one of the big tells when differentiating between drivers. It is noticeable when looking at Figure 13, which represents Driver 1, there is a slight decline with a spike in the last interval. There might be several reasons as to why the last interval spike but the primary reason is that the interval is everything above a threshold, thus a much larger interval than the previous. Comparing Driver 1 to Driver 2, shown in Figure 14, there is quite a different distribution.

It is proven, that it is possible to distinguish between drivers, and even more important, it is possible to create driver profiles. Given an arbitrary trip it would be possible to draw similarities between the trip and the driver profiles. From a usage-based insurance point of view, it would be possible to assess the risk of a given driver. As an example, a driver with a higher amount of braking delinquencies, all represented as brakes with a hard degree, might have a higher risk of crashing and get a more expensive insurance claim.

4.5. User Experiments

The entire Drive-LaB system has been tested through an end user experiment involving a subset of users. The experiment includes 10 drivers, who have been instructed to use the system for all their vehicular trips. To get started, all users received a two-page guide for setting up Drive-LaB on their smartphone. This guide can be seen in Appendices. The experiment ran from the 1st of May till the 3rd of June. As a motivator for using the system thoroughly, a competition was hosted, offering a prize for the user with the lowest average score percentage. The competition could be followed live through the application, allowing the drivers to monitor current results and compare them with other participants. At the end of the competition the 10 users completed 345 trips spanning 8.191 kilometers.

There are several points to asses in this experiment, and one of them is to test the system in its entirety, in a setting true to the environment where the system will eventually be released. This test setup is as true to the real setting as possible. The frontend application was deployed on Google Play, where participants could downloaded from. Throughout the experiment, participants had access to a leaderboard with average scores for all participants. It would be desirable to receive user inputs on the system, to learn about perceived ease of use and possible improvements. As of the publishing of this paper, that feedback is still on its way.

One of the more interesting things to investigate through this experiment, is whether participants improve their score percentages over time. Figure 15

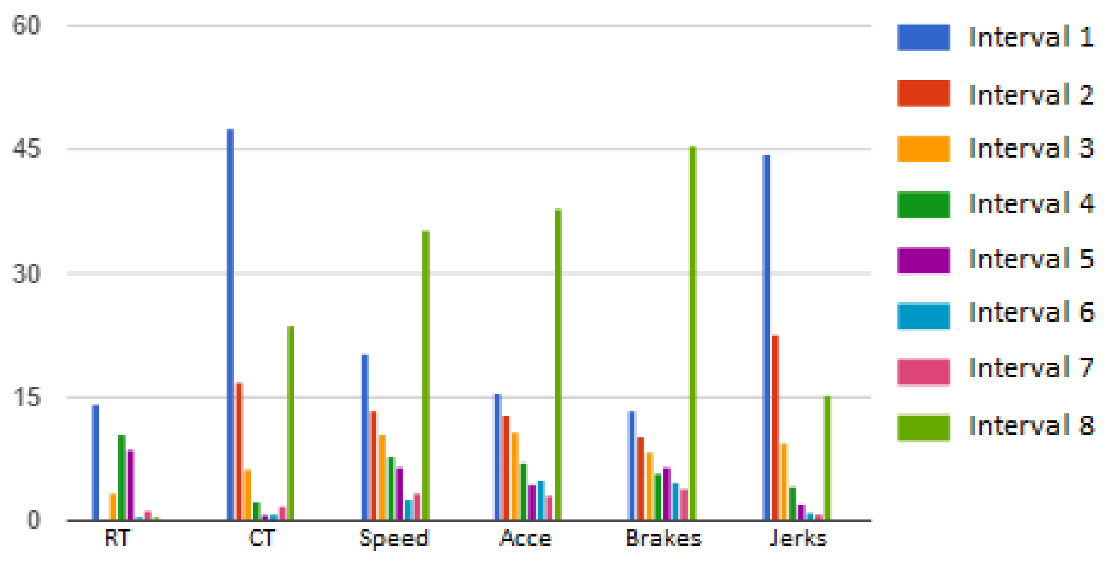


Figure 13: A bar chart of the distribution of metrics within the intervals for Driver 1

shows tendency lines for every participant throughout the experiment. It is important to notice this is merely tendency lines and not projection lines. Figure 15 shows a downward trend for 8 of the 10 drivers, with slope ratios between -0,284 and 3,273. It shows an upward trend for two of the drivers, with slope ratios at 0,13 and 3,54. The highest numerical slope ratios represents the drivers with the fewest amount of trips, and drivers with 0 trips later in the test period. What these results reveal, is that users of the system scores slightly better when having used the system for a period of time. This could mean that users of the system slowly improve their driving habits, dependent on the system indicating when they are actually a good driver.

Lastly, there is an interesting point to whether or not there is an incentive to keep using the system. As mentioned, the winner of the experiment received a prize at the end. As shown in Figure 16 there is a slight negative tendency in score summation, but it roughly correlates with the negative slope ratios of the tendencies in trip percentages. This means there was incentive enough to keep using the system, with the given prize. In a future insurance environment the incentive could be a cheaper insurance for the end user or engaging competitions.

5. RELATED WORK

P. Händel et. al. discusses the technology aspects of smartphone-based telematics, and highlights challenges in using smartphones as measurement probes [25]. They further suggest a number of metrics to differentiate between trips, and discuss the relevance and observability of these. The work is only concerned with the smartphone and its possibilities. It does not concern itself with implementing a system to support usage-based insurance.

P. Händel et. al. outlines a fully implemented system, capable of supporting usage-based insurance [20]. They further describe the release of said system and the collection of 250.000 kilometers worth of data in a span of 10 months. The authors present findings that prove data quality is a problem using smartphones for telematics. They do however not follow up on whether the metric-based detection of driving style worked as intended. It is also unknown if the system had any effect on the driving style for the end users.

6. CONCLUSIONS

This paper has discussed the entire system of Drive-LaB. It describes the design and implementation of the system described in *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model* [19] with a variety of changes and improvements. A goal of this paper is to propose a possible design and implementation for an experimental platform for usage-based insurance. This is clearly demonstrated in Section 2 and 3, design and implementation of Drive-LaB, respectively.

The paper offers a system that can provide informa-

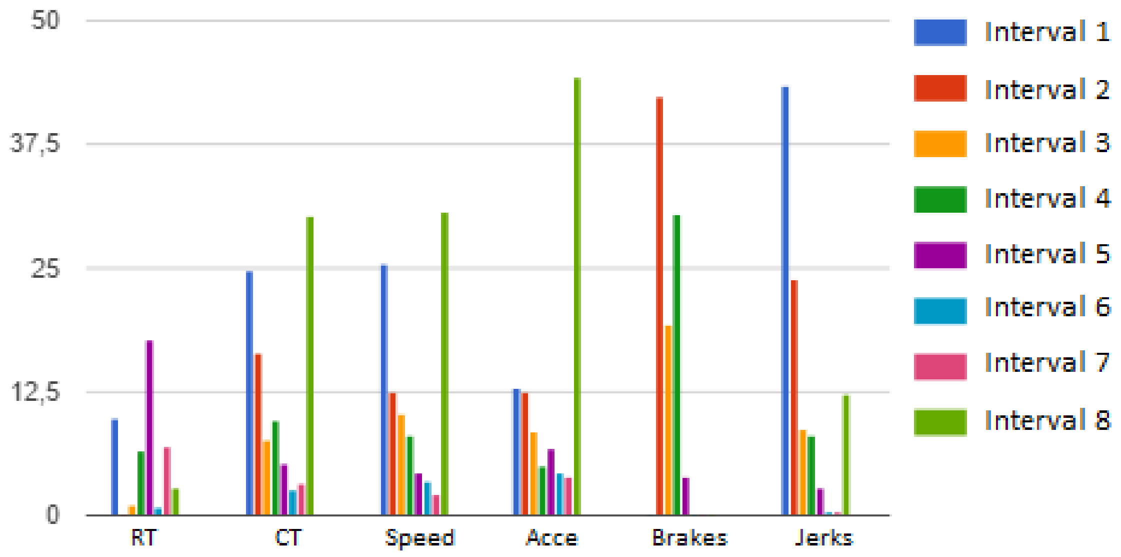


Figure 14: A bar chart of the distribution of metrics within the intervals for Driver 2

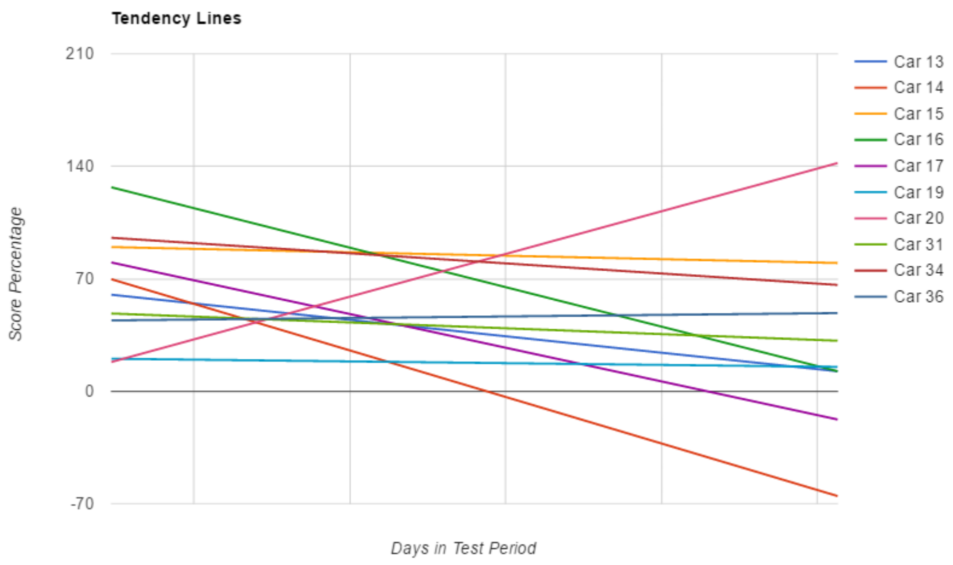


Figure 15: A line chart showing the tendency lines for each individual driver

tion that has not previously been available to insurance companies. It shows a driver profile in terms of comparable metrics, which can possibly be used to characterize and identify drivers with a higher risk of being involved in an accidents. This is proven by the experiments conducted in Section 4.4, Driver Profiling. Given a time period with this system in action, statistical data for risk assessment can be collected by monitoring which drivers actually are involved in an accident, and their driver profile could become a model to detect patterns in driving style that poses a higher risk. This type of data could benefit greatly from comprehensive collaboration with an insurance company, and is not part of this paper.

The paper presents a series of experiments on the implemented system. One experiment tests whether the system performs well in a real environment, comparable to commercial setting. Section 4.5, User Experi-

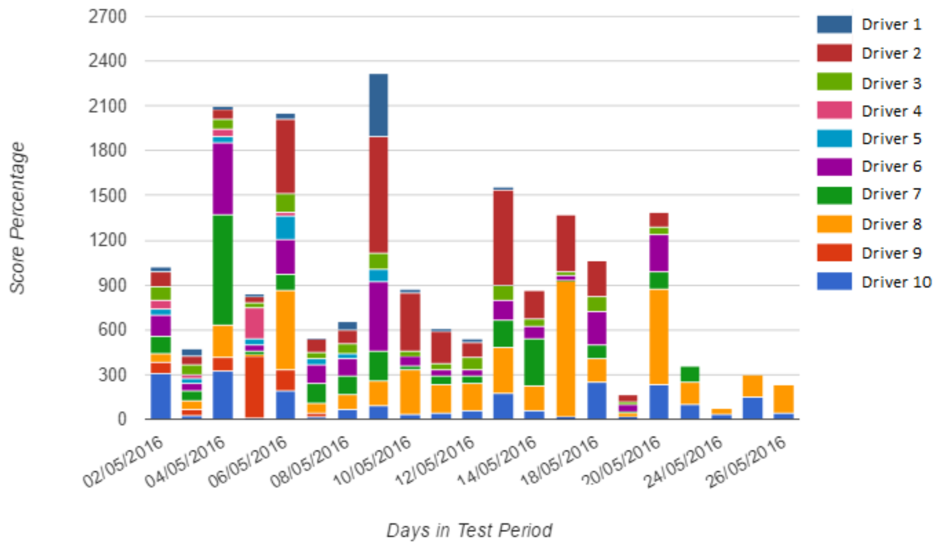


Figure 16: A summation of scores based on days

ments, put the system to the test. In conclusion it performed sufficiently, with excess performance capabilities to spare in the live setting, with at least 10 drivers continuously using the system.

Another experiment was conducted to examine whether users of the system understands the metrification chosen in the scoringmodel described in Section 1.1, Prerequisites. Additionally, these users should return their experience, opinion and comments to using the system. Part of these responses should be whether they consider the scoring mechanism fair. The test is described in-depth in Section 4.5, User Experiments. Unfortunately, their response did not arrive in time, to make it into this paper. Consequentially, the authors are unable to answer whether the Drive-LaB supports a fair and understandable metrification of driver styles.

On the other hand, it can be answered whether modern smartphones are adequate to support UBI. In Section 4.2, System Experiments, two applicability tests are conducted to examine how five different smartphones and two high quality GPS devices perform in cooperation with Drive-LaB. The experiment is conclusive that smartphones are not adequate for usage-based insurance in collaboration with Drive-LaB. The test displays too diversified scores among different types of devices, but a list of possible solutions to the problem are stated as well. The experiment concludes that Drive-LaB should explore the possibility to implement model-based signal processing and outlier rejection schemes, to make the scoringmodel compute scores on more stringent trajectories. If this could be achieved, the scores might align themselves, or at least become accurate enough to consider a calibration mechanism.

The platform itself, provides a basis for numerous experiments involving GPS coordinates and/or user interaction.

A possible extension for the existing system is to improved the competition implementation in terms of diversity -possibly handling a wider variety of competitions.

ACKNOWLEDGMENT

The authors thanks Lærerstandens Brandforsikring who provided valuable feedback that helped shape and evaluate Drive-LaB. Furthermore, their help in testing Drive-LaB provided the authors with significant amounts of data.

The authors also thank all alpha-testers who have downloaded and used Drive-LaB. This was a great help, both for debugging and further data collection.

References

[1] A. T. Balaei and A. G. Dempster (2009). *A Statistical Inference Technique for GPS Interference Detection*. IEEE 2009.

[2] A. T. Balaei and A. G. Dempster and J. Barnes (2006). *A Novel Approach in Detection and Charac-*

terization of CW Interference of GPS Signal Using Receiver Estimation of C/No. IEEE 2006.

[3] Allstate (2016). *Allstate Drivewise.* `https://www.allstate.com/drive-wise.aspx`.

[4] Bidouille (2016). *Android version distribution history.* `http://www.bidouille.org/misc/androidcharts`.

[5] C. Troncoso and G. Danezis and E. Kosta and J. Balasch and B. Preneel (2010). *PriPAYD: Privacy-Friendly Pay-As-You-Drive Insurance.* IEEE 2011.

[6] D. Foster (2004). *GPX: The GPS Exchange Format.* `http://www.topografix.com/gpx.asp`.

[7] Fabrice Marchal (2016). *TrackMatch.* `https://mapmatching.3scale.net/`.

[8] Geoffrey Huntley (2016). *Geo-Coordinate Portable.* `https://ghuntley.com/archive/2016/01/05/geocoordinate-portable-released/`.

[9] Google (2016a). *Activity.* `https://developer.android.com/reference/android/app/Activity.html`.

[10] Google (2016b). *bindService.* `https://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent,android.content.ServiceConnection,int)`.

[11] Google (2016c). *BroadcastReceiver.* `https://developer.android.com/reference/android/content/BroadcastReceiver.html`.

[12] Google (2016d). *Drive-LaB.* `https://play.google.com/store/apps/details?id=sw10.ubiforsikring`.

[13] Google (2016e). *FusedLocationProvider-Api.* `https://developers.google.com/android/reference/com/google/android/gms/location/FusedLocationProviderApi`.

[14] Google (2016f). *Handler.* `https://developer.android.com/reference/android/os/Handler.html`.

[15] Google (2016g). *Messenger*. https://developer.android.com/reference/android/os/Messenger.html.

[16] Google (2016h). *sendBroadcast*. https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent,java.lang.String).

[17] Google (2016i). *Service*. https://developer.android.com/reference/android/app/Service.html.

[18] Google (2016j). *ServiceConnection*. https://developer.android.com/reference/android/content/ServiceConnection.html.

[19] Gregersen, J., M. Jakobsen, and C. Laustsen (2016). *An Advanced Usage Based Insurance And Privacy-Secure Pricing Model*. AAU Student Report.

[20] Händel, P., J. Ohlsson, M. Ohlsson, I. Skog, and E. Nygren (2014). *Smartphone-based measurement systems for road vehicle traffic monitoring and usage-based insurance. IEEE.*

[21] Lane, N., E. Miluzzo, D. P. H. Lu, T. Choudhury, and A. Campbell (2016). *A survey of mobile phone sensing. IEEE.*

[22] Mono Project (2016). *Mono*. http://www.mono-project.com/.

[23] Newtonsoft (2016). *Json.NET*. http://www.newtonsoft.com/json.

[24] Npgsql Community (2016). *Npgsql*. http://www.npgsql.org/.

[25] P. Händel and I. Skog and J. Wahlström and F. Bonawiede and R. Welch and J. Ohlsson and M. Ohlsson (2011). *Insurance Telematics: Opportunities and Challenges with the Smartphone Solution*. IEEE 2014.

[26] PostgreSQL (2016). *PostgreSQL Open Source DBMS*. https://www.postgresql.org/.

[27] Progressive (2016). *Snapshot*. https://www.progressive.com/auto/snapshot/.

[28] QBE (2016). *QBE Insurance Box*. https://www.qbe.com.au/personal/quote/vehicle/insurance-box.

[29] QSTARZ (2016). *GPS Sports Recorder*. http://www.qstarz.com/Products/GPS\%20Products/BT-Q1300ST-F.htm.

[30] Research, I. (2015). *Smartphone OS Market Share, 2015 Q2*. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[31] Skog, I. and P. Händel (2016). *In-car positioning and navigation technologies - A survey. IEEE.*

[32] Skog, I., P. Händel, M. Ohlsson, and J. Ohlsson (2016). *Challenges in smartphone-driven usage based insurance. IEEE.*

APPENDICES

Table 10: Trip 1 - Aalborg to Haverslev

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Huawei Y330	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	36203.6	36244.9	36327.4	71450.1	36114.1	36215.7	38888.2
Time (s)	1444	1457	1467	1456	1394	1476	1452
Optimal score	34755.5	34650.1	34801.7	80452.8	34525.1	34694.6	37177.2
Tripscore	50190.9	40091.5	75063.2	81819.4	37010.3	37909.8	69955.7
Accelerations	81	69	174	5	48	29	125
Brakes	49	30	157	5	8	14	112
Jerks	152	126	407	11	61	46	300
Speeding (m)	1521.38	1202.71	1918.33	0	948.122	949.985	3242.5
Number of points	962	928	937	256	850	1475	1448

Table 11: Trip 2 - Haverslev to Aalborg

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Huawei Y330	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	28375	28196.7	28233.1	98400.9	28185.4	20808.7	46178.6
Time (s)	1209	1232	1246	1216	1210	925	1279
Optimal score	25963.1	25771.8	25805.1	110357	25705.1	19362.5	45462.9
Tripscore	58922.3	56780.8	28734.2	128056	27761.6	25372.5	72784.6
Accelerations	137	127	32	8	16	27	114
Brakes	95	112	13	8	13	26	97
Jerks	283	283	53	15	26	76	311
Speeding (m)	3164.46	3303.57	2064.87	7822.14	1202.25	1658	1471.53
Number of points	804	785	769	170	723	925	1279

Table 12: Trip 3 - Aalborg to Nørresundby

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Huawei Y330	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	13443.4	13415.4	13765.9	37611.7	13419.7	13509	22497.8
Time (s)	1767	1761	1777	1693	1794	1798	1855
Optimal score	13766.1	13744	14185.8	42256.7	13775.3	13867	23712.7
Tripscore	42751.4	24026	21012.4	50622.1	16927.1	20980.8	85138.6
Accelerations	175	66	64	31	25	78	249
Brakes	106	56	32	32	18	44	219
Jerks	310	66	96	31	25	137	583
Speeding (m)	1275.5	888.226	913.595	3310.11	567.519	652.36	4927.92
Number of points	1158	1087	1116	204	1060	1796	1798

Table 13: Nørresundby to Aalborg

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Huawei Y330	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	6493.1	14431.1	14467.9	9973.32	14417.6	14495.5	10113.1
Time (s)	755	1844	1819	315	1811	1856	1855
Optimal score	6502.84	15574.1	15584.8	11713.7	15545.1	15614.6	10593.5
Tripscore	23228.1	27530.3	19202.5	13082.1	18824.8	23916.6	27074.8
Accelerations	82	72	38	6	35	66	60
Brakes	59	63	32	4	26	38	50
Jerks	160	142	69	3	61	130	127
Speeding (m)	965.408	601.856	660.062	817.946	498.329	794.159	2333.21
Number of points	506	1140	1153	57	1072	1852	1854

Table 14: Aalborg to Haverslev

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	36396.2	36238.9	36402.6	36364.7	36344.3	36122.8
Time (s)	1432	1427	1458	1493	1417	1370
Optimal score	34831.1	34644.4	34764.5	34837.4	34745.1	34497.2
Tripscore	64511.4	46668.4	54563.5	37474.6	37800.1	41260.4
Accelerations	130	64	68	15	27	38
Brakes	90	59	87	5	10	25
Jerks	270	143	184	24	38	78
Speeding (m)	2659.4	2490.6	2568.04	2212.36	2389.12	2653.5
Number of points	970	922	933	936	1419	1373

Table 15: Haverslev to Aalborg

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	28152.9	28120.7	28200	28107.7	28175.9	28328
Time (s)	1190	1178	1195	1188	1203	1203
Optimal score	25703.6	25702.4	25774.8	25690.4	25780.9	25891.8
Tripscore	31074.8	48169.3	39439.1	29242.3	30397.4	37028.7
Accelerations	35	67	78	40	47	89
Brakes	23	63	97	13	70	171
Jerks	49	146	205	47	22	51
Speeding (m)	1389.58	2710.54	2148.87	1356.84	1246.78	1414.33
Number of points	786	751	757	701	1200	1202

Table 16: Aalborg to Nørresundby

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	13416.6	13460.5	13390.7	13410	13637.6	14037
Time (s)	1530	1526	1533	1533	1542	1539
Optimal score	13778.8	13817.2	13718.7	13745.2	13999	14472.2
Tripscore	17102.9	19029.6	27674.4	16671.8	26439.6	36530.7
Accelerations	33	48	68	32	83	147
Brakes	21	28	63	10	172	339
Jerks	48	86	128	32	55	98
Speeding (m)	1328.44	1496.71	1880.29	1482.53	1664.39	2020
Number of points	998	974	966	923	1542	1539

Table 17: Nørresundby to Aalborg

	OnePlus One	Samsung Galaxy S5	HTC One Mini 2	Samsung Galaxy S4	BT-Q1300ST(#1)	BT-Q1300ST(#2)
Distance (m)	14341.5	14409.3	14327.3	14369	14408.5	15051.8
Time (s)	1549	1545	1546	1539	1555	1554
Optimal score	14728.7	14805.5	14742.8	14771.3	14811.9	15473.2
Tripscore	18223.1	22779.1	29766.8	18094	25064.1	45327.1
Accelerations	56	77	85	46	94	187
Brakes	21	47	68	17	159	427
Jerks	57	129	172	43	44	134
Speeding (m)	1083.53	1105.97	1205.53	1101.66	1218.91	1442.7
Number of points	1030	990	980	914	1555	1526

Drive-LaB
Opstartsguide

Det skal du bruge for at komme i gang

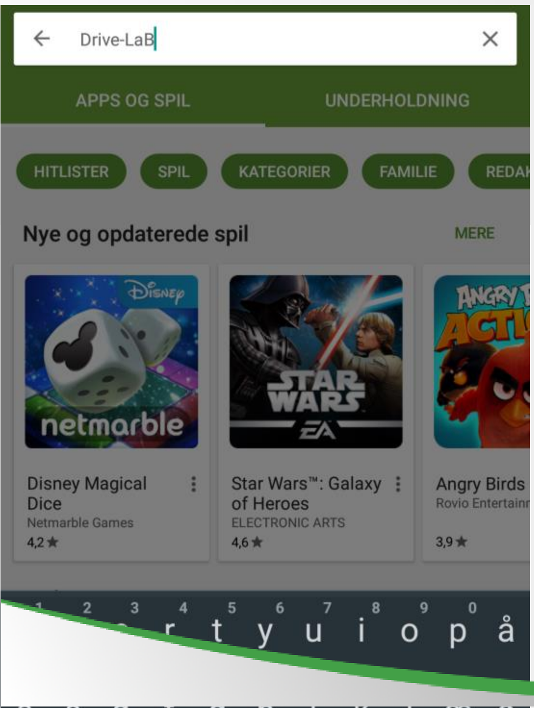
- Android 4.0 Smartphone
- Google Play Butik
- Internet (fx 3G, 4G eller Wi-Fi)

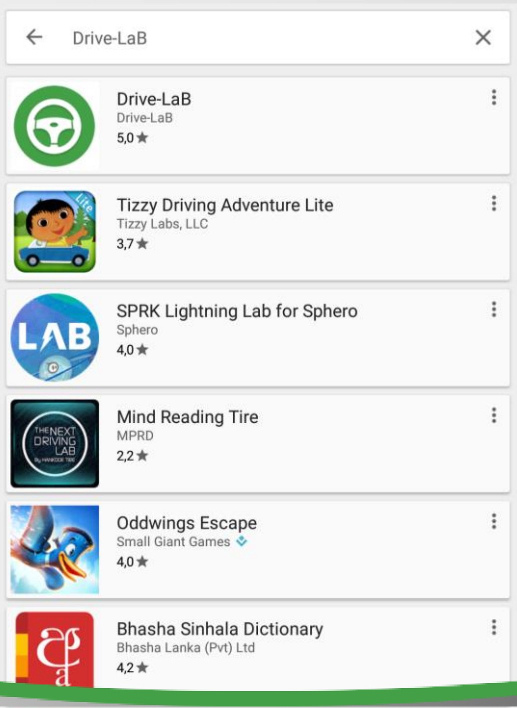
Din Android version kan findes under:
Indstillinger > Om telefonen > Android-version

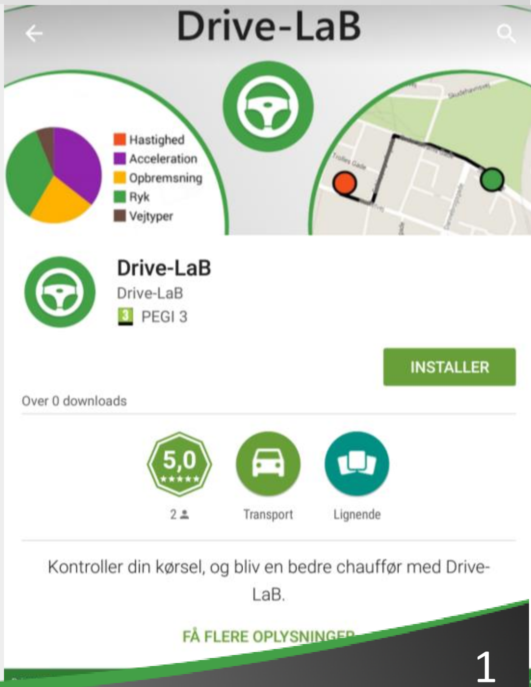
Sådan downloader & installerer du Drive-LaB

1. Åbn Google Play Butik
2. Indtast "Drive-LaB" i søgefeltet
3. Vælg Drive-LaB fra listen af resultater
4. Tryk på knappen *INSTALLER*

Herefter downloades og installeres applikationen.

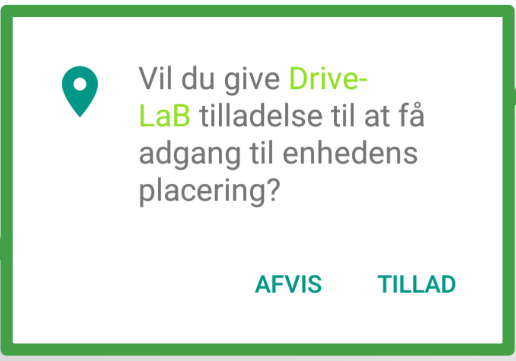






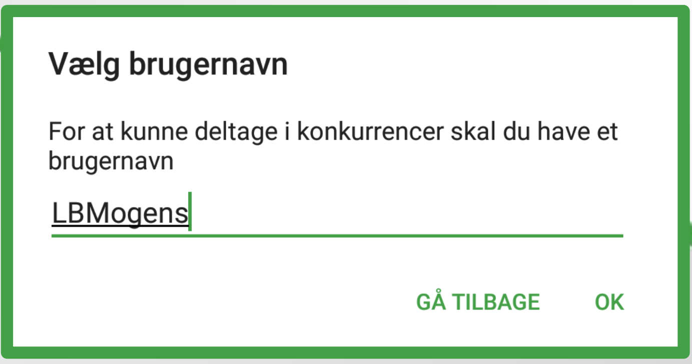
Kontroller din kørsel, og bliv en bedre chauffør med Drive-LaB.

Drive-LaB
Opstartsguide


Vil du give Drive-LaB tilladelse til at få adgang til enhedens placering?

Første opstart

Drive-LaB benytter sig af enhedens IMEI nummer, samt adgang til GPS. På Android 6.0+ vil Drive-Lab spørge om adgang til disse, første gang de benyttes. Tilladelserne er nødvendige for at Drive-Lab kan fungere korrekt, og du skal derfor afgive disse tilladelser når adspurgt.


Vælg brugernavn

For at kunne deltage i konkurrencer skal du have et brugernavn

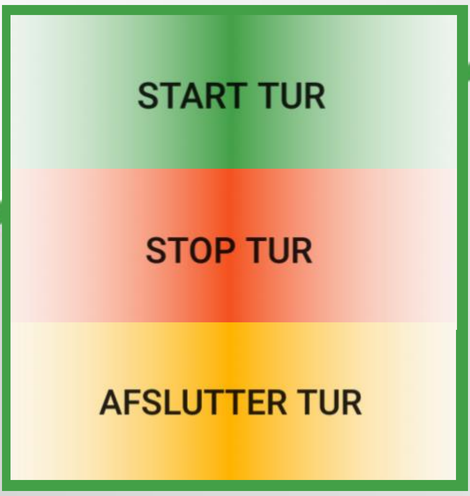
LBMogens

Sådan deltager du i LB-konkurrencen

1. Tryk på knappen *KONKURRENCER*
2. Du skal nu vælge et brugernavn

   NB: For at kunne se konkurrencen skal dit brugernavn starte med 'LB', fx 'LBMogens'.

3. Tryk *DELTAG* for at deltage i konkurrencen



Sådan tracker du din kørsel

1. Tryk på *START TUR* når du er klar til at køre

Drive-LaB er designet til at køre i baggrunden, og du kan derfor stadig gøre andre ting, eller slukke for skærmen.

2. Tryk på *STOP TUR* når din køretur er slut

Din tur vil nu blive behandlet og gemt i systemet (*AFSLUTTER TUR*). Denne proces kan vare lige fra et par sekunder, op til et par minutter. Efter turen er færdigbehandlet kan du igen starte en ny tur.

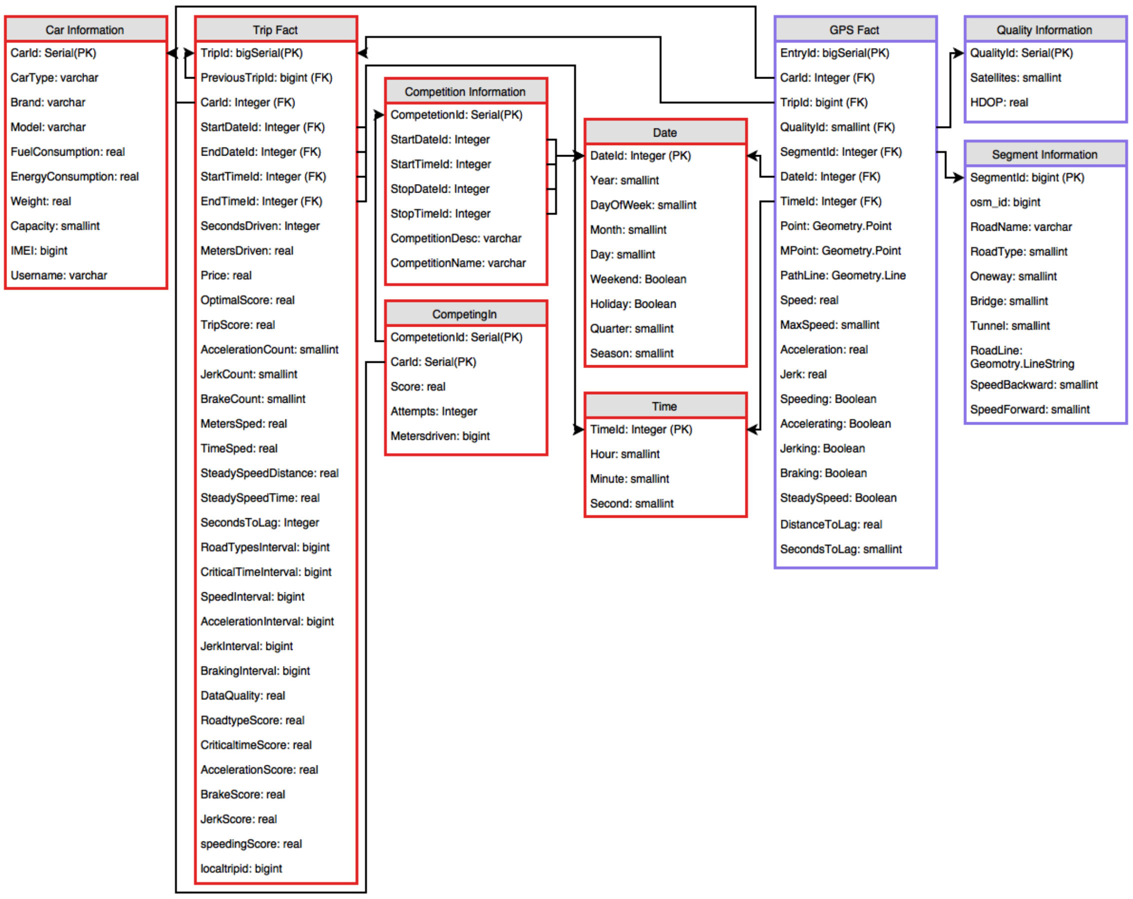


Figure 17: A picture of the entire data warehouse as it is used in the project