University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science
Software Engineering

Oliver Meus

Electric Vehicle Control and Journey Planning System on the Basis of Electric Motorcycle

Master's thesis (30 ECTS)

Supervisors: Helle Hein
Rainer Paat

Tartu 2014

Electric Vehicle Control and Journey Planning System on the Basis of Electric Motorcycle

Abstract

The purpose of the present thesis is to design and develop a complete universal open source control and journey planning system for electric vehicles. All materials and code will be public to help increase the public competence in electric vehicles and to popularize different custom electric vehicle projects. This thesis is going to take the reader through the process of designing and developing electric vehicles control and journey planning system. Each chapter covers separate part of the process in logical order. Different technical decisions and implementations are analyzed in each of the sections. Each of the chapters concludes description of outcomes.

Keywords

Electric vehicles, navigation system, control system, Android, Google Maps

Elektrisõidukite kontrollsüsteem ja teekonna planeerija elektrimootorratta näitel

Lühikokkuvõte

Käesoleva magistritöö eesmärk on välja töötada universaalne avatud lähtekoodiga kontroll- ja teekonnaplaneerimissüsteem elektrisõidukitele. Kogu informatsioon ning lähtekood süsteemi kohta saavad olema avalikud, et populariseerida elektrisõidukitega seotud projekte. Antud töö kirjeldab süsteemi valmimise protsessi ning analüüsib erinevaid tehnilisi aspekte. Iga peatükk käsitleb ühte konkreetset arendusprotsessi ning võtab kokku sellega seotud tulemused.

Võtmesõnad

Elektrisõidukid, navigatsioonisüsteem, kontrollsüsteem, Android, Google Maps

Content

1 Introduction

During last few years electric vehicles have become more and more popular all over the world. This has happened due to increasing environmental requirements in transportation and thanks to national contribution into spread of electric vehicles. Governments around the globe are spending millions of Euros on building infrastructure for electric cars, on subsidizing purchase of electric cars and on granting tax exemptions to owners of these environmentally friendly vehicles. But despite of the growth of electric cars traveling on the roads, it is difficult to find expertise and competence in this field.

Author of the thesis has been investigating and developing electric vehicles since 2007. He has designed and built the first street legal electric motorcycle in Estonia, contributed in development of electric go-kart and is currently participating in building another electric motorcycle. Author of the thesis believes that electric vehicles are the future of the personal transportation. That makes the present topic interesting and challenging.

The purpose of the present thesis is to design and develop complete open source control and journey planning system for electric vehicles. Open source is the desired approach to allow other interested people to take advantage of already existing system and to use it in similar projects. All materials and code will be publicly available to help increase the competence of electric vehicles and to popularize different custom electric vehicle projects.

The present thesis bases on electric motorcycle which is still being developed. The project has been funded by Enterprise Estonia and has final deadline of 1.09.2014 [1]. Electric motorcycle will be used to test and verify the first prototype of electric vehicles control and navigation system. Additional configuration or changes in system might be required to integrate the given system with various other electric vehicles, but the general purpose is to build it as universal as possible.

There are few other solutions already available on the market but none of these fulfills requirements for power-users. Different control systems are usually proprietary solutions that do not have enough configuration options. Most of the navigation and journey planning systems do not consider nature of electric vehicles by not taking into account range of the vehicle or locations of charging stations. There is no single system at all that would integrate all these features into one usable application. Therefore it was found that such solution is essential and could help in realizing of many other projects.

The present electric vehicle solution will consist of two different logical sections, each offering a batch of different features. First part of the system relates everything that is required to drive and control the vehicle. It visualizes important technical parameters to the user and therefore can be called a dashboard. It also provides different configurable options to the user to allow customizing the application for different vehicles and their needs. The second part of the final application is the navigation and journey planning system. It will provide multiple features to the user of electric vehicle. It enables user to browse charging stations, to query driving directions and to do many other map-based actions.

This thesis is going to walk through the process of designing and developing electric vehicles control and journey planning system. Each chapter covers a separate part of the process in logical order. Different technical decisions are analyzed in each of the sections. Each of the chapters concludes the description of outcomes.

2 Electric motorcycle

The goal of the thesis is to develop an open-source and generic control and navigation system that can be used on any electric vehicle. Nevertheless there are aspects which are unique to the vehicle where application is being used on. Therefore given application is designed, developed and tested especially for one specific vehicle - electric motorcycle. This chapter gives short overview of given motorcycle and its different mechanical and electronic parts.

2.1 Overview

Electric motorcycle is built based on BMW F650GS bike from year 2000. Complete motorcycle was taken as the base for the project. Engine, fuel tank, wiring harness were removed from the original bike. Frame, wheels and brakes remained original with some minor modifications to fit with new components. Lightning system of the bike remained original as well to be in compliance with current legislation.

2.2 Batteries

Batteries are one of the most critical parts of any electric vehicles. They are expensive, take much space, weight a lot and could be damaged when used incorrectly. Therefore it is essential to choose wisely among multiple different lithium battery manufacturers around the world. For this motorcycle batteries from United States company A123 Systems are used. A123 Systems is one of the best known and trusted battery manufacturer in the world. It has been producing and developing batteries for more than 10 years and done cooperation with car manufacturers such as Chevrolet. For electric motorcycle, 20 Ah prismatic cells were chosen mostly because of the good energy density and many available charging cycles [2].

2.3 Motor

EnerTrac MH602 liquid cooled dual hub motor is used as the engine on the motorcycle. It is the most powerful electric hub motor available and designed especially for motorcycles.

Given motor's housing includes actually two separate motors each with its own coils and wires. Both motors together are capable of developing up to 40 kW of power constantly or up to 60 kW peak. EnerTrac hub motor is Brushless Direct Current (BLDC) motor which makes it reliable and durable [3].

2.4 Controller

Each BLDC motor needs an electric controller to be operated. As EnerTrac motor contains two separate motors, also two separate controllers are used on the motorcycle. Controllers used on the electric motorcycle are manufactured by company called Sevcon. Controller takes inputs from throttle sensor, brake lever sensor and motor position sensor and uses given data to supply motor with correct amount of electric power. Controller is also responsible for producing electric energy and storing it in the batteries when regenerative braking is engaged.

3 Requirements

Requirements analysis has been done to specify functional and non-functional requirements to the whole electric vehicle control and navigation system. This chapter presents these requirements for both software and hardware that is going to be developed.

3.1 Functional Requirements

The new system should fulfill the following functional requirements.

1. Displaying speed of the vehicle
2. Displaying power consumption
3. Displaying total travel distance
4. Displaying current time
5. Displaying temperature of the battery, motor and controller
6. Displaying current battery state of charge
7. Displaying estimated remaining driving distance before battery runs out
8. Displaying map and current location of the vehicle
9. Ability to navigate and zoom in/out on the map
10. Ability to focus the map to the current position of the vehicle
11. Having a list of Estonian public charging stations sorted by the distance from the present location
12. Showing charging stations on the map
13. Ability to refresh the list of the charging stations
14. Showing important data of the charging stations such as the name, address and current state (occupied or available)

15. Calculating and showing optimal driving directions to any point on the map, to any address or to any charging station
16. Routing driving directions through charging stations if required
17. Calibration of the speed and distance

3.2 Non-functional Requirements

The new system should fulfill the following non-functional requirements.

1. Whole system must be rain and dust proof
2. System must tolerate vibration
3. Accuracy of every parameter being measured and calculated must exceed 95%
4. System must be able to connect with the electric motorcycle
5. System must be able to connect to the Internet in 2G, 3G and LTE networks
6. It must be possible to configure the system to be used with other electric vehicles
7. Cold start of the system must take no more than 1 minute
8. Hot start of the system must take no more than 10 seconds
9. System must be reliable
10. Using the system must not affect the driving safety compared to traditional vehicle instrumentation
11. Source code of the system is open

4 Existing Solutions

Everything about electric vehicles (EV) is very fresh and so is the existing market of gadgets and solutions for electric vehicles. Nevertheless there are few solutions which are made to satisfy at least some of the same goals with our electric motorcycle control system project. This chapter will give an overview of those existing solutions. It will reveal advantages and disadvantages of the programs and make conclusions. It will be decided whether or not it is reasonable to take some of the existing solutions as base or start developing a completely new system.

4.1 EOLAS-EV

EOLAS-EV is cloud based routing platform developed by Abalta Technologies [4]. It is a map software which reads input from electric vehicle battery and calculates the range based on the current location of the user. What makes this software unique from others is its polygon approach which does not require user to enter specific destination. It shows the range on the map as a radius from current position depending on the actual possible distance in any specific direction [4]. Figure 1 shows the polygon approach in iPhone application.

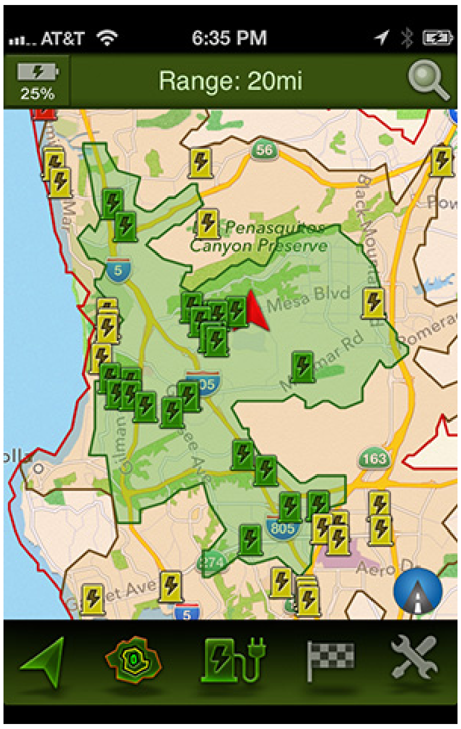


Figure 1: EOLAS-EV iPhone application [5].

EOLAS-EV can be especially adjusted to be used with two wheeled electric vehicles. It offers great accuracy in range calculations thanks to the cloud based server, real time traffic information and vehicle specific data input. It also uses important map data, such as slopes and types of road. It allows user preliminary to download maps and use the application offline as well as use online maps. Application also displays charging stations [4].

For the purpose of this project, the examined software is not suitable because of its very limited functionality. Moreover, the further development of this software is strictly addressed to US vehicle manufacturers and there are no plans of making it available to the wider public.

4.2 EMotorWerks EV Android Dashboard

EMotorWerks is the company based in United States developing and selling complete solutions for electric vehicle conversions. The main argument for EMotorWerks is simple modular design of the conversion kit. All modules will fit to the specific car models so that only typical automotive mechanical skills are required. Also, the choice of modules allows different customizations [6].

A monitoring system called Android Dashboard is developed as one module to the system. It has been built for tracking and visualizing electric vehicle dynamic parameters, battery state of charge and any other related data. The system can be configured to be used with any electric vehicle. It can be used with both phones and tablets running at least Android version 2.3. The module itself and screenshot of the Android application can be seen on Figure 2.

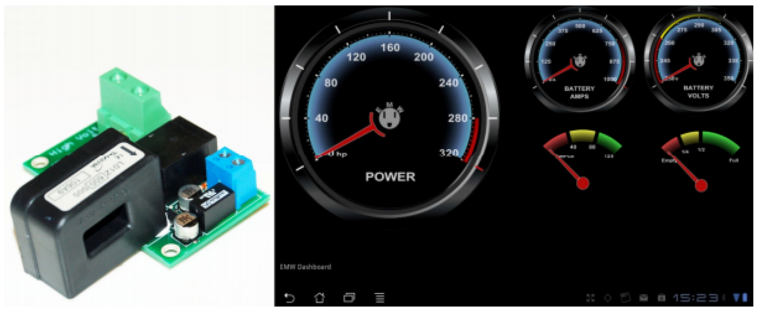


Figure 2: EMotorWerks EV Android Dashboard module and screenshot [6].

One main component of the Android Dashboard is the bluetooth unit which is used for establishing connection between the vehicle and Android device. For current measuring it uses the Hall Effect sensor which is capable to measure up to 1000 amperes. It also measures voltage. Both values are transferred to mobile device which calculates rest of the data. The given system does not have any map support. It only displays important data of the vehicle [6].

Lack of the map support, expensive price of 250 Euros and unreliable bluetooth connection make this platform not suitable for electric motorcycle project.

4.3 Tumanako's Dashboard

Tumanako Group is producing different open source hardware and software systems for electric vehicles. Source code, schematics and many other artifacts have been made available for public access. The team is working on multiple different products, for example battery management system and electric motor controller. They are also making an open source Android application called Tumanako's Dashboard [7].

The purpose of the application is to visualize important vehicle data to the driver. The data to be shown are divided between three different groups [7].

1. Primary driver data - data which need to be known regularly and easily.
2. Secondary driver data - data which need to be known less regularly.
3. Technical data - data which are needed only for different analysis or fault investigation.

Screenshot of the application can be seen on Figure 3.

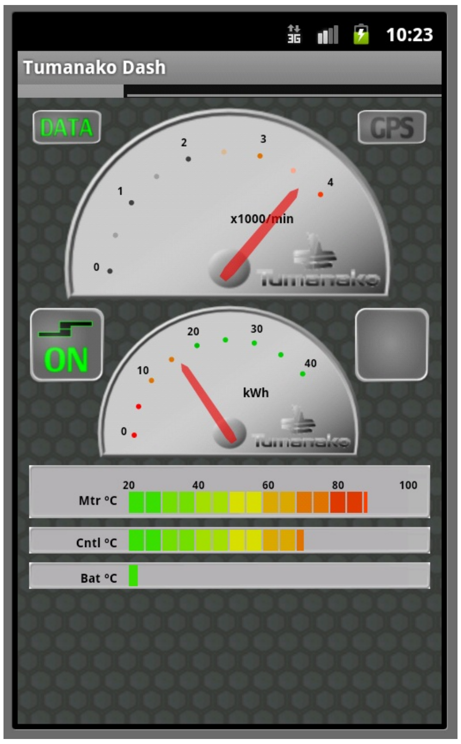


Figure 3: Tumanako's Dashboard Android application [7].

Information on the main application screen is considered as the primary driver data. It contains values such as main contactor state, motor speed, power consumption and temperature values of motor, controller and battery. Secondary driver data and technical data can be accessed from the menu. It includes battery voltage, total battery capacity and fault log [7].

Bluetooth module is used for connecting Android device with electric vehicle. Currently Tumanako Group is working on the module which works only with their own motor controller, which makes it useless for our electric motorcycle project. They do not have any plans on map based features either.

4.4 Nissan Leaf

Nissan Leaf has been the world's best sold all electric car since its launch in December 2010. With over total of 100 000 sold cars it has reached 45% of the market share among electric vehicles [8]. Nissan Leaf also has a powerful central navigation unit with many different functions. It has been tested by thousands of users and improved over last few years. This section will give an overview of the system to understand how it works and how this knowledge could be used in the present electric motorcycle control system.

Nissan Leaf central unit looks like any other embedded car navigation system on the first look but in addition to conventional navigation functions, the Leaf system offers a wide range of functions specific to driving an electric vehicle. These features make the system interesting for electric motorcycle project (see Figure 4) [9].

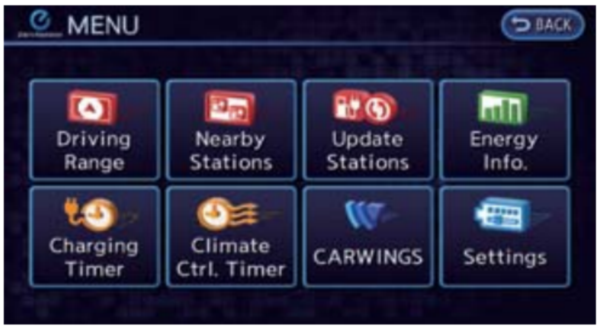


Figure 4: Nissan Leaf navigation system main menu [9, p. 2-2].

The main features of the system are following:

- **Driving range** is displayed on the map as a circle around car's location. There are two circles: one is called extended range with fully charged battery. Second radius shows the normal estimated driving range with current state of battery. Numeric value of the latter is also displayed on the actual dashboard. Elevation data are not used in calculations [9, p. 2-4].

- **Charging stations** are displayed as icons on the map. Different type of icons are used for quick and normal chargers. Clicking on the icon provides options to navigate to the station or to check availability. It is also possible to show charging stations as a list where closest chargers are in the top of the list. List of the chargers can be updated from server at any time [9, p. 2-6].

- **Energy information** shows data about energy usage and economy. It includes estimated driving range, average energy consumption and instant consumption. It is possible to view history of last 10 usage periods. Each period has to be reset manually. Energy information screen also shows the estimated impact of climate control on driving range, electric motor power, climate control power and energy consumption by all other accessories [9, p. 2-9].

- **Timer functions** allow to control charging batteries and climate control automatically at specified time and day. It is possible to set time when charging should start and time when it should end. It is also possible to change the maximum charge level to 80% to extend the battery life. Climate control timer allows changing the expected temperature to be achieved [9, p. 2-10 to 2-13].

- **Battery and power information** screen is used for warning messages related to battery. It starts first appearing when user sets a destination out of charge range. It might give the extra help tips to the user about extending the range with turning off climate control. This screen also appears when battery charge level reaches critical or when output power is limited for technical reasons [9, p. 2-17].

4.5 Tesla Model S

Tesla Model S is probably the most well known high-end production electric vehicle. Despite being developed and built in United States, Tesla has been able to maintain competitive price, excellent quality and great safety ratings. In addition to that, Tesla has done excellent job with the biggest issue in the world of electric vehicles: range. Model S is able to travel up to 420 km with its 85 kWh battery pack [10].

Model S features a 17-inch central touch screen unit (see Figure 5) which is used for controlling almost everything: media, navigation, web browsing, climate control and car settings, phone and energy settings. This section will give overview of the electric vehicle specific features of the onboard controlling unit [11].

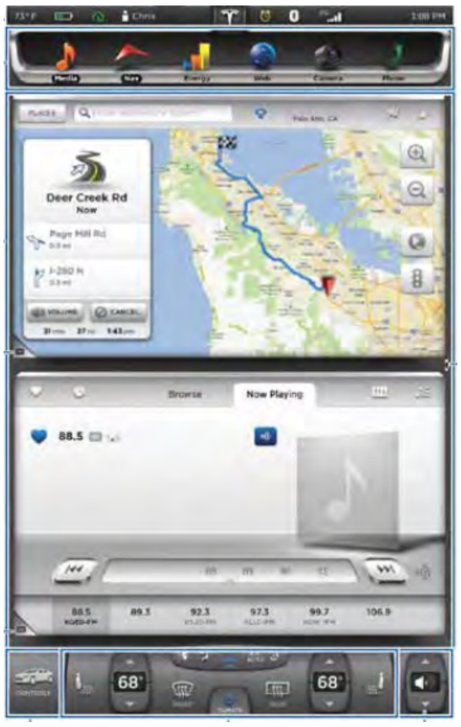


Figure 5: Tesla Model S central screen [11].

The energy usage screen displays the graph that visualizes energy consumption and regeneration of the vehicle. It also displays the estimated range. Graph scale can be changed from past 10 to 50 kilometers. Consumed energy is displayed as orange and surplus energy as green on the graph [11, p. 4.19].

Maps of Model S are based on Google Maps. They provide step-by-step directions and voice navigation. Charging points are displayed on the map, however there is no option to get charging stations as a list. It is possible to get driving directions to any point on the screen, including chargers. Range estimation is not part of the navigation system at all [11, p. 5.20].

Tesla's central system has much less electric vehicle specific features compared to Nissan Leaf. One reason might be the great driving range of Model S - driver is not required to pay so much attention on these parameters. But it also needs to be taken into consideration that Tesla is constantly developing the system and providing over-the-air software upgrades. It might happen that new features will be added soon.

4.6 Conclusion

After reviewing the existing electric vehicle dashboard solutions currently available on the market, it was clear that there is no existing software matching to our electric motorcycle project's needs. Two open source projects were examined as well, but it was decided that there is no good reason to take any of them as a base for this project. But still, reviewing the existing software has given many useful thoughts and ideas for implementing the custom solution. Table 1 presents an overview table of all applications covered in this chapter.

Table 1: Comparison of existing solutions.

	EOLAS-EV	**EMotorWerks EV Android Dashboard**	**Tumanako's Dashboard**	**Nissan Leaf**	**Tesla Model S**
Map/Navigation	Yes, with real time traffic information, elevation data and road types	-	-	Yes	Yes
Motor parameters	-	power (kW) based on battery current	revolutions per minute; main contactor state	power (kW)	-
Battery parameters	Measured and used for range calculations, not shown to the user	voltage; current; capacity (calculated from voltage); state of charge	voltage; current; state of charge; capacity	state of charge	state of charge; total capacity
Temperature	-	battery	motor; controller; battery	battery	-
Online/offline	Needs constant connection	Offline	Offline	Online/offline	Online/offline
Range estimation	Yes, polygon view on the map, depends on elevation	-	Yes, elevation not considered	Yes, elevation not considered. Range with and without climate control enabled. Displayed on map as well	Yes, elevation not considered

Connection interface	Unknown	Bluetooth	Bluetooth	CAN bus	CAN bus
Charging stations	Yes, currently US only	-	-	Worldwide. Can be updated manually. On map/as list	Worldwide. Can be updated manually. On map only
Price	Unknown	250 EUR (sensor unit and bluetooth data module)	Application - free; Bluetooth module and sensors not available yet	System available only with the car	System available only with the car
Open source	No, purely commercial	Yes - software only	Yes	No	No
Timer	-	-	-	Yes, for charging and climate control	Yes, for charging
Dynamic directions (using chargers' location)	-	-	-	-	-

5 Choice of Platforms

Choosing the best available platform for electric vehicle control and navigation system is a critical decision which needs to be well considered and verified to avoid major issues and costs in later phases of development. The platform in the given context represents both hardware and software base for the given system. As it was declared in the previous chapter that there was no suitable existing solution available at given time, choice of platform is not limited in any ways. Although it is recommended to choose between platforms with considerable community, knowledge base and support underneath. This chapter will explain advantages and disadvantages of available hardware and software platforms.

5.1 Arduino

Arduino is a single-board microcontroller intended for making prototyping and developing small real time systems more accessible. The board is built around Atmel AVR or ARM microcontroller. It provides necessary circuitry for small or medium size applications: microprocessor, input and output pins, memory and clock generator. By default Arduino's microcontroller has pre-programmer bootloader which allows uploading programs without need for special chip programming tools. It is ideal for developing applications without spending much time on the hardware layer [12].

Writing code for Arduino boards has been made as simple as possible. Arduino provides the cross platform Integrated Development Environment (IDE) which allows uploading programs to Arduino board without any other software. It also provides the basic debugging tools, code editor and compiler. All Arduino applications are written in C or C++ [12].

Everything in Arduino is open-source: code, circuit diagrams and many sample applications. This fact has made possible of existing have hundreds of different Arduino extensions and libraries available on the market. Small extensions which connect directly to Arduino pin connectors are called Arduino Shields. There are many official shields, for example wifi and bluetooth shields for adding connectivity to Arduino board, but there are even more unofficial extension boards like LCD screens or small motor controllers. All shields usually come with software library that allows using it without knowing low level details [12].

Arduino is considered as a good choice for prototyping different applications. Wide range of extensions and software libraries make it perfect for many projects. Unfortunately it is not thought to be stable enough for long term use.

5.2 Raspberry Pi

The Raspberry Pi is credit card size single board computer developed in UK. It connects to the regular keyboard and screen and could be used for games, word processing and many other applications that any regular PC has. The main purpose of the Raspberry Pi is promoting teaching of basic computer science and programming in schools [13].

Raspberry Pi primarily runs on Linux kernel-based operating systems. Although Python is recommended, especially for learners, as an official programming language for Raspberry Pi, this is not the only coding language which works. Raspberry Pi operating systems come with Python, Java, C, C++, Scratch and Ruby which have been installed by default. But it is also possible to run programs in any other language that compiles for ARMv6 architecture [13].

Raspberry Pi is capable of playing 1080p videos thanks to its CPU is running at 700 MHz and GPU that supports OpenGL ES 2.0 and hardware accelerated H.264 coding. Overall performance is considered comparable with 300 MHz Pentium2 PC [13].

Size, performance and price of the Raspberry PI make it considerable candidate for using in the present electric motorcycle project. As any regular VGA capable display can be used, finding a good screen should not be an issue. Raspberry PI also provides all needed interfaces to establish connection to electric motorcycle central controller unit.

5.3 Android

Android is an operating system for mobile devices that delivers a complete set of software for using all basic features of the device. It has been built up from ground with a purpose to enable developers to take full advantage of everything that device has to offer. It has been built to be open in every way, including the open source code. Android is being developed on top of Linux kernel but unlike other Linux distributions, it includes custom virtual machine to optimize memory and other resources usage in limited mobile environment. Android also offers a great range of programming interfaces for developing world class applications. It has support for services such as location, networking, maps and motion detection [14].

As Android is merely an operating system, there is a wide choice of different devices running Android available on the market. Each device manufacturer usually makes modifications to the operating system to make it run smoothly on particular hardware and also to make it more recognizable for the user. Despite endless possibilities for modifications there are number of requirements that device needs to follow in order to be sold with Android running on it, called Android Compatibility Definition. These requirements ensure that device would be capable of running any application written on top of Android Application Programming Interface (API) [15].

In addition to wireless networking, Android also provides Android Open Accessory Protocol which allows external accessories to interact with Android device over USB connection. In this mode, external accessory will act as USB host and provide charging power for the android device [16].

For electric motorcycle control system Android could offer a great choice of services and programming interfaces, like embedded Google Maps support, built in GPS and motion detection sensors and networking capability. Android Accessory protocol is ideal for connecting Android device with motorcycle central controller unit. Thanks to Android Compatibility Definition it is possible to choose between wide range of devices even in the latest phases of the development.

5.4 Conclusion

As a result of comparing advantages and disadvantages of three different platforms it was chosen to start developing electric motorcycle control and journey planning application for Android operating system. Main advantage of the Android is the biggest community with over a million available applications on Google Play Store [17]. Android also has the best performance among these widespread platforms and it provides the biggest number of embedded services and most varied API for different purposes. Also wide choice of devices running Android as an operating system allows choosing the device with best fit in terms of size, screen and other specifications. In addition to application, also additional hardware is required for connecting Android device with electric motorcycle. For that purpose Arduino was chosen as a best choice for developing a prototype. Arduino offers wide range of extension shields and libraries that make using it as a connecting link between Android and motorcycle's network very tempting.

6 Hardware Solutions

In addition to software, this thesis also covers hardware that is required to build and use electric motorcycle control system on the actual motorcycle. This chapter gives an overview of Android device that is used in the system. It also describes network which is used to transmit data from motorcycle controlling systems to the Android and it gives an overview of the prototype devices used for connecting Android device with the central network of motorcycle.

6.1 Sony Xperia Z Ultra

Sony Xperia Z Ultra smartphone was chosen to be used as electric motorcycle control solution central screen and CPU. The main advantages of the device are bright and big screen with diagonal dimension of 6.4 inches and IP58 certified casing that should make it as waterproof as required to be used on the motorcycle. Device is running on the latest Android 4.4 KitKat version. It provides GPS which is required for navigation and LTE network capability for fast mobile data connection. It also supports both bluetooth and Android Accessory mode connections to be connected with the CAN (controller area network) network on electric motorcycle [18].

6.2 CAN Bus Overview

CAN is a bus standard defined by International Standardization Organization (ISO) for allowing automotive microcontrollers to communicate with each other without a central controller unit. It was developed for replacing complex wiring harnesses with two wire bus. Its cost and performance provide the needed flexibility for many different systems [19].

Main features of CAN include [20]:

1. Error detection, notification and recovery - if any of the system modules detects an error, it will notify all other units as well. Original message will then be repeated until transfer is successful.

2. Connection - there is no logical limit for amount of devices connected simultaneously.

3. Connection speed - it supports any connection speed that is suitable with the amount of devices in the same network.

4. Flexibility - units in the network have no identifying information. Therefore they can be added or removed without any configuration.

5. Message transmission - if bus is unoccupied, any of the modules can send a message. If two units send a message at the same time, priority will be decided based on the ID of the message. Device which is sending a message with lower priority, will go into receiving mode.

6. Remote data request - data can be requested from other modules.

6.3 CANopen

CANopen is internationally standardized protocol and profile specification for embedded systems based on CAN. As underlying CAN specifies physical structure of the network, CANopen describes application layer and communication profile, also devices and interface profiles. CANopen is used by the electric motorcycle controller. By defining application and communication profiles, CANopen specifies large amount of standardized applications and devices. It describes behavior and parameters for each profile. It enables communication between devices from different manufacturers and guarantees their interchangeability [21].

Each CANopen device needs to implement certain features in its software (see Figure 6):

- Communication interface - defines service data objects (SDOs) which are mostly used for configuration or transmitting larger blocks of data. It also defines process data objects (PDOs) which are used for transferring most of the recurrent data [22].

- Object dictionary - standardized device description, which describes parameters, functions and all other important data of the device [22].

- Application process - implementation of actual functions of the device [22].

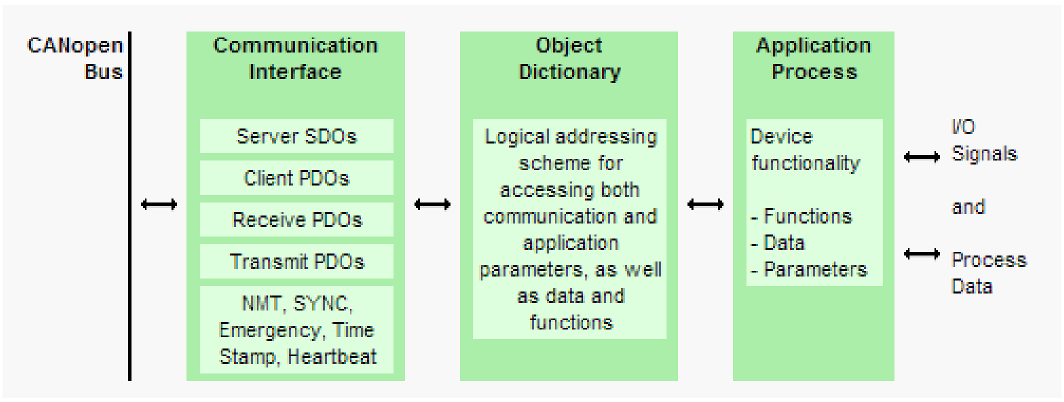


Figure 6: CANopen implementation [22].

6.4 Android USB Accessory

Android USB Accessory protocol allows designing and implementing USB accessories specially for Android powered devices. USB accessory must follow specific design and protocol outlines to be compatible with Android devices. It is important to notice that Android device will act as an USB slave whilst accessory itself must implement USB host capability. This also allows charging the device through same USB cable. Android USB accessory mode has been natively supported since Android version 3.1 [23].

But not all Android devices support USB accessory mode as it is dependant on the hardware. Each accessory must check if connected device is capable of talking in accessory mode. There is a special ACCESSORY_GET_PROTOCOL request for querying device protocol version. Response could be either an error which means no accessory support or some of following [24]:

Protocol version 1:

- 0x2D00 - accessory mode (no support for audio accessories)
- 0x2D01 - accessory mode and Android debugging bridge (ADB)

Protocol version 2:

- 0x2D02 - audio accessories only
- 0x2D03 - audio accessories and ADB
- 0x2D04 - other accessories and audio accessories

- 0x2D05 - other accessories, audio accessories and ADB

This response is required to confirm that Android device has been connected and ready to receive data.

For enabling developers to realize prototypes that match with Android accessory criteria with minimum effort, Google provides firmware written in C++ [25] for Arduino Mega 2560 based board [26] and Max3421E USB host controller shield [27]. These two boards with given firmware allow to exchange data between Android device and anything connected with input/output pins of the CPU on Arduino Mega 2560.

6.5 CAN Bus and Android Accessory

ATMega2560 CPU, which is used as an reference example for Android Accessory is not capable of listening CAN bus signal. Therefore, an additional shield is required. MCP2515 CAN bus controller based shields and boards are most commonly used with Arduino boards. SeeedStudio has built one that can be used directly with Arduino Mega 2560 board [28].

SeeedStudio CAN bus shield follows Arduino standard pin layout which allows using it with any Arduino board. It implements CAN open protocol and provides terminals for connecting directly with motorcycle CAN network. SeeedStudio provides Arduino compatible firmware that can be used for transmitting CAN signal from this shield to the connected Arduino board [28].

In conclusion, Arduino will take care of forwarding data from SeeedStudio CAN shield which is connected with motorcycle CAN network to Max3421E USB host controller shield which in turn is connected with Android device. Using the provided firmwares, it is not difficult to implement the given prototype. Example code for forwarding data from CAN bus to Android device is seen on Figure 7.

```
void loop() {
    if(MCP_STAT_RXIF_MASK == CAN.checkReceive()) {
        CAN.readMsgBuf(&len, buf);
        if (acc.isConnected()) {
            acc.write(buf, len);
        }
    }
}
```

Figure 7: Code sample for forwarding data.

6.6 Bluetooth Support

In addition to support of Android USB Accessory mode, Android application has also implementation of bluetooth interface that allows using CAN to bluetooth converter instead of USB device. Such bluetooth devices are available and enable to connect any bluetooth enabled device with CAN networks.

The current implementation is based on devices with ELM327 microcontroller which is one of the most commonly used chips in CAN-bluetooth adapters. ELM327 has wide range of different configuration parameters available that make it perfectly suitable for this project [29].

Electric motorcycle dashboard application allows user to choose the bluetooth device desired for the data transfer. It then initiates the connection using Android Bluetooth API [30] when device is in the bluetooth range and uses it for getting data from CAN network (Figure 8).

```
BluetoothDevice device = bluetoothAdapter.getRemoteDevice(address);
BluetoothSocket socket = device.createRfcommSocketToServiceRecord(uuid);
socket.connect();
inputStream = socket.getInputStream();
outputStream = socket.getOutputStream();
```

Figure 8: Code for initiating bluetooth connection.

When connection with bluetooth device is established there are instances of input and output streams which remain open until bluetooth connection is disconnected. The functions `inputStream.read()` and `outputStream.write()` are used to send and receive data to the ELM327 adapter.

6.7 Conclusion

USB prototype device was implemented as part of the system that allows connecting Android device with electric motorcycle CAN network. As CAN is industrial standard for automotive controllers, similar device could be used for any electric vehicle. Its only task is forwarding data which makes it universal for different purposes. Prototype for running software described in this thesis consists of following parts:

- Android device - Sony Xperia Z Ultra
- Arduino Mega 2560 [26] with integrated Max3421E USB host controller [27]
- SeeedStudio CAN bus Shield [28]

All other items expect the Android device combined can be seen on Figure 9. Using integrated USB host controller helps saving some space but still allows using the same firmware with as external USB host shield.

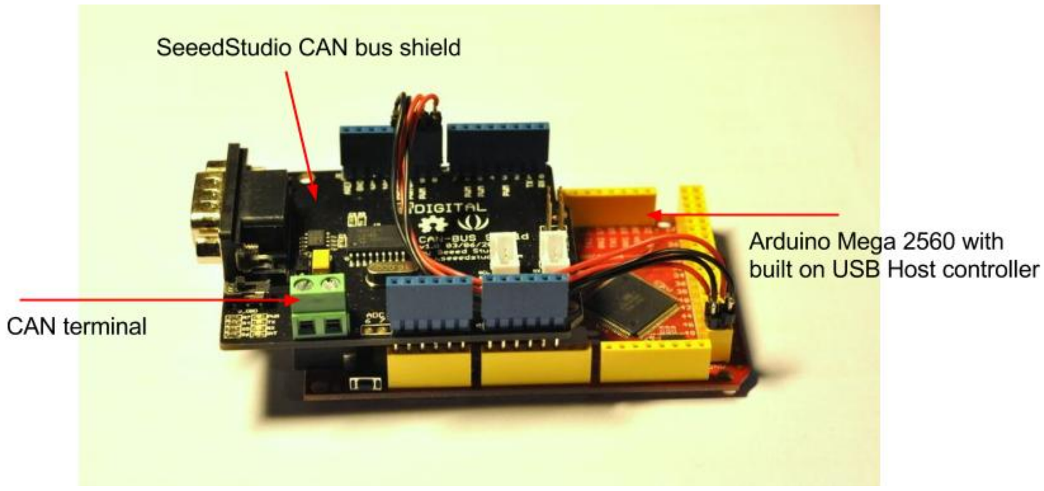


Figure 9: Prototype device for connecting Android with CAN.

The support for general purpose ELM327 microcontroller based CAN to bluetooth converters was also implemented. These converters are available in the Internet and therefore enable electric motorcycle dashboard application to be used without any custom made hardware. In some cases it might be much more convenient to use wireless bluetooth connection even though it is less reliable. ELM327 bluetooth interface can be seen on Figure 10.

Figure 10: ELM327 Bluetooth interface device

This chapter described two different implementations how Android device where electric motorcycle dashboard application is running can be connected with CAN network: USB or bluetooth. As data sent over the different protocols serve the same purpose, the next chapters concentrate only on what data is being sent and do not differentiate the actual channel being used.

7 Software Solutions

Most parts of the given thesis and outcomes of the work are related with different pieces of software. There is a lot of new software implemented to achieve results described in the thesis but there is also very much existing software and programming interfaces that are essential parts of this work. The purpose of the present chapter is to give overview of the software and interfaces that are used to deliver the final outcome.

7.1 Android API

Android provides a complex and rich framework for developing applications. Android applications are written in Java on top of Android application programming interface (API). Regular Java bytecode will be converted to be compatible with Dalvik which is a process virtual machine in Android. Each application runs in its own virtual machine assigned with an unique Unix user. By doing that it is ensured that every application has access only to its own files and data [31].

For building Android applications there are multiple different built-in components available. Each of the blocks can be used by the operating system as an entry point into the application, but each one servers completely different purpose in the application lifecycle [31].

Activity is a component that provides the user interface. Usually each activity implements one specific screen or view. Main activity of the application is started when user launches the application. Additional activities can be defined to be launched on different user actions [32].

Service is a component used for long time background tasks. Service does not have its own user interface and it does not require any other parts of the application to be running. Service could be used only by specific application or it could be public so that any application can start and use its features [33].

Content Provider is meant for managing access to structured set of data. It provides functionality for setting permissions and access rules to the data. It is the most common pattern for enabling cross application data exchange in Android. Usually content provider is implemented on top of SQLite database [34].

Broadcast Receiver is a component that receives system wide events called Intents. Every application in Android can send out Intents with a special command and data. Each broadcast receiver is registered to receive specific type of intents which makes it possible to listen to those events that are important for particular application [35].

Using these four main components make it possible to implement interactive and rich applications for Android. For each component it is important to follow recommended design patterns and guidelines available in the documentation. It allows to have solid user experience through the whole system and different applications.

7.2 Google Maps

Embedded support for Google Maps is one of the key features of Android that is used by electric motorcycle journey planning application. It provides functionality of visualizing tracks, selecting locations, calculating distances and many more. To make it more specific, it contains three different APIs/libraries that are used in this project:

1. Google Maps Android API v2

   Google Maps Android API v2 is the Android specific API that can be included directly to any Android project as a library. This is used as a visualization layer for the Google Maps. It provides following features for developers [36]:

   - Downloading and displaying the map to the user
   - Finding user's own position with the best possible accuracy
   - Markers - icons on specific location on the map
   - Polylines - lines or tracks on the map
   - Polygons - enclosed selected area on the map
   - Overlays - application specific graphics to be displayed on the map

2. Google Maps API Web Services

   Google Maps API Web Services represents a collection of interfaces that can be accessed over HTTP for acquiring geographical data for the visualization layer. These services use different web URLs for different requests. Each query requires certain URL parameters as input data. Output data will be returned either in XML or JSON format. Google Maps Web Services has the following features [37]:

- Directions - calculates the directions between locations and displays these on the map
- Distance - calculates distance between locations
- Elevation - returns elevation for any point on the Earth
- Geocoding - converts address to coordinates
- Timezones - returns timezone for any coordinates
- Places - gives a list of places and POIs (Points of Interest) near any location

3. Google Maps Android API Utility Library

   Google Maps Android API Utility Library is an extension to Google Maps Android API v2 that is being developed separately from the main library. It is still produced and maintained by Google. It is needs to be added to Android project as an external library. It provides following features [38]:

   - Heatmaps - an alternative for markers, where colored shape shows distribution of the data
   - Bubble icons - API for customizing snippets of information and bubbles shown on the map
   - Cluster manager - for combining multiple markers depending on the zoom level
   - Polyline utils - for encoding and decoding the format of the polyline sent by Web Services
   - Spherical utils - for calculating distances and areas between coordinates

7.3 Android USB Accessory API

Android USB Accessory API exists since Android version 3.1 when the native USB accessory support was added. Android offers programming interfaces that can be used for communicating with USB devices. To make sure that particular Android device is capable to connect with USB accessories at all, it is needed to specify it as an requirement in the application manifest. This makes sure that application cannot be installed on device without USB support [23].

Another component that is needed to specify in the manifest, is the device manufacturer and model. Android API provides a possibility that particular application will be started up right after Android device was connected with matching USB accessory [23]. This is very useful for electric motorcycle as well. The most important part of the API is UsbManager that implements access to actual USB devices. Connecting to USB devices is multiple step process [39]:

1. `UsbManager.getAccessoryList()` returns list of connected accessories.
2. `UsbManager.requestPermission()` request permission for accessing the particular accessory.
3. `UsbManager.openAccessory()` opens actual connection with the accessory after permission has been granted.

After those steps have been completed, application will have references to instances of Java input and output streams that hold connection with USB accessory.

8 Control System for Electric Vehicles

As a part of the present theses, an Android application was designed and implemented. This application is developed considering requirements for electric motorcycle but it can also be used for any electric vehicle with CAN bus interface. As source code will be public, anyone will be able to make necessary modifications for having best fit for their vehicle.

The purpose of the whole system is to provide complete user interface for an electric vehicle. It covers two separate sections: control system and journey planning with navigation. The role of the control system is to visualize multiple important vehicle parameters to the user. It includes battery, motor and many other parameters like speed and distance. As there are more data available that could fit on the screen, it is up to user to decide which parameters are shown.

This chapter analyzes and describes the application's control system and its implementation in more detail. Each subsection of the chapter is about one particular feature set of the application.

8.1 Collecting Data

To visualize important data related with dynamics of electric vehicle to the user, it is needed to transfer data from different parts of the vehicle into the Android device. For electric motorcycle single CAN bus network is used for that purpose. CAN connects all parts together that need to exchange information.
On electric motorcycle there are three kind of components connected with CAN:

1. Motor controllers - 2 pieces. Both controllers broadcast data about motors dynamics.
2. Battery packs - 8 pieces. Each battery pack broadcasts information about battery parameters in particular pack.
3. Android device - listens/requests data from the network and uses it for visualizing purpose.

As CAN does not limit the amount of devices in the same network, this approach allows adding new sensors, devices and battery packs on the run. It means that battery packs can be swapped without any changes required to the system. It is also possible to drive with only some of the battery packs. Any additional sensors or other input devices can be connected with the same network.

8.1.1 Battery Parameters

Battery is probably the most important component in any electric vehicle. It is also one of the most expensive parts which could be permanently damaged by incorrect usage. Therefore it is required to have ability of monitoring different battery parameters.

Battery of an electric vehicle usually consists of multiple battery packs where each of the packs contains multiple cells. Amount of battery packs and cells in each pack defines voltage and capacity of the battery. Each of the battery packs provides BMS (Battery Management System) that manages each of the cells in the particular pack. The purpose of the BMS is to protect every cell from operating outside its recommended voltage, current and temperature limits. More sophisticated BMS systems even provide cell voltage balancing to allow all cells empty equally. BMS also manages charging by transmitting proper voltage and current values to the charger and making sure that all cells are charged equally [40]. Each battery pack is connected to vehicle's central CAN bus interface allowing BMS to transmit all its readings via CAN.

The list of parameters that are transmitted over CAN is seen in Table 2. Each request gives a response from every battery back.

Table 2: Battery related data.

Parameter	**Parameter ID**	**Number of bytes returned**	**Data format**	**Units**
Voltage	0x01 0xD1	1	A	V
Energy	0x01 0xD2	2	((A*256)+B)/4	Wh
State of charge	0x01 0xD3	1	A*100/255	%
Temperature	0x01 0xD4	1	A-40	°C

In Table 2, A denotes the first byte, whereas B denotes the second byte.

These values allow application in Android device to display voltage, remaining amount of energy, state of charge and temperatures of the battery packs.

8.1.2 Controller Parameters

The motor of the electric vehicle is always operated by sophisticated driver system called controller. The controller is responsible of delivering electric power to the motor depending on various configuration parameters and different sensor inputs. Controller is also connected to vehicle central CAN bus network and constantly broadcasts different parameter values to the network. Some electric vehicles might have multiple motors and even multiple controllers. In that case all controllers are broadcasting their data to the same CAN allowing Android device to monitor every motor separately. In addition, as controller behaves as a linkage between battery and motor, it also enables monitoring multiple battery related properties.

The data which are transmitted to Android device for use in dashboard application, can be seen in Table 3.

Table 3: Motor/controller related data.

Parameter	**Parameter ID**	**Number of bytes returned**	**Data format**	**Units**
Motor speed	0x01 0x0C	2	((A*256)+B)/4	RPM
Total voltage	0x01 0xA1	2	A+B	V
Current	0x01 0xA2	1	A*3	A
Motor temperature	0x01 0xC8	1	A-40	°C
Controller temperature	0x01 0xC6	1	A-40	°C

8.2 Battery State of Charge

Battery state of charge (SOC), called also the fuel gauge of electric vehicle, is the level of energy available in the battery compared to its maximum charge. The units of SOC are percentage points where 0% stands for completely empty and 100% completely full. As each battery pack with its BMS in responsible for keeping track of its own state of charge, it is not directly a task for Android application. However, it is one of the most important parameters of electric vehicles and therefore this section gives an overview of the challenge of calculating SOC.

Very often, especially with lead-acid batteries, battery voltage is used as an only input for calculating SOC. This technique is called the voltage translation. It bases on a fact that battery voltage decreases more or less linearly as the battery is discharged. As lead-acid batteries have significant drop in voltage when battery is discharged (Figure 11), voltage can be translated into SOC by knowing the relation [41].

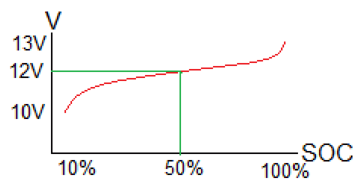


Figure 11: Relation between voltage and SOC of lead-acid battery [41].

However, voltage of lithium batteries remains very constant for large area of its SOC making voltage translation not accurate enough (Figure 12).

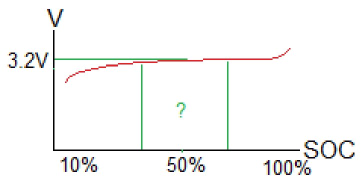


Figure 12: Relation between voltage and SOC of lithium batteries [41].

There is significant drop of voltage when battery is completely discharged and raise of voltage when battery is fully charged which makes the voltage translation practical for only estimating SOC of full or empty lithium battery. For rest of the range of SOC some other technique needs to be used [41].

Coulomb counting is a method where the current is measured in addition to voltage. It means that BMS of the battery is constantly monitoring current entering and current leaving the battery. By doing that it is possible to keep a state of current amount of energy in the battery. It is considered one of the most accurate method for finding SOC of lithium batteries [42]. As there can be a small drift between calculated and actual state of charge, additional calibration is required. Voltage of the lithium battery raises significantly when battery is completely charged and due to that it is possible to calibrate SOC level by measuring voltage and detecting full charge [42].

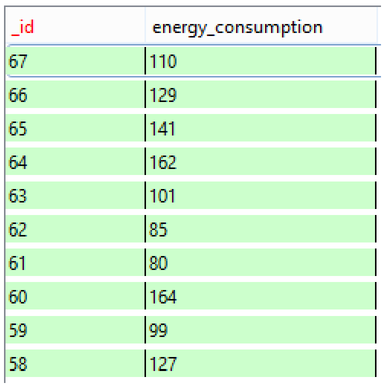
8.3 Driving Range

As charging batteries of an electric vehicle can be more time consuming than filling up a petrol tank and quite often its range is much less than the range of vehicles with combustion engine, it is useful to know how far this vehicle can travel with current state of charge. For that reason most of the electric vehicles constantly calculate remaining range and display it to the user. To do that it is first required to have data of energy consumption. Rather than using a constant user-set value, this application collects its own history of energy consumption.

For calculating the value of energy consumption, average power (kW) is measured during driving. Time spent driving for each kilometer is also required. After that energy consumption Wh/km can be calculated using formula (1). For example, driving one kilometer with average power of 10kW within 1 minute means energy consumption of 167 Wh/km: $10kW \times \frac{1}{60}h = 0.167kWh = 167Wh$.

$$Energy = Power \times Time \tag{1}$$

For storing data of energy consumption, Android built-in SQLite database is used [43]. Structure of the table can be seen on Figure 13. New row is added after driving every kilometer.



_id	energy_consumption
67	110
66	129
65	141
64	162
63	101
62	85
61	80
60	164
59	99
58	127

Figure 13: History of energy consumption (Wh/km).

Having the history of energy consumption is the only way for making sure that the range display is updated depending on the style of driving and other circumstances such as speed or traffic. As application is already aware of current state of charge of the battery as described in the previous section, calculating the estimated range is fairly straightforward. Average energy consumption of last 10 kilometers is used for calculations. The current amount of energy in the battery is divided by energy consumption. For example, 6 kWh (6000 Wh) of energy in the battery and average consumption of 120 Wh/km is: $\frac{6000Wh}{120Wh/km} = 50km$.

8.4 Speed and Distance

The speed and distance are important parameters for any kind of vehicle. It is required to achieve good accuracy to make the vehicle possible to drive in traffic. Showing the proper speed has been a challenge even for traditional car and motorcycle manufacturers due to different tyre sizes and possible changes in gear ratio. The present control system needs to be universal for different vehicles and that adds additional complexity.

As described in the Section 8.1 Android device has knowledge of how many revolutions per minute (RPM) electric motor is performing. To calculate the speed, gear ratio and perimeter of the wheel are also required. Most electric vehicles are direct drive or single geared so there is no need for considering multiple gears. Android application provides settings where the user can enter his vehicle specific data about the gear ratio and perimeter of the wheel. Direct drive motors have gear ratio of one.

Android `PreferenceFragment` [44] is a special tool for offering custom settings to the user. It allows to define different settings in XML file and handles storing entered values in Shared preferences [43]. `PreferenceFragment` takes care of creating the user interface and storing values entered by user. Those values can then be used by any other part of the application.

Android application is requesting the electric motorcycle controller system to broadcast motor speed 50 times per second over CAN. Every time Android calculates vehicle speed based on how many revolutions motor is performing using formula (2).

$$V = \frac{60 \times S \times P}{R}, \tag{2}$$

where V is the speed of vehicle, S is RPM of the motor, R is the gear ratio and P is the drive wheel perimeter.

For instance, the calculation with direct drive motor performing 700 RPM and drive wheel with perimeter of 170 cm equals: $\frac{(60\times700)rev/h\times(170\div1000\div100)km/rev}{1} = 71.4km/h.$

With every new value of calculated speed, also time stamp of the speed is stored in the memory. That allows calculating the distance covered between two measurements of speed. For that reason application keeps history of 100 last calculated speeds. It then calculates the average speed and calculates the distance based on time stamps. The distance is manually stored using Android Shared Preferences [43] to make it persistent.

The last speed is always displayed to the user when the control system view of the application is opened. Displaying distance is optional and can be switched to some different parameter.

8.5 Power

The power in the given context represents electrical power at which electric energy is transferred from the battery. This is not equal to the actual power output of the motorcycle due to the loss of energy in controller, wires and transmission.

Every controller of the electric motorcycle is broadcasting its voltage and current. From this data it is possible to calculate power of every controller using the following formula:

$$P = I \times V, \tag{3}$$

where P is the power in watts, I is the current in amperes and V is the voltage in volts.

The powers of every controller is summed to calculate the total electric power of the motorcycle. It is important to notice that power of any amount of controllers can also have negative value. This happens when energy is transferred from motor to the battery. This process is called regenerative braking and is used to transfer kinetic energy back to the battery instead of heating brake discs.

8.6 Conclusion

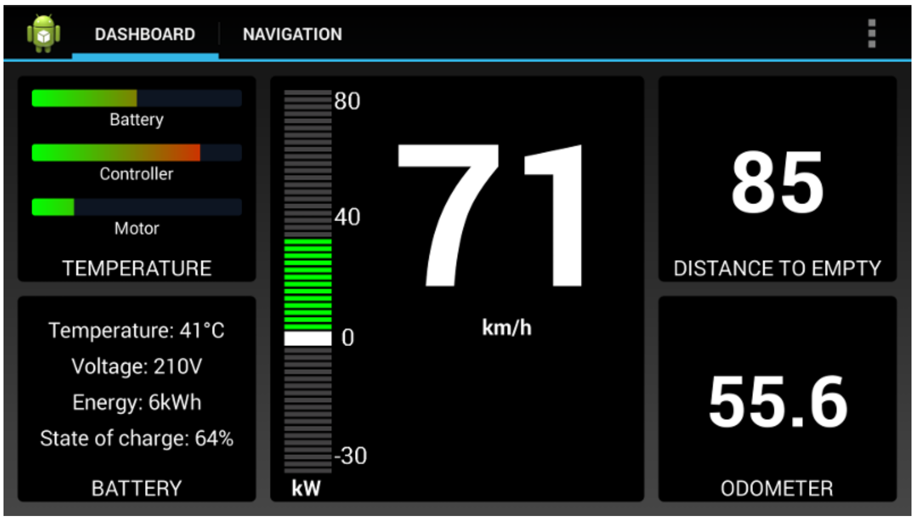
The control system was implemented as a part of the Android application. It contains four configurable windows which are all displayed on the screen and where each window can contain data from one of the following items:

- **'Temperature'** window visualizes the temperature values to the user. It shows battery, motor and controller temperatures. By default the temperature scale for each indicator is from 0 to 100°C.
- **'Battery'** window shows some most important battery parameters. These are temperature, voltage, state of charge and amount of energy in the battery.
- **'Distance to empty'** window shows driving range. It is amount of kilometers that can be driven with current state of charge.
- **'Odometer'** window shows total distance covered by the vehicle in kilometers.
- **'Time'** window shows the current time.

Application is implemented so that adding new window configurations is as easy as possible. As application's source code will be open it is important that others can modify it to fulfill their needs.

In addition to four configurable windows, dashboard has a center part of the screen that always displays the speed and power. The speed is displayed as numeric value in km/h and the power is displayed on scale from -30 to 80 kW. Minimum and maximum values of the scale are configurable.

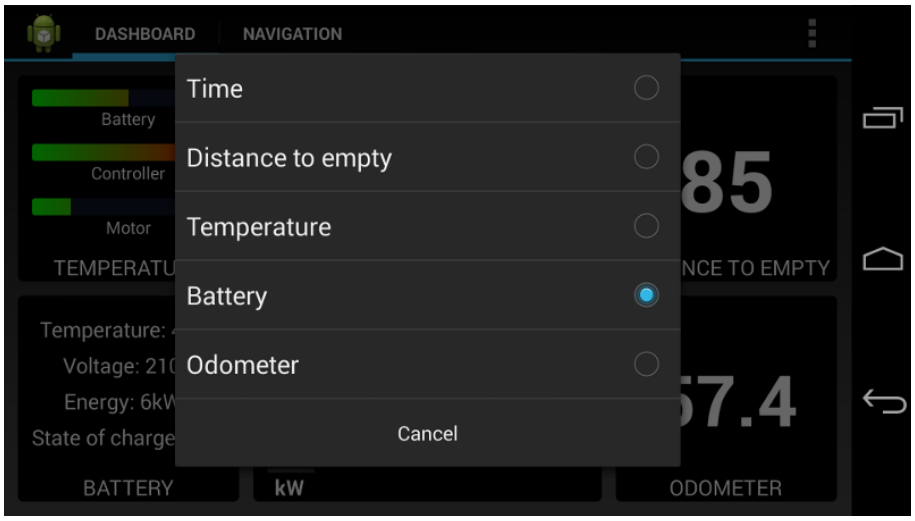
Figure 14 illustrates how dashboard looks on Android device. Configurable windows are in each corner and central part fixed for speed and power.



Temperature: 41°C
Voltage: 210V
Energy: 6kWh
State of charge: 64%

Figure 14: Control system.

When user taps on any of the configurable windows, a popup dialog is opened which allows configuring. Figure 15 illustrates that dialog.



Time ○
Distance to empty ○
Temperature ○
Battery ●
Odometer ○

Figure 15: Dialog for customizing control system.

Selected window configurations are stored in Android class `SharedPreferences` [43] to make sure that the configuration persist if Android device is rebooted.

9 Journey Planning and Navigation System

Second part of the Android application is called navigation. The purpose of the navigation system is to provide the complete map support. It visualizes charging points and navigation to them. It also provides complete directions to any point on the map.

This chapter analyzes the navigation part and technical details of the solution. Each subsection of the chapter describes one particular feature set of the application.

9.1 ELMO Charging Stations

ELMO (Estonian Electromobility Programme) is the national programme in Estonia with the purpose to popularize and promote emission-free personal transportation and electric vehicles. The program was started in year 2011 by trading 10 million units of AAU (Assigned Amount Units) of allowance to emit greenhouse gases in return of 507 Mitsubishi iMiev electric cars and cross-country infrastructure for charging these vehicles. Estonia became the first country in the world with nationwide electric car charging network in 2012 when the grid was completed [45].

Today ELMO manages the network of charging stations. It is possible to see Google Maps based map on their website (`http://elmo.ee/charging-network/`) with all charging stations. Status of the chargers is updated in real time: whether the charger is available, busy or out of order. More importantly, the same data are also available for using in any other application. Data are available in KML format, on the url: `http://klient.elmo.ee/download/file/ELMO.kml`. This file contains charging stations' descriptions, addresses, location coordinates, names and statuses. The file is updated in real time.

9.1.1 Reading and Parsing Chargers' Data

KML (Keyhole Markup Language) is used to display the geographical data in Google Earth and Google Maps. It is based on XML (Extensible Markup Language) with tag-based structure [46].
One sample of charging station described in KML language can be seen on Figure 16.

```
<Placemark>
    <name>37002</name>
    <styleUrl>#local_kml_default_style_map</styleUrl>
    <MultiGeometry>
        <Point>
            <tessellate>1</tessellate>
            <coordinates>24.544358,59.430168,0</coordinates>
        </Point>
    </MultiGeometry>
    <description>Tabasalu Rimi</description>
    <address>Klooga maantee, Tabasalu alevik</address>
</Placemark>
```

Figure 16: Tabasalu Rimi charging station in KML.

Although KML is natively made for use with Google Maps, this support is not publicly exposed by Google Maps Android API v2. All markers need to be added manually on the map by using `GoogleMap.addMarker()` function [47]. To do that, KML file needs to be parsed first. To avoid recurrent parsing of the same file, KML is parsed right after downloading and then stored in a more convenient way.

As KML is very similar to XML, Android built-in XML parser can be used. There are multiple choices of XML parsers available, but `XmlPullParser` [48] is recommended by Android documentations as an efficient way for this task. To start the parsing it is required to select which tags need to be read [49]. In this context it is required to read `name`, `description`, `styleUrl`, `coordinates` and address tags for every `placemark`, as seen on Figure 16. How `XmlPullParser` implementation looks like for reading these tags, can be seen on Figure 17. Function `readPlacemark()` as seen on the figure, will be called in loop while iterating through the content of KML file.

```
private Placemark readPlacemark(XmlPullParser parser) {
    parser.require(XmlPullParser.START_TAG, null, "Placemark");
    Placemark placemark = new Placemark();
    while (parser.next() != XmlPullParser.END_TAG)  {
        String tag = parser.getName();
        parser.next();
        if (tag.equals("name")) {
            placemark.setName(parser.getText());
        } else if (tag.equals("description")) {
            placemark.setDescription(parser.getText());
        } else if (tag.equals("address")) {
            placemark.setAddress(parser.getText());
        } else if (tag.equals("coordinates")) {
            placemark.setCoordinates(parser.getText());
        } else if (tag.equals("styleUrl")) {
            placemark.setStyleUrl(parser.getText());
        }
    }
    parser.require(XmlPullParser.END_TAG, null, "Placemark");
    return placemark;
}
```

Figure 17: KML parser using XmlPullParser [48].

As a result of parsing there will be an array of `Placemark` objects in the memory of Android device. This array of data can easily be manipulated further. It can be used for displaying charger stations on the map or for storing this data persistently in the Android device storage.

9.1.2 Storing Chargers' Data

To ensure the application works in the offline mode, the latest state of charging points stored in the device is always needed. Android API provides multiple different options for saving data persistently:

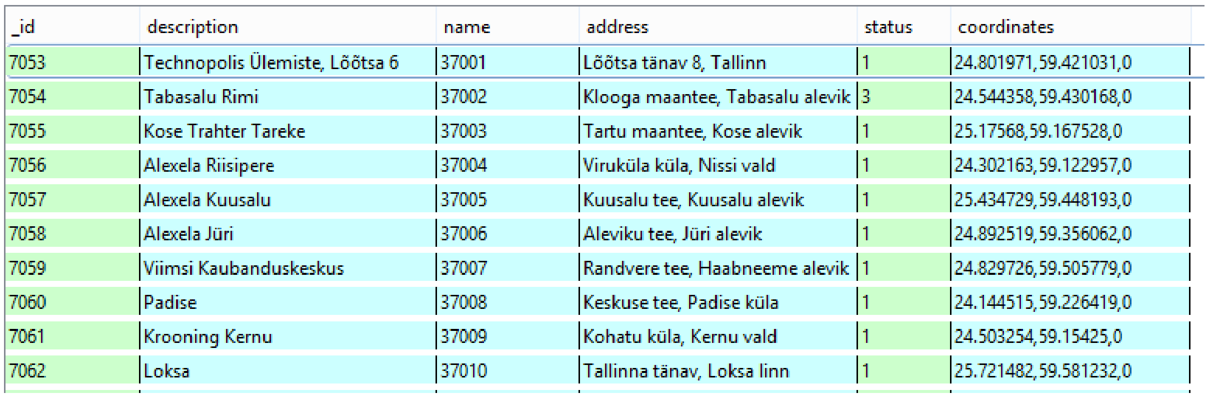
- **Shared preferences.** Key-value pairs of primitive data types. Values are written to a file in application private data directory. The framework takes care of all input-output operations and also caching data which allows synchronous access to data [43].

- **File storage.** The data files can be stored either in private data directory of the application or in any public folder. Separate directory is provided for cache files that could be cleaned by users. All input-output operations need to be taken care by the programmer on byte-level basis. Therefore asynchronous access must be implemented [43].

- **SQLite databases.** Android provides full support for SQLite databases which is one of the most used database engine in the world. Its API provides many helper functions for queries, inserts or any other operations. By default SQL database is private but it is possible to allow other applications gain access to the data [43].

For storing data of ELMO charging points, SQLite database is definitely the best selection among these options. In Android there are few different possible SQLite implementations. In this project the `ContentProvider` implementation [34] is used.

`ContentProvider` is meant for managing access to the data stored in SQLite database. It is the most used pattern for enabling cross-application data exchange. This gives possibility for other applications to use the same data of charging points without the need of KML parsing and their own persistent storage. It is possible to define permissions which allow only selected applications to access the data. `ContentProvider` has been written on top of Android default SQLite implementation `SQLiteOpenHelper` [34]. Figure 18 shows the structure of the database table that holds ELMO charging stations' information.



_id	description	name	address	status	coordinates
7053	Technopolis Ülemiste, Lõõtsa 6	37001	Lõõtsa tänav 8, Tallinn	1	24.801971,59.421031,0
7054	Tabasalu Rimi	37002	Klooga maantee, Tabasalu alevik	3	24.544358,59.430168,0
7055	Kose Trahter Tareke	37003	Tartu maantee, Kose alevik	1	25.17568,59.167528,0
7056	Alexela Riisipere	37004	Viruküla küla, Nissi vald	1	24.302163,59.122957,0
7057	Alexela Kuusalu	37005	Kuusalu tee, Kuusalu alevik	1	25.434729,59.448193,0
7058	Alexela Jüri	37006	Aleviku tee, Jüri alevik	1	24.892519,59.356062,0
7059	Viimsi Kaubanduskeskus	37007	Randvere tee, Haabneeme alevik	1	24.829726,59.505779,0
7060	Padise	37008	Keskuse tee, Padise küla	1	24.144515,59.226419,0
7061	Krooning Kernu	37009	Kohatu küla, Kernu vald	1	24.503254,59.15425,0
7062	Loksa	37010	Tallinna tänav, Loksa linn	1	25.721482,59.581232,0

Figure 18: Information of chargers stored in Android.

For accessing the data, `ContentResolver` needs to be used. It provides functions like `insert()`, `delete()`, `update()` and `query()` which are used for different database operations. It is important to note that all database operations need to be done asynchronously to avoid any blocking delays in the main thread of the device [34]. Current status of the charger is stored as an integer for easier comparison, where 1 means available, 2 means out of order and 3 means that someone else is using it at the moment. Figure 19 shows how `ContentResolver` is used to update data in the SQLite database.

```
// clear previous data
getContentResolver().delete(CONTENT_URI, null, null);
// insert new data
for (Placemark placemark : placemarks) {
    ContentValues values = new ContentValues();
    values.put(ADDRESS, placemark.getAddress());
    values.put(COORDINATES, placemark.getCoordinates());
    values.put(DESCRIPTION, placemark.getDescription());
    values.put(NAME, placemark.getName());
    values.put(STATUS, placemark.getStatus());
    getContentResolver().insert(CONTENT_URI, values);
}
```

Figure 19: Storing array of placemarks in ContentProvider.

As a result of running this code, all ELMO chargers are persistently stored in the database format as seen on the Figure 18.

9.1.3 Displaying Chargers on the Map

Android Google Maps API v2 provides a class called `MapFragment`. It is the most convenient way for showing Google Map in any application. `Fragment` is an Android user interface module that specifies its own layout and can be used for various dynamic user interface implementations [50]. `MapFragment` is an implementation of `Fragment` and can therefore be combined with any other user interface elements in Android [36].

The main view of the electric motorcycle application contains two fragments: Dashboard and Navigation. The latter is a `MapFragment` with some modifications. `MapFragment` provides access to `GoogleMap` instance via `getMap()` function, which enables to modify map visible to the user [47]. Figure 20 shows how `GoogleMap.addMarker()` function is used for displaying charging stations on the map.

```
for (Placemark placemark : placemarks) {
    getMap().addMarker(new MarkerOptions()
        .position(placemark.getPosition())
        .title(placemark.getName())
        .snippet(placemark.getAddress() + ", " + placemark.getDescription())
        .icon(placemark.getIcon());
}
```

Figure 20: Displaying array of placemarks on the map.

After this code has been run, `MapFragment` does the rest and ELMO charging points are shown on the map. Figure 21 shows how that `MapFragment` looks like. One of the chargers is occupied (yellow), others are available.

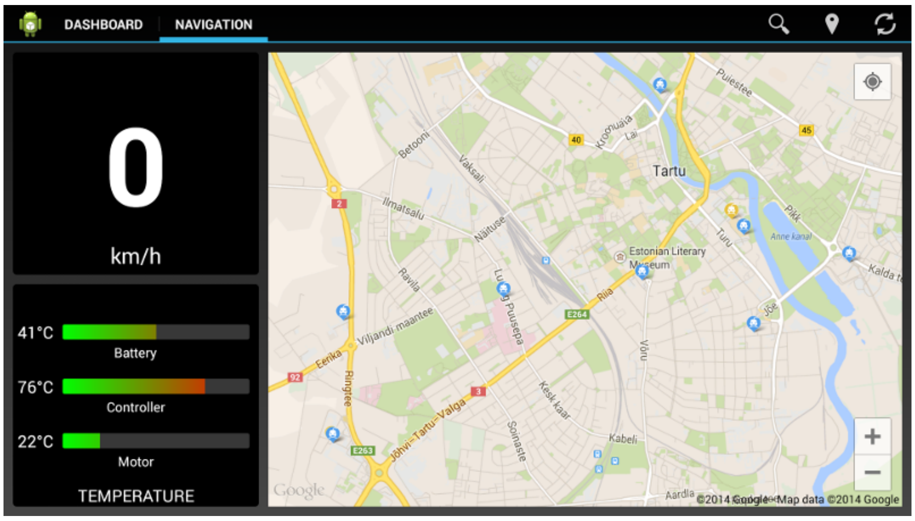


Figure 21: Charging stations shown on the map.

Clicking on a marker of charger shows small popup with name of the charger, address and description. Application also offers an option for the user to get navigation directions to the selected charging station.

9.1.4 Displaying Nearby Chargers

The application allows to see nearby charging stations as a list. List is sorted based on the distance from user's current location, starting from the lowest. Each row shows description and address of the charging station and distance from user's location. Distance is calculated using coordinates which means that actual driving distances may vary. The main reason for this kind of distance calculation is complexity of the driving distance calculations while the distance over the air still provides useful information about the distance to the particular charging station. Calculation of the driving distance would require online query while coordinate based calculation can be done offline and even synchronously in the main thread of the application. Android provides a function `Location.distanceBetween(latitudeFrom, longitudeFrom, latitudeTo, longitudeTo, result)` which does the exact job using Haversine's formula [51].

9.2 Directions

Another important part of the navigation application is visualizing route directions to the user. It is possible to get driving directions to any point on the map or to any specific address. It is also possible to take directions to any charging point, whether on the map or in the list of nearby chargers. For all directions, user's own location is used as the origin. For getting the directions, Google Directions API is used which is a part of Google Maps API Web Services.

9.2.1 Making Direction Requests

Simple HTTP request is used for accessing the API. All query parameters are added in URL encoded format. The parameters include [52]:

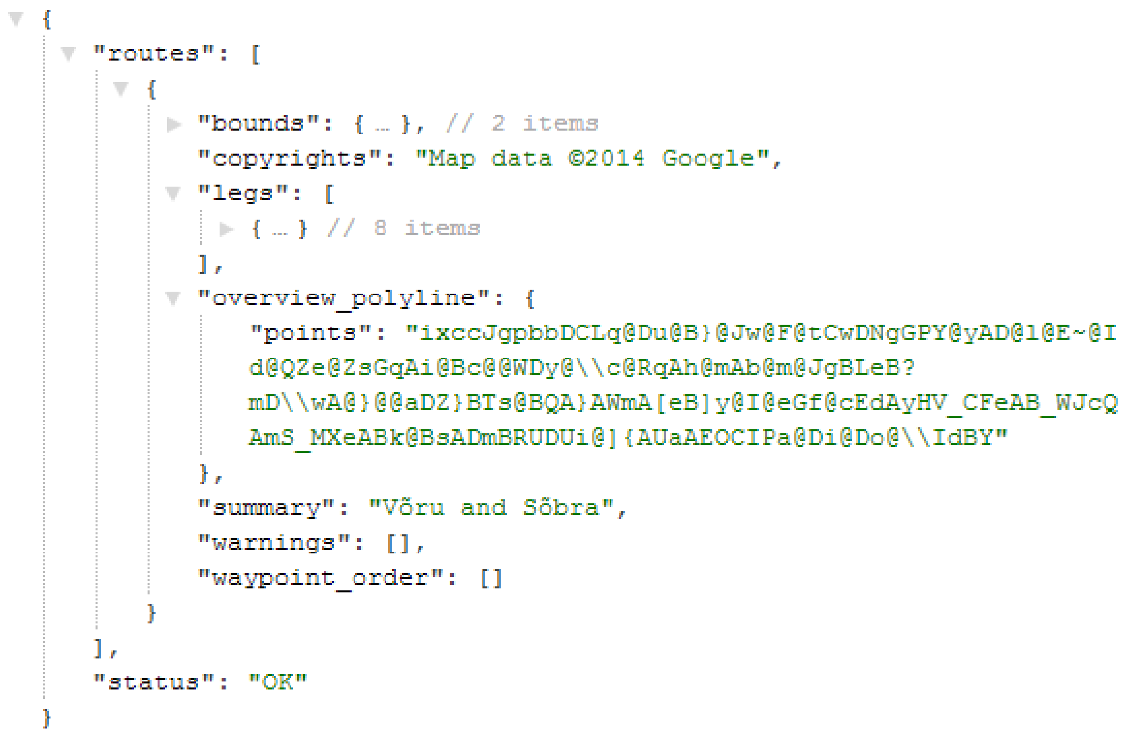
- origin - user's current location coordinates
- destination - desired destination of the route; it can be either address or coordinates
- units - value will be set depending on Android device current locale; usually it has value "metric"
- mode - "driving" is used for all queries
- sensor - set to "true", meaning that request comes from a device with location sensor

Output of the query can be requested either in XML or JSON format. For this application JSON is used. JSON (JavaScript Object Notation) is lightweight data format, mostly used for transferring data in web applications. It supports collections of name-value pairs and lists of values [53]. Android has built in org.json [53] JSON library which makes parsing JSON output trivial.

One possible sample of directions request made by journey planning application looks like: `https://maps.googleapis.com/maps/api/directions/json?origin=58.3415277%2C26.7341194&destination=58.36805400279372%2C26.737820841372013&sensor=true&mode=driving&units=metric`

9.2.2 Directions API Responses

Response of the sample request can seen on Figure 22. Bounds and legs have been collapsed.



```
{
  "routes": [
    {
      "bounds": { … }, // 2 items
      "copyrights": "Map data ©2014 Google",
      "legs": [
        { … } // 8 items
      ],
      "overview_polyline": {
        "points": "ixccJgpbbDCLq@Du@B}@Jw@F@tCwDNgGPY@yAD@l@E~@I
        d@QZe@ZsGqAi@Bc@@WDy@\\c@RqAh@mAb@m@JgBLeB?
        mD\\wA@}@@aDZ}BTs@BQA}AWmA[eB]y@I@eGf@cEdAyHV_CFeAB_WJcQ
        AmS_MXeABk@BsADmBRUDUi@]{AUaAEOCIPa@Di@Do@\\IdBY"
      },
      "summary": "Võru and Sõbra",
      "warnings": [],
      "waypoint_order": []
    }
  ],
  "status": "OK"
}
```

Figure 22: Directions request output.

The main container of the response is ”routes” which contains possible routes. Unless alternative routes were specifically requested, the list of routes always contains one item. Each route have the following items [52]:

- `’bounds’` - corner coordinates for the viewport.

- `'legs'` - list of legs of the route. For every leg there are distance, duration, start address, end address and description of every step that needs to be taken on that leg.
- `'summary'` - short textual description of the route, which is good for differencing one route from alternatives.
- `'waypoint_order'` - list of waypoints if they have been reordered for optimization.
- `'overview_polyline'` - encoded array of points that represent approximate route, what can be used for displaying the route on the map.

9.2.3 Showing Route on the Map

For showing route on the map, `overview_polyline` from the Google Maps Directions API is used. `Overview_polyline` is encoded data in Base64 format (see Figure 22), that represents starting coordinate of the route and every other point of the route as an offset from previous point. Base64 is a common way for encoding binary data to String format for transmitting over the network [54]. The data need to be decoded for showing on the map. For that purpose, Google Maps Android API Utility Library provides a function `PolyUtil.decode()`. This function takes encoded path as an input and returns array of latitude-longitude coordinates [38]. Array of coordinates can then be displayed on the map using code seen on Figure 23.

```
PolylineOptions options = new PolylineOptions().color(Color.BLUE);
for (LatLng point : PolyUtil.decode(encodedPolyline)) {
    options.add(point);
}
getMap().addPolyline(options);
```

Figure 23: Adding polyline to GoogleMap.

Together with directions, the distance of the route needs to be calculated as well. It would be possible to calculate the distance from the points of polyline but for better accuracy it is recommended to use distance values provided in the Directions API output. Each leg of the Directions response contains length of that leg in meters. The values from each leg are summed up to gain the total distance of the route. The total value is then formatted and shown on the top left corner of the screen.

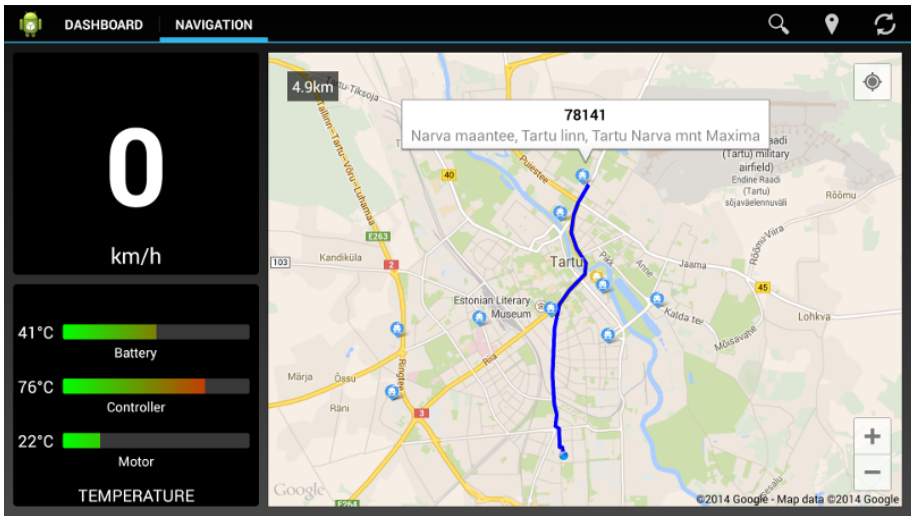


Figure 24: Directions and distance shown on the map.

Figure 24 shows how directions polyline and distance of the route appear on Google Maps in electric motorcycle journey planning application.

9.3 Dynamic Directions

One of the biggest issues with modern navigation applications is that they are not optimized for use with electric vehicles. They might have data about charging points but they do not necessarily use it for finding the best route. Electric motorcycle journey planning application was implemented to overcome that issue. The idea of dynamic directions is to use the estimated driving range and location data of charging stations to find the best route which allows actually reaching the destination even if the total distance might be longer. Figure 25 describes the implementation of this feature.

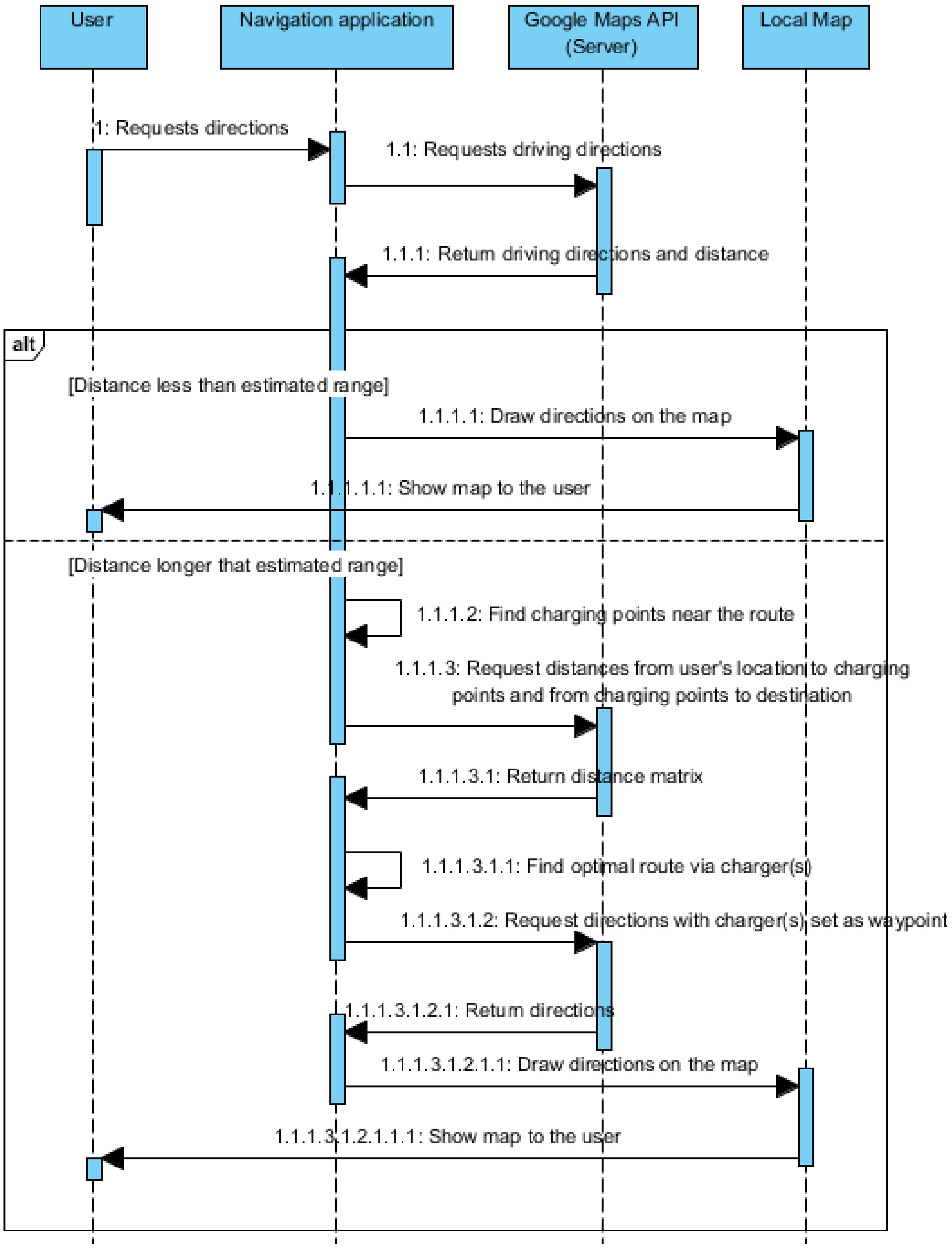

User
Navigation application
Google Maps API (Server)
Local Map
1: Requests directions
1.1: Requests driving directions
1.1.1: Return driving directions and distance
alt
[Distance less than estimated range]
1.1.1.1: Draw directions on the map
1.1.1.1.1: Show map to the user
[Distance longer that estimated range]
1.1.1.2: Find charging points near the route
1.1.1.3: Request distances from user's location to charging points and from charging points to destination
1.1.1.3.1: Return distance matrix
1.1.1.3.1.1: Find optimal route via charger(s)
1.1.1.3.1.2: Request directions with charger(s) set as waypoint
1.1.1.3.1.2.1: Return directions
1.1.1.3.1.2.1.1: Draw directions on the map
1.1.1.3.1.2.1.1.1: Show map to the user

Figure 25: Sequence diagram of dynamic directions.

Sequence diagram gives an overview about how Google Maps Web API is accessed and used to determine the route. In addition to directions requests described in the previous section, dynamic directions require use of Google Maps Distance Matrix service. The distance Matrix API provides the driving distance between the given origins and destinations. It is possible to pass multiple locations with the same request which results the distance calculated for each pair [55]. The electric motorcycle navigation application finds first all charging points no more than 15 kilometers from the route, calculated by Haversine's formula [51] and then requests the distances matrix between the origin, destination and all charging points.

Using these distance values, the optimal route will be found. The application calculates the distance from origin to destination through as many charging stations as needed so that none of parts of the route exceeds range of the vehicle. First leg of the route is found using current available range of the vehicle. Maximum distance for other legs is calculated from current range and SOC considering 80% as the state of charge after every charging station. If the best path has been chosen, it can request the actual driving directions from the Google Maps API. Selected charging points are used as the optional waypoint values in the directions' request [52].

9.4 Conclusion

The navigating system has been implemented as a part of the electric motorcycle dashboard application. It is the essential part of the electric vehicle to be used in optimal way and for making sure that user finds the best charging station whenever needed. It contains the following features available to the user:

- **Map.** Application shows map based on Google Maps. User's current location is always displayed and updated when user moves. It is possible to browse the map all over the world.

- **Charging stations.** Charging stations are displayed on the map together with address and descriptional data for location. The status of the charging station is also displayed - whether or not charging station is available. It is also possible to see the list of nearest charging stations.

- **Directions.** Application provides the driving directions to any possible destination. It allows to request directions to charging stations, addresses or to any point manually selected on the map. Direction calculations also use data about vehicle range and locations of charging stations to make sure that suitable route is selected for the given electric vehicle.

Different parts of the application have been divided into separate logical parts of the source code. This allows improving or modifying the application with needs of different users. If anyone outside of Estonia wants to use the code, he can just change part of the system, which currently takes care of loading Estonian charging stations and everything else can remain the same.

10 Future Improvements and Ideas

While implementing the present application as described in previous chapters, multiple good ideas were generated how this application could be even more useful for average electric vehicle user. But every software project once reaches the point where the release has to be made and other cool features remain waiting for their time. As author of the thesis plans to continue developing the application it was found appropriate to give a short overview of problems which were addressed during the development but did not find their solution yet.

10.1 GPS Speed Calibration

Currently the user is required to manually find and configure gear ratio and drive wheel perimeter of the vehicle. This makes it possible to calculate speed and distance only using information about how many revolutions per minute the motor is performing. The idea is to develop the GPS calibration which would calculate these parameters based on the actual speed measured by GPS. The user must be able to put the application into calibration mode before doing the measurements. After that the application will be able to tell proper speed without the GPS signal lock.

10.2 Elevation Data

The differences in elevation might have considerable impact on electric vehicle range in more mountainous areas than Estonia. Google Maps API which is already used in the application has possibility to query for elevation profile for any path. This allows to know the exact climbs and falls on the route. The idea is to use this data in conjunction with vehicle's previous history of energy consumption in order to calculate the more accurate range estimation. This would allow better route planning as next charging point can be selected more wisely.

10.3 Time Estimations

While Google Maps API provides estimates about driving time from one point to another, this data could be useless if driver needs to charge the vehicle halfway to the destination. The idea is to combine driving time estimations with estimations on charging time to calculate the total time required to reach the destination. It is already possible to calculate the remaining battery charge level in each of the charging points. The power output of each of the charging stations is also known and charging time can be calculated using characteristics provided by manufacturers. This would also make it possible to find the fastest route depending on charging stations available.

10.4 Battery Statistics

The idea is to collect statistics about how battery of the vehicle is being used. This would allow application to analyze how efficiently battery is being used and might be helpful for increasing the battery life in some cases. Good example would be a user who is constantly using less than 50% of the battery capacity but still charges the vehicle up to 100% of charge level. In that case application could suggest to start charging up to 80% which is considered much more friendly in terms of battery life.

Summary

The purpose of the thesis was to design and develop the complete open source solution to be used as control system and journey planning application for electric vehicles. The thesis is based on project ”Developing Mechanical Solution for Electric Driven Motorcycle” funded by Enterprise Estonia from 2013 to 2014 [1]. This project was also the source of the idea and need for the system. The scope of the thesis includes both hardware and software needed to fully monitor and control the vehicle. As the idea of the whole project is to popularize spread of electric vehicles, open source was the desired approach.

Beginning of the development process started with comparison of existing projects and solutions currently available on the market. It included some other projects with similar goals but also full solutions already integrated with some of the best selling electric vehicles in the world. As there was no existing solution available that would fulfill needs of the given project, it was decided to continue with completely new system.

The next part of the thesis describes the process of choosing development platform of the solution. The advantages and disadvantages of multiple common open source platforms have been collected, analyzed and compared to find the best suitable platform. As the platform sets very strict requirements on both software and hardware that can be used, it was an essential decision which could not be changed after development has started. Today it is clear that decision has been made correctly and the selected platforms have proved themselves.

The fifth chapter of the thesis concentrates on the hardware that is used to make this system work. The actual devices used for building prototype device and also the code running on these devices have been described. The network protocol which is used to connect different devices all over the electric vehicle, found its section in given chapter. As a result of developing hardware solution, two different solutions were implemented. One solution is based on USB connection and another one works over bluetooth. This chapter analyses technical solutions of both.

As most of the outcomes of the thesis are directly related with different software, there is a chapter which analyzes multiple essential programming interfaces used to develop the application. It includes well-known APIs such as Google Maps API or Android API.

The final electric motorcycle application consists of two different logical sections, each offering batch of different features. The next chapter in the thesis covers first of the application feature-sets: dashboard. The purpose of the chapter was to analyze and describe how different features of the dashboard were implemented. It covers the most important technical details as well as purposes behind user interface design. As a result, the working solution has been made which visualizes important data about dynamics of the vehicle to the user which makes it possible to operate given motorcycle in traffic. It also provides some configurable options to the user to allow customizing the application for different needs.

The second part of the final application is the navigation system. The purpose of that chapter was also to analyze and describe technical possibilities and solutions used to implement different features of navigation. As a result, the navigating section of the final application provides multiple features to the user of electric vehicle. It enables user to browse charging stations, to query driving directions and to do many other map based actions.

Finally, multiple ideas that could bring great value to the user but were decided to be out of the scope of given thesis have been presented. Some of them turned out to be very time consuming or complex. As author of the thesis is planning to continue working on the application, some of the problems are going to be addressed in next few months.

Elektrisõidukite kontrollsüsteem ja teekonna planeerija elektrimootorratta näitel

Magistritöö (30 EAP)

Oliver Meus

Resümee

Seoses järjest karmimaks muutuvate keskkonnanõuetega transpordisektoris ning tänu riiklikele toetusprogrammidele ning infrastruktuuri arengule on elektrisõidukid muutumas üha olulisemaks osaks tänapäeva ühiskonnast. Aga hoolimata viimaste aastate kiirest arengust on tänasel päeval keeruline leida erialast kompetentsi ja kogemusi elektrisõidukite arenduse valdkonnast. Antud lõputöö eesmärgiks oli valmistada avatud lähtekoodiga komplektne lahendus, mis viiks sisenemisbarjääri antud valdkonda mitmete iseehitajate ja muidu huviliste jaoks natukene madalamale, võimaldades alguspunktina kasutada olemasolevat tarkvara.

Töö ühe osana on koostatud ülevaade juba varem eksisteerinud analoogilistest lahendustest, nende eelistest ja puudustest. Analüüsitakse nii vabavaralisi süsteeme kui ka lahendusi, mis on kasutusel Euroopa levinumates elektriautodes. Samuti sisaldab töö põhjalikku analüüsi erinevate võimalike platvormide kohta, millel antud süsteem põhineda saaks. Ülevaade antakse ka elektrimootorrattast, mis saab olema esimene praktiline väljund valminud süsteemi kasutamisel.

Töö esimeseks eesmärgiks oli disainida ja välja arendada elektrisõiduki tehniliste parameetrite monitoorimise rakendus koos vajamineva riistvaraprototüübiga. See tähendab, et juht peab saama reaalajas jälgida sõiduki juhtimiseks olulisi väärtusi nagu näiteks kiirus, mootori võimsus, aku laetuse tase, akude temperatuur jpm. Selle jaoks on valmis tehtud kaks erinevat riistvaraprototüüpi, mis võimaldavad antud tarkvara käitava Android operatsioonisüsteemil põhineva seadme ühendamist mootorrattal oleva lokaalse standardiseeritud võrguga. Samuti on välja arendatud tarkvara, mis neid andmeid kuvab. Töös kirjeldatakse mõlema prototüübi ja kogu tarkvara tehnilisi lahendusi, analüüsitakse

erinevaid otsuseid ning tehakse kokkuvõte tulemustest.

Töö teises etapis valmis elektrisõiduki tehniliste parameetrite monitoorimise rakenduse lisana elektrisõidukitele omaseid eripärasid arvestav teekonna planeerimis- ning navigeerimissüsteem. Süsteem pakub kasutajale mitmeid kaardiga seotud võimalusi. Nendest olulisemad on üleriigilisse laadimisvõrgustikku kuuluvate kiirlaadijate kuvamine juhile ning sõidujuhiste pakkumine erinevatesse sihtkohtadesse. Sõidujuhiste leidmisel arvestatakse ka laadimisjaamade asukohti ning konkreetse sõiduki läbisõitu ühe laadimiskorraga. Sarnaselt ülejäänud süsteemiga on ka selles osas põhjalikult analüüsitud erinevaid tehnilisi lahendusi, kirjeldatud implementatsiooni ning esitatud ülevaade saavutatud tulemustest.

Kuna töö autor plaanib jätkata töötamist antud süsteemi ja lahenduse edasiarendamise suunas, tehakse töö viimases osas kokkuvõte mitmetest ideedest, mis antud lõputöö skoobist välja jäid. Välja on toodud mitmed lisavõimalused, mis rakenduse kasutajale kindlasti lisaväärtust looksid. Ent tegu on suure keerukusega probleemidega, mis vajavad eraldi analüüsi ja implementatsiooni. Seetõttu leiavad nad eraldi käsitlemist läheneva paari kuu jooksul.

Kokkuvõtteks võib öelda, et antud lõputöö autor on lahendanud endale algselt püsitatud probleemid. Kõik töö tulemused tehakse avalikuks ning neid on võimalik kasutada mitmesuguste projektide elluviimiseks.

Bibliography

[1] EAS, Toetatud projektide andmebaas. http://www.eas.ee/et/eas/sihtasutusest/toetatud-projektid/toetatud-projektid-alates-2004a-aprill?page=0&pageitems=25&display=1&company_name=digitigu&project_year=&county=&schema= Referred 30.04.2014

[2] Prismatic Cell, A123 Systems. http://www.a123systems.com/prismatic-cell-amp20.htm Referred 2.04.2014

[3] EnerTrac motors. http://www.enertrac.net/product.php Referred 4.04.2014

[4] Abalta Technologies, EOLAS-EV. http://www.abaltatech.com/index.php?/site/solution/eolas-ev and http://www.abaltatech.com/images/uploads/Brochure_EolasEV2Wheel_1012.pdf Referred 19.01.2014

[5] Automakers consider polygons on map to show electric car range. http://www.plugincars.com/automakers-consider-polygons-map-show-electric-car-range-127248.html Referred 19.01.2014

[6] EMotorWorks homepage and Android Dashboard technical documentation. http://www.emotorwerks.com/tech/modular and http://www.emotorwerks.com/code/EMW_EV_Dash/EMW_EV_Android_DashBoard-Base_Edition_V1.2-User_Guide.pdf Referred 20.01.2014

[7] Tumanako open source projects homepage. http://sourceforge.net/apps/mediawiki/tumanako/index.php?title=Main_Page and http://sourceforge.net/apps/mediawiki/tumanako/index.php?title=Dashboard Referred 22.01.2014

[8] Nissan Leaf global sales reach 100000 units. http://nissannews.com/en-US/nissan/usa/releases/nissan-leaf-global-sales-reach-100-000-units Referred 16.02.2014

[9] Nissan Leaf Navigation system owner's manual. http://www.nissan-techinfo.com/View.ashx?d=1&z=1&sku=2013-Nissan-LEAF-Navi Referred 16.02.2014

[10] Model S, Tesla Motors. http://www.teslamotors.com/models Referred 16.02.2014

[11] Tesla Model S Owner's manual. http://www.teslamotorsclub.com/attachment.php?attachmentid=30905&d=1379533778 Referred 16.02.2014

[12] Wikipedia, Arduino. http://en.wikipedia.org/wiki/Arduino Referred 23.01.2014

[13] Raspberry Pi foundation, FAQs, http://www.raspberrypi.org/faqs Referred 25.01.2014

[14] Open Handset Alliance, Android Overview. http://www.openhandsetalliance.com/android_overview.html Referred 25.01.2014

[15] Android Compatibility Definition. http://static.googleusercontent.com/media/source.android.com/en//compatibility/android-cdd.pdf Referred 25.01.2014

[16] Android Open Accessory Protocol. http://source.android.com/accessories/protocol.html Referred 25.01.2014

[17] Apps - Google Play. https://play.google.com/intl/en-US_us/about/apps/index.html Referred 26.01.2014

[18] Xperia Z Ultra specifications. http://www.sonymobile.com/global-en/products/phones/xperia-z-ultra/specifications/ Referred 4.04.2014

[19] Texas Instruments, Introduction to CAN. http://www.ti.com/lit/an/sloa101a/sloa101a.pdf Referred 25.01.2014

[20] Renesas, Introduction to CAN. http://documentation.renesas.com/doc/products/mpumcu/apn/rej05b0804_m16cap.pdf Referred 25.01.2014

[21] CANopen basics - Introduction. http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml Referred 12.02.2014

[22] CANopen basics - Communication. http://www.canopensolutions.com/english/about_canopen/communication.shtml Referred 12.02.2014

[23] USB Accessory — Android. http://developer.android.com/guide/topics/connectivity/usb/accessory.html Referred 10.06.2014

[24] Android Open Accessory Protocol. http://source.android.com/accessories/aoa2.html Referred 12.03.2014

[25] Android Accessory development firmware and examples. https://dl-ssl.google.com/android/adk/adk_release_20120606.zip Referred 11.03.2014

[26] Arduino Mega Board 2560. http://arduino.cc/en/Main/arduinoBoardMega2560 Referref 11.03.2014

[27] Max3421E USB Host controller. http://www.maximintegrated.com/datasheet/index.mvp/id/3639 Referred 11.03.2014

[28] CAN bus shield. http://www.seeedstudio.com/wiki/CAN-BUS_Shield Referred 11.03.2014

[29] ELM327 AT Commands. https://www.sparkfun.com/datasheets/Widgets/ELM327_AT_Commands.pdf Referred 8.04.2014

[30] Bluetooth — Android. http://developer.android.com/guide/topics/connectivity/bluetooth.html Referred 8.04.2014

[31] Application fundamentals — Android. http://developer.android.com/guide/components/fundamentals.html Referred 13.03.2014

[32] Activities — Android. http://developer.android.com/guide/components/activities.html Referred 14.03.2014

[33] Services — Android. http://developer.android.com/guide/components/services.html Referred 14.04.2014

[34] Content Providers — Android. http://developer.android.com/guide/topics/providers/content-providers.html Referred 21.02.2014

[35] BroadCastReceiver — Android. http://developer.android.com/reference/android/content/BroadcastReceiver.html Referred 14.04.2014

[36] Google Maps Android API v2 documentation. https://developers.google.com/maps/documentation/android/intro Referred 11.02.2014

[37] Google Maps API Web Services documentation. https://developers.google.com/maps/documentation/webservices/ Referred 12.02.2014

[38] Google Maps Android API Utility Library. https://developers.google.com/maps/documentation/android/utility/ Referred 26.02.2014

[39] UsbManager — Android. http://developer.android.com/reference/android/hardware/usb/UsbManager.html Referred 13.03.2014

[40] Wikipedia, Battery management system. http://en.wikipedia.org/wiki/Battery_management_system Referred 2.03.2014

[41] Estimating the state of charge of Li-ion batteries. http://liionbms.com/php/wp_soc_estimate.php Referred 6.03.2014

[42] Battery State of Charge Determination. http://www.mpoweruk.com/soc.htm Referred 6.03.2014

[43] Storage options — Android. http://developer.android.com/guide/topics/data/data-storage.html Referred 21.02.2014

[44] PreferenceFragment — Android. http://developer.android.com/reference/android/preference/PreferenceFragment.html Referred 17.04.2014

[45] ELMO - About. http://elmo.ee/about/ Referred 18.02.2014

[46] KML Turorial. https://developers.google.com/kml/documentation/kml_tut Referred 18.02.2014

[47] GoogleMap — Android. https://developer.android.com/reference/com/google/android/gms/maps/GoogleMap.html Referred 20.02.2014

[48] XmlPullParser — Android. http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html Referred 20.02.2014

[49] Parsing XML data — Android. http://developer.android.com/training/basics/network-ops/xml.html Referred 19.02.2014

[50] Building a Dynamic UI with Fragments — Android. https://developer.android.com/training/basics/fragments/index.html Referred 22.02.2014

[51] Location — Android. http://developer.android.com/reference/android/location/Location.html Referred 26.02.2014

[52] The Google Directions API. https://developers.google.com/maps/documentation/directions/ Referred 22.02.2014

[53] JSON http://json.org/ Referred 23.02.2014

[54] Encoded Polyline Algorithm. https://developers.google.com/maps/documentation/utilities/polylinealgorithm Referred 26.02.2014

[55] Google Maps Distance Matrix API. https://developers.google.com/maps/documentation/distancematrix/ Referred in 30.03.2014

Appendix

Source code

Source code of the solution provided in the thesis can be downloaded from the GitHub:

`https://github.com/oliver6/EV_dashboard`

Abbreviations

AAU	Assigned Amount Units
ADB	Android Debug Bridge
API	Application Programming Interface
BLDC	Brushless Direct Current
BMS	Battery Monitoring System
CAN	Controller Area Network
ELMO	Estonian Electromobility Programme
EV	Electric Vehicle
GPS	Global Positioning System
IDE	Integrated Development Environment
ISO	International Standardization Organization
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LTE	Long-Term Evolution
PDO	Process Data Object
POI	Point of Interest
RPM	Revolutions Per Minute
SDO	Service Data Object
SOC	State of Charge
USB	Universal Serial Bus
XML	Extensible Markup Language

License

Non-exclusive license to reproduce thesis and make thesis public