Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3118

Development of a User Interface for Electric Cars

Thomas Walter

Course of Study:	Engineering Cybernetics
Examiner:	PD Dr. rer. nat. Michael Schanz
Supervisor:	Prof. Dr. rer. nat. habil. Thomas Bräunl
Commenced:	June 3, 2010
Completed:	December 3, 2010
CR-Classification:	J.2, J.7

Abstract

In the current climate debate, electric vehicles are widely considered to be the future of individual transport. The REV Project of the University of Western Australia has converted three regular petrol driven cars to electric vehicles. This thesis focusses on the user interface for one of those vehicles, taking into account the special requirements of electric cars.

The thesis begins by analysing existing solutions and evaluating the important features of those systems (Chapters 2 and 3). In addition, special requirements for test driving a prototype car are defined. These features are essential for the REV Project to evaluate the performance of the cars.
It then describes the general ideas behind the development of the program and details their implementation (Chapters 4 and 5). This includes communication with the battery management system, processing of GPS data, calculation of statistics, generation of a map and control of the multimedia functionality.
Lastly, the thesis explains the resulting program (Chapter 6), discusses experiences in using it and presents data collected during test drives (Chapter 7).

Acknowledgements

During my work on this thesis, I received substantial support from many people who I would like to mention here.

First and foremost, I would like to thank my supervisor Prof. Thomas Bräunl for his invitation to the University of Western Australia and for his support and guidance during the last six months.

Many thanks also go to Dr. Michael Schanz for his great support before and during my stay in Perth.

Appreciation also extends to Linda Barbour for her organizational support, Ken Fogden for help with the installation of the hardware and Ivan Neubronner for help with the battery management system.

Furthermore, I would like to thank all members of the REV team for their work and dedication to the project and for the interesting discussions.

Special thanks to Jonathan Wan for his assistance in the REV lab and for showing me so much of Perth.

Many thanks also go to Jerome for providing the French translation, to Sharron for proof-reading and to Christopher, Jiang and Jeremy for the great time we had together.

Finally, I would like to thank my girlfriend and my family for all of their support and encouragement during the last months.

Contents

List of Figures

List of Tables

List of Listings

Nomenclature

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BMS	Battery Management System
CRC	Cyclic Redundancy Check
CSV	Comma Separated Values
FM	Frequency Modulation
GPL	General Public License
GPS	Global Positioning System
GUI	Graphical User Interface
INI	Initialization File Format
LCD	Liquid Crystal Display
NMEA	National Marine Electronics Association
PC	Personal Computer
REV	Renewable Energy Vehicle
USB	Universal Serial Bus
UTC	Universal Time Coordinated
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1. Introduction

Climate change is no longer a topic only for environmentalists but is now widely discussed among the general public. In 2009, the leaders of the world's largest countries met in Copenhagen to discuss ways to slow down or halt global warming. There is a general consensus that climate change is due to anthropogenic greenhouse gas emissions, in particular the emissions of carbon dioxide. According to [Her09], road transportation contributed 13.6% of the world's carbon dioxide emissions in 2005 and is therefore one of the major causes of this phenomenon.
Consequently, the reduction of vehicle emissions is crucial in the development of new cars. Reduction in fuel consumption is an important first step in achieving this goal but any long term solution will require zero-emission vehicles. One way to realise cars without any emissions is to change from combustion engines to electric drives. These battery powered cars can take their energy from any source of electricity, including all kinds of renewable sources, thus allowing a seamless substitution of fossil fuels.
Electric cars are not only able to use energy from regenerative sources, they are also generally more energy efficient. This is mainly due to the limited capacity of the batteries which enforces a more conservative use of energy. Another aspect is the higher efficiency of electric motors compared with combustion engines. Whereas the latter only convert up to 50% of the chemical energy into movement, even at their optimal operating point, electric motors can have an efficiency of more than 90% across a wide range of speeds.
The following sections provide a short introduction to the Renewable Energy Vehicle Project at the University of Western Australia and to automotive instrumentation in general. The chapter finishes with an outline of this thesis.

1.1. The REV Project

The Renewable Energy Vehicle (REV) Project in its current form was started by the University of Western Australia in 2008 to research and demonstrate the capabilities of electric vehicles. Staff and students from the School of Electrical, Electronic and Computer Engineering (EECE) and the School of Mechanical and Chemical Engineering are working together on three different vehicles for this project.
The first car, named REV Eco, aims to fulfil the needs of commuter vehicles. It is a five seater, based on a Hyundai Getz 2008 model and equipped with a 39 kW electric motor. This car is

intended to be a small, comfortable and inexpensive car for everyday usage.
The REV Racer, which is the second car of the project, is an electric performance car. It is a two seater, based on a Lotus Elise S2 2002 model and has a 54 kW motor. It is more powerful than the first car and meant to demonstrate the ability of electric cars to outperform vehicles with combustion engines. The current project involved redesigning the instrumentation for that car.
The latest vehicle of the REV Project is a Formula SAE [for10] race car. This vehicle is a single-seater designed to participate in the Formula SAE electric league. In contrast to the other two cars, it is a pure motor sports vehicle and is therefore not street-legal.

Figure 1.1.: The REV Racer, based on a Lotus Elise S2 (photo by Jonathan Wan)

1.2. Automotive Instrumentation

Within the last two decades, driver information systems have become an increasingly important component in the design of new cars. They are more powerful, more complex and more diverse today than ever before. Twenty years ago, analogue radio and cassette players were state of the art and automotive instrumentation focussed mainly on the speed of the car and the temperature of the water coolant. Today, the entertainment system of a car often features CD, DVD and MP3 playback, and has Bluetooth connection and a navigation system. The driver can access information about tire pressure and ambient temperature, can monitor fuel consumption, and is warned about service intervals and unbuckled seatbelts. With the transition to electric cars, this development will progress even further. In these cars, the driver needs to be notified about the distance remaining, batteries need accurate monitoring, and power consumption is essential information.

1.3. Structure

The following chapter outlines some existing solutions for automotive instrumentation and analyses their differences and similarities. The results of this analysis are given in chapter 3 and are the basis for this work.

Chapter 4 begins with a description of the hard- and software that was used for the implementation of the program and then explains some of its basic characteristics.

In chapter 5, the implementation of the backend modules is described. It is follow by chapter 6 where the user interface of the program is explained.

The experiences in using the program and the results of the first test drives are outlined in chapter 7. In chapter 8, a conclusion is given, along with suggestions for future work.

2. Systems Survey

This chapter provides an overview of the existing work of the REV team and other solutions for electric cars. Here, the features of each system and the ways in which they provide the driver with information are summarised.

2.1. REV Eco

The REV Eco, based on a Hyundai Getz, is the first car of the REV Project. It uses an Eyebot [eye08], which has a touchscreen with a 128x64 resolution and an ARM9 processor. The Eyebot runs on BusyBox Linux with the RoBIOS libraries from [rob06].

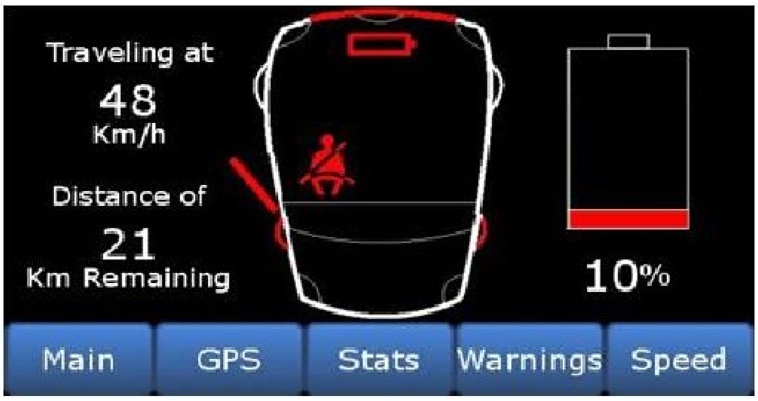


Figure 2.1.: The main screen of the REV Project Hyundai Getz [Var09]

On its main screen (figure 2.1), it displays basic information such as speed, battery level and an estimation of the remaining distance. Additionally, the driver is warned about open doors and low battery level and is reminded to fasten the seat belt. The other screens are used to display the postion of the car (figure 2.2), usage statistics and additional warnings. The last screen enables the driver to set up acoustic warnings when a certain speed is exceeded.

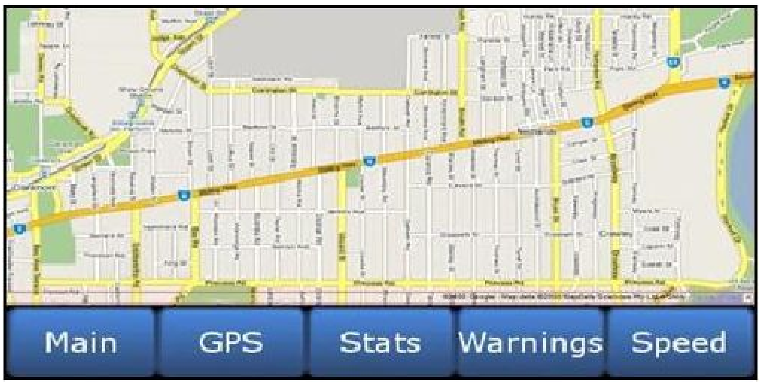


Figure 2.2.: The map view of the REV Project Hyundai Getz [Var09]

2.2. REV Racer

The Lotus Elise is the second car that was retrofitted for electric drive as part of the REV Project. The driver information system described in this thesis is designed to run on this car. The computer is more powerful than in the REV Eco and the screen has a higher resolution. More information about the hardware is given in chapter 4.1.

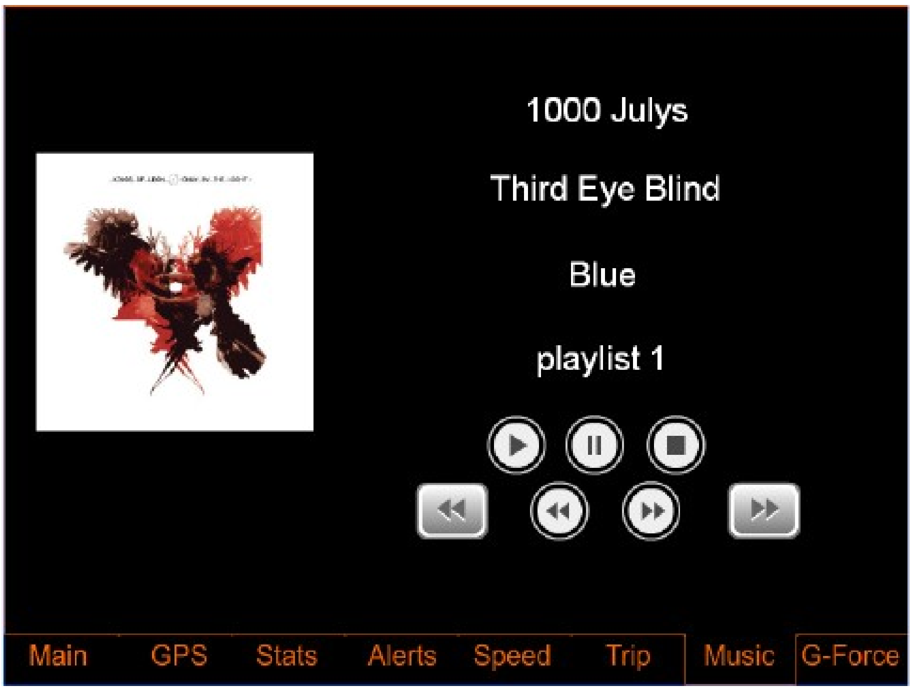


Figure 2.3.: Music player of the REV Project Lotus Elise [Var09]

Compared to the Getz, the original software has three additional tabs. Its more powerful hardware enables the integration of a music player as pictured in figure 2.3. Two other features are focussed more strongly on test driving; one tab displays statistics about the current trip, while the other tab shows the lateral and longitudinal acceleration.

2.3. Tesla Roadster

The Tesla Roadster has many similarities with the REV Lotus as it is also a pure electric sports car with only two seats and a focus on lightweight construction. Its driver information system (figure 2.4) is split into three parts: a big screen for entertainment and navigation, a small screen for information about the battery and the car, and the instruments cluster behind the steering wheel to display speed, current and other important values.

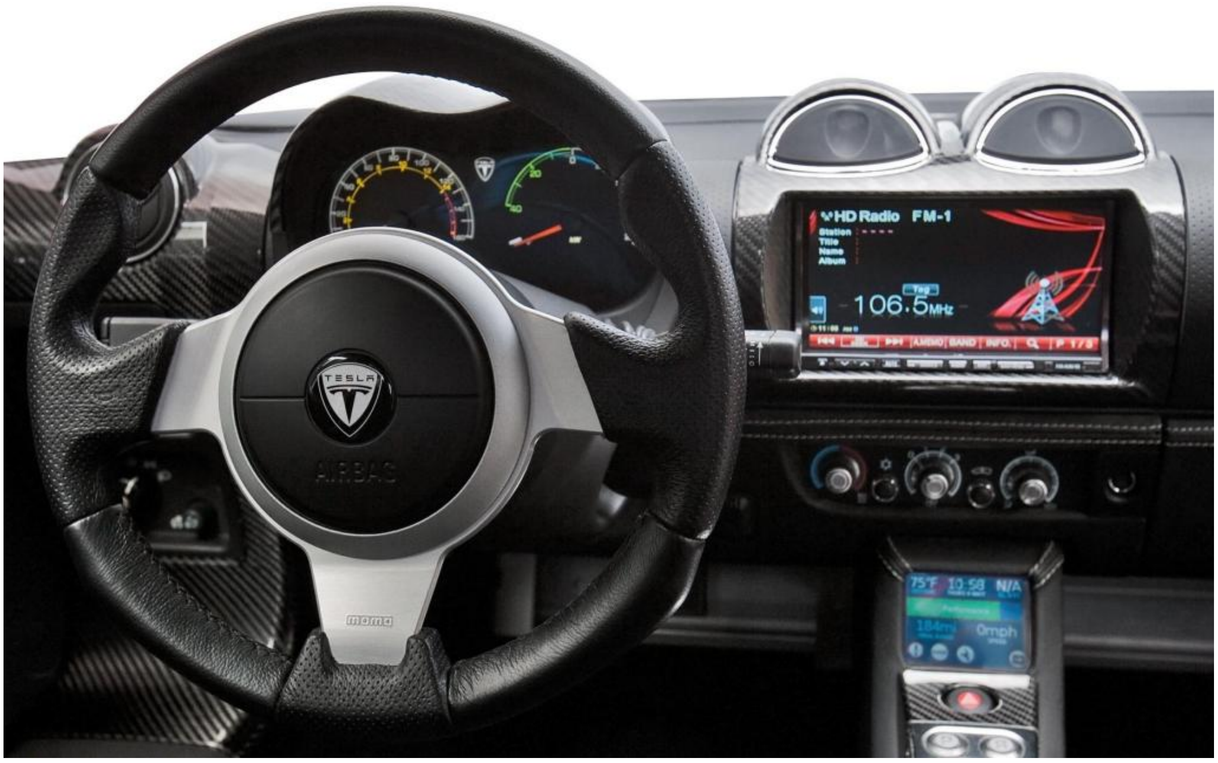


Figure 2.4.: Tesla Roadster 2.5 instrumentation [tes10]

2.4. Tesla Model S

“Model S“ is the second model developed by Tesla Motors and is expected to be on the market in 2012. Compared to the Roadster, it is larger and places a stronger emphasis on comfort. The instruments cluster behind the steering wheel has been replaced by a display to show speed and battery status and the two screens in the Roadster have been replaced by one big screen (figure 2.5). This screen not only gives access to the entertainment functions and the navigation, but also controls most devices in the car, including air conditioning, ventilation, seat heating and interior lighting.

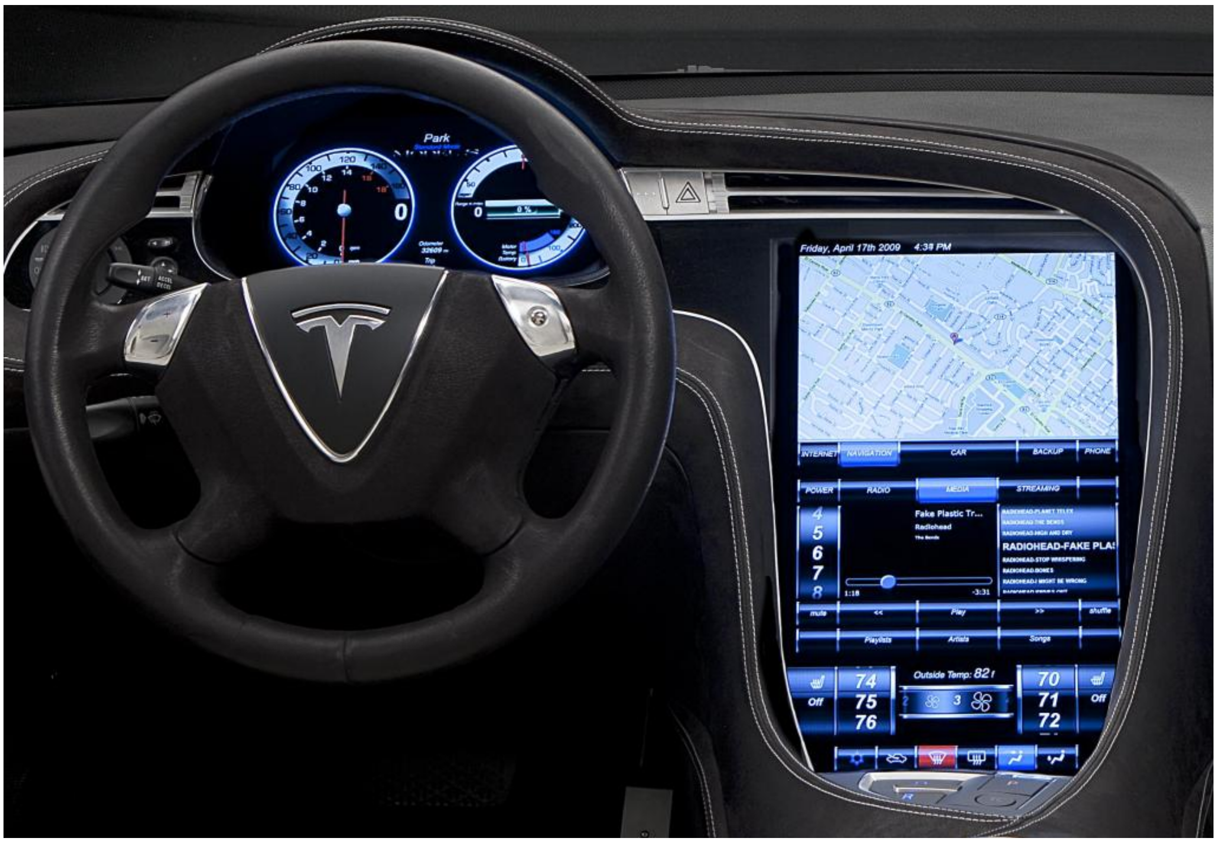


Figure 2.5.: Tesla Model S instrumentation [tes10]

2.5. BMW Mini E

The driver information system of the Mini E is completely different to all the other cars analysed in this chapter. Instead of having screens to display the status of the car, the Mini E is equipped with analogue indicators. Figure 2.6 shows the indicator behind the steering wheel that displays the battery level. The main instrument between driver and front passenger displays the current speed and controls the music playback (figure 2.7). The red lights signal power consumption; when the car is regenerating power the lights turn green.

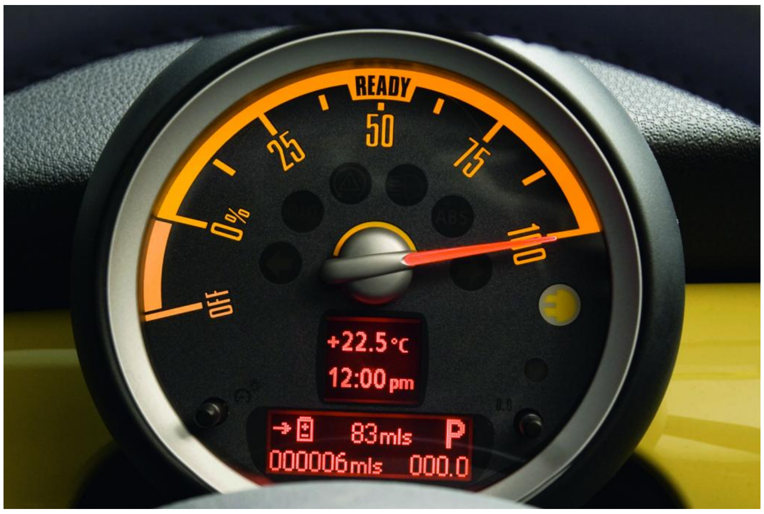


Figure 2.6.: Mini E battery indicator [bmw10]

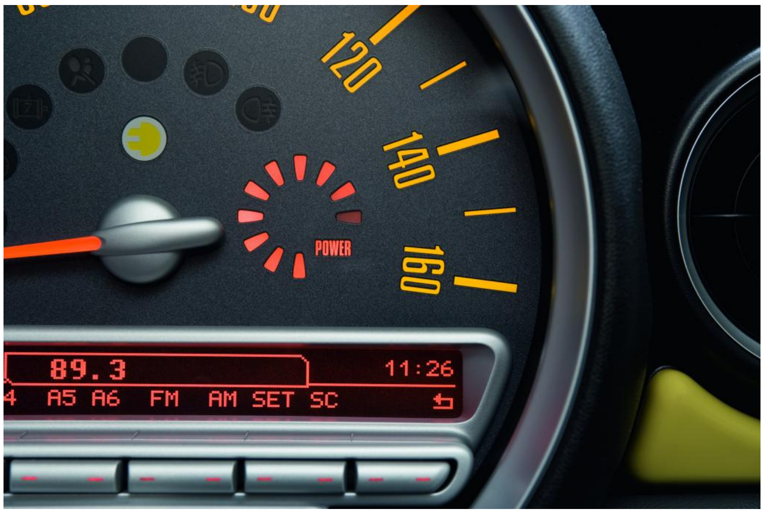


Figure 2.7.: Mini E central instrument [bmw10]

2.6. Mercedes-Benz SLS AMG E-Cell

The SLS AMG E-Cell is a Mercedes-Benz prototype car. Similar to the other cars, it has a touchscreen in the centre console (figure 2.9) that is used for navigation, entertainment and battery management. The battery status, as well as speed and warnings, are displayed in the instruments cluster (figure 2.8).

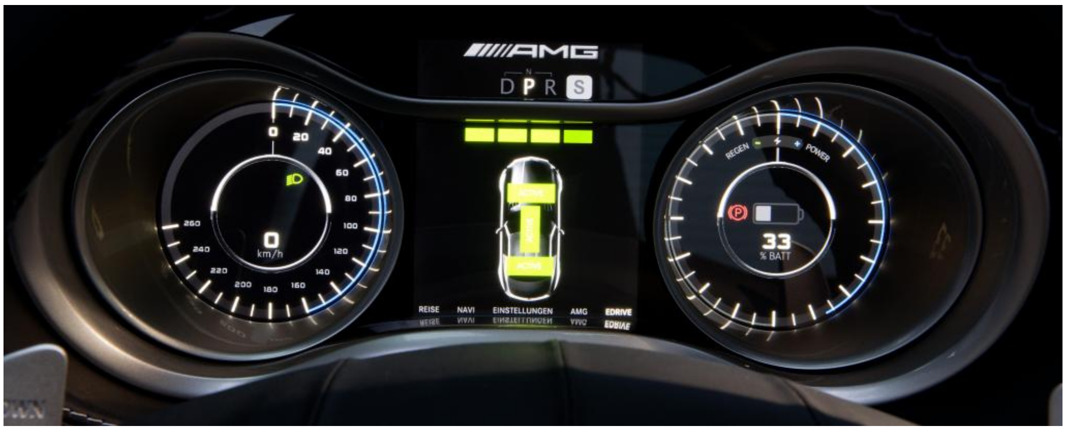


Figure 2.8.: Mercedes Benz SLS AMG E-Cell instruments cluster [dai10]

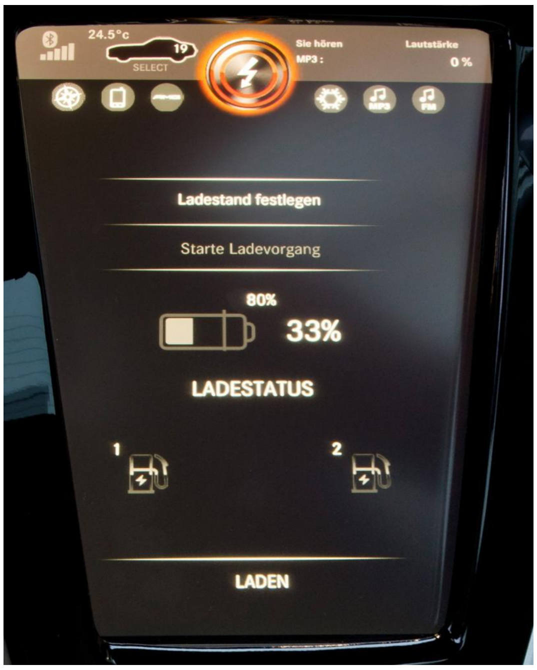


Figure 2.9.: Mercedes Benz SLS AMG E-Cell centre console screen [dai10]

2.7. Chevrolet Volt

For the Chevrolet Volt, the instruments cluster has been replaced by a screen. As the car is still in development, the design of the user interface is not yet completed and the pictures on the Chevrolet homepage [che10] show different versions of the software (figures 2.10 and 2.11).

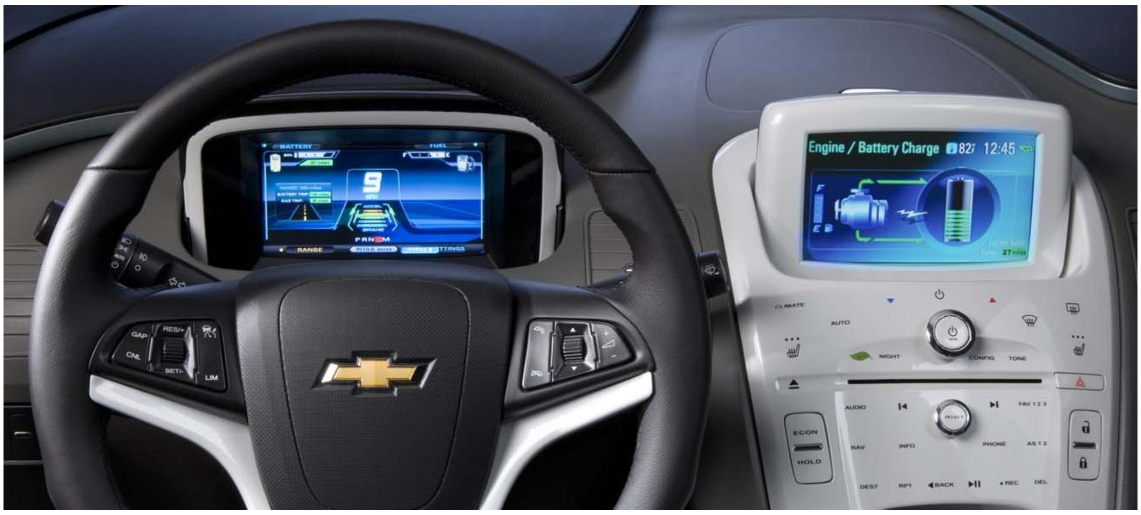


Figure 2.10.: Chevrolet Volt instrumentation [che10]

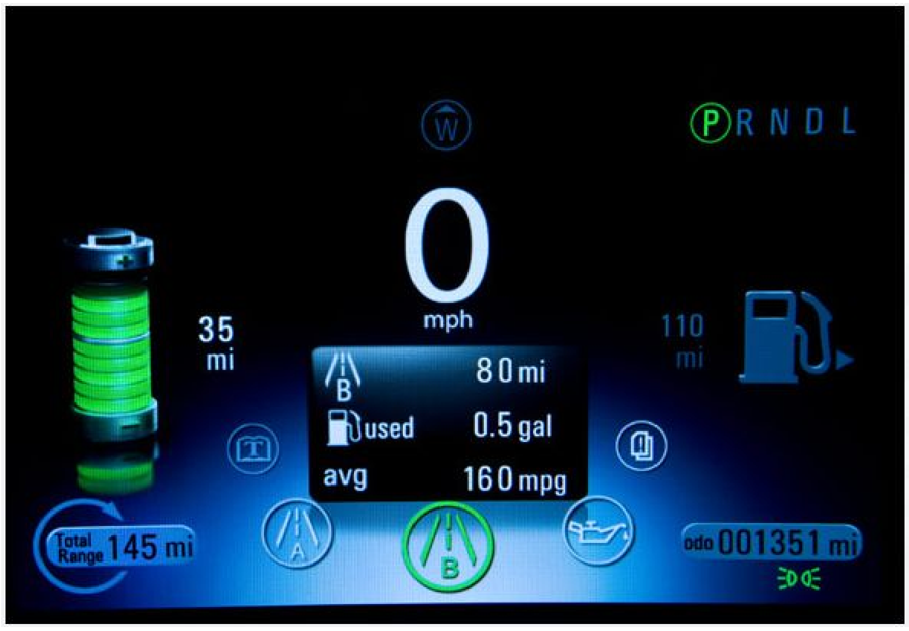


Figure 2.11.: Chevrolet Volt instruments cluster [che10]

3. Requirements

Review of existing systems in chapter 2 highlighted the features that a driver information and battery management system should have. This chapter summarises these functionalities and lists additional requirements that are derived from the experiences in the REV Project. The requirements listed in this chapter were the basis for the user interface that has been developed for the current project.

3.1. Basic Information

Most systems have some basic features in common which are essential for every driver. These features are independent of the type of car and can also be found in older vehicles.

Speed The speed of the car needs to be displayed at all times. Indeed, in most countries this is a legal requirement and the accuracy must be within a defined range.

Distance All cars show the overall distance the car has covered since manufacturing, which is important when reselling the vehicle.

Time Most cars also display the current time to eliminate the need for the driver to look at his watch.

Warnings It is very important to inform the driver of possible safety issues, such as open seat belts or open doors. Problems with the car also need to be displayed because they can not only damage the vehicle, but also cause dangerous situations.

3.2. Battery Management

Unlike the original Lotus, the REV car needs a system to monitor the battery and to display this information to the driver. As the battery is the most valuable and most sensitive component of an electric car, it needs special care.

Voltage The overall voltage needs to be within certain limits at all times. If the battery is discharged beyond its minimum level, this can cause irreparable damage. The same applies for voltages that are too high during charging or regenerative braking.

Capacity The capacity of the battery displays how much energy is remaining. It is equivalent to the fuel gauge of a car with a combustion engine.

Distance Remaining The remaining distance is an estimation based on the capacity of the battery and the electric consumption of the car.

Current The display of the current informs the driver of the current energy consumption of the car or, with regenerative braking, how much energy is charged back to the battery. This can help the driver to adopt a more energy efficient driving behaviour.

Cell Status It is important to monitor the status of every cell to detect failing cells as early as possible. Cells that are damaged must be replaced immediately as they reduce the performance of the car and are potentially dangerous.

3.3. Entertainment

For many drivers it is essential to have music while driving. In the Lotus, the screen of the driver information system is placed in the previous position of the radio; therefore, it also needs to be able to play back music.

Music Player The program needs to be able to play music files from the hard disk. Managing the playlist is important for users with a larger music collection.

Radio Local radio is not only entertainment, but also broadcasts safety warnings and traffic information. Support for FM radio requires additional hardware and was therefore not implemented; however, if the car has internet connection, it should be able to play back radio web streams.

3.4. Navigation

An integrated navigation system eliminates the need for an additional device in the car. For the REV team, navigation is especially important because many team members are not familiar with the Perth area and therefore need assistance during test drives.

Car Position Position and travelling direction of the car should be displayed on the map to enable the driver to localise himself/herself.

Route Loading the test route into the map makes it easier for the test driver to follow.

Directions In an advanced version, turn-by-turn navigation could be realised. As the focus of the REV Project is on the electric drive, this feature was not included.

3.5. Testing

Testing a prototype car requires additional features that are usually not part of the driver information system of a stock car.

Data Logging Most important is the logging of all relevant data in the car. This allows an evaluation and comparison of the different test drives and is essential for creating statistics when doing long-term evaluations.

Statistics Some of the data collected during the test drive should be immediately available to the driver. This instant feedback eliminates the need for an additional computer or laptop.

Server communication To speed up the evaluation, the current status of the car should be streamed to a server. This gives the team the possibility to evaluate the performance of the car live and enables all team members to easily access the data from the test drives.

Debug output For tracing errors in the car or in the software, it is important to log all warnings, errors and debug messages. In case of a system crash, this can be valuable information and help to locate the problem.

4. Implementation

The user interface developed in the current project has been designed for the second car of the REV Project, a converted Lotus Elise S2. The goal of this vehicle is to test the suitability of electric drives for sports cars.

Figure 4.1.: The VoomPC-2 Car PC [min10]

4.1. Hardware

4.1.1. Computer

The computer used in this car is based on a ZOTAC ION ITX D-E mainboard in a VoomPC-2 Car PC enclosure (figure 4.1). This enclosure is designed to work in automotive environments with high temperatures, dust and vibrations. The power supply unit is a M2-ATX Intelligent Automotive DC-DC Car PC Power Supply that can handle voltages between 6 and 24 V. It boots up and shuts down the computer automatically when the car is started and stopped. In addition, it controls the amplifier and prevents deep discharge of the battery. Its CPU is an Intel Atom 330 with 2 Cores, both running on 1.6 GHz, and it has 2 GB DDR2 memory and a 160 GB hard drive. It is controlled via a 7" touchscreen with a native resolution of 720x480 and connected to the internet via WLAN or a mobile broadband USB modem. The operating system is Microsoft Windows XP.

4.1.2. GPS Receiver

The program locates the position of the car from a QStarz GPS 818X GPS receiver (figure 4.2). It features a maximum sampling rate of 5 Hz and outputs the position data according to the NMEA-protocol [nme10]. The receiver is attached via USB but is recognized as a virtual serial port device. It can therefore be used on all operating systems.

Figure 4.2.: The QStarz GPS 818X GPS Receiver [qst10]

4.1.3. Battery Management System

The battery management system protects the battery and controls the charging process. It monitors the voltage of every battery cell to detect damages and measures the voltage and current of the battery pack. When the maximum voltage of a cell is reached during the charging process, the battery module bypasses the current to avoid overcharging the cell. At the same time, it issues a command to limit the charging current in order to prevent the bypass resistor from overheating. Every cell has its own module (figure 4.3) for monitoring the voltage and controlling the bypass resistor. These cell modules are controlled by the master module, which also measures the battery voltage and the current. The master module sends the data via serial port and a USB-to-Serial adapter to the PC. The battery management system is custom-built for the REV Project.

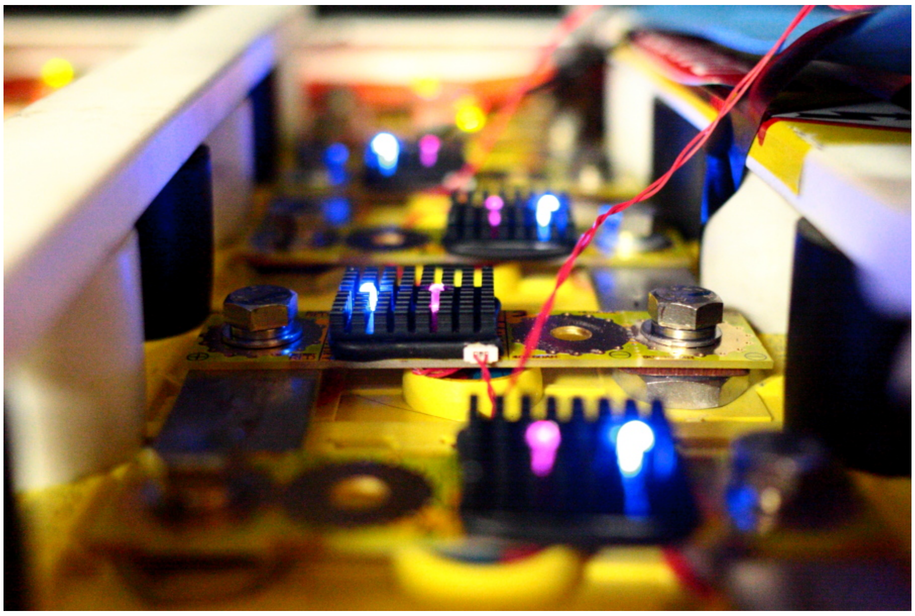


Figure 4.3.: The REV Project Battery Management System (photo by Jonathan Wan)

4.2. Development Framework

The choice of programming language is an important decision at the beginning of every development. For the current project, the Nokia Qt C++ development framework was used for the following reasons:

4.2.1. Portability

Programs developed with Qt can be compiled for all major platforms, including Windows, Linux, Unix, Mac OS X and even embedded systems. This is very important to enable the team to switch to different hardware and software without major modifications to the program.

4.2.2. Speed

C++ is fast compared to other programming languages (e.g. Java) as it does not need a runtime environment. Qt also uses the native APIs of each supported platform. This makes it possible to run the program on low-performance hardware.

4.2.3. Robustness

The Qt extensions provide convenience functions which speed up the development, eliminate many potential pitfalls and therefore make the program more robust.

4.2.4. Internationalisation

Qt has a powerful and easy to use translation support. It separates the text that needs to be translated from the source code, which enables team members without programming knowledge to do the translation.

4.2.5. Open Source

The Qt development framework, including the Qt Creator IDE, is open source and freely available for all major platforms. This gives students the opportunity to participate in the development of the program independent of the operating system that they use and without the need to purchase software. Qt is available under two different licenses, one of them is the GPL.

4.3. General Features

All the backend modules have some general features in common that increase the flexibility and reusability of the program, or parts of the program.

```
  void Logger::setDefaultSettings()
  {
    stopLogging();

    settings.beginGroup("logging");
    settings.remove("");
    settings.setValue("logDir", defaultLogDir);
    settings.setValue("logName", defaultLogName);
    settings.setValue("logInterval", defaultLogInterval);
    settings.setValue("trkName", defaultTrackName);
    settings.setValue("trkNumber", 0);
    settings.setValue("unitIMEI", defaultUnitIMEI);
    settings.setValue("serverName", defaultServerName);
    settings.setValue("serverPort", defaultServerPort);
    settings.endGroup();

    emit debugMsg(QtDebugMsg, objName, "default settings applied ");
}
```

Listing 4.1: setDefaultSettings() function of the logging module

4.3.1. Settings

Qt provides a platform independent class for managing the settings of a program. The settings are stored at the native location for each platform to comply with programming standards. If desired, it uses the INI file format (listing 4.2) to enable the user to unify the settings under different operating systems. If the program tries to access a nonexistent setting, it will fall back to a predefined default value. All classes have a function to reset the settings to their default values as pictured in listing 4.1

```
[logging]
trkNumber=0
logDir=log_files
logName=yyyy-MM-dd_hh-mm
logInterval=1000
trkName=Lotus Test Drive
unitIMEI=123456789012345
serverName=shellsrv.ee.uwa.edu.au
serverPort=2001
```

Listing 4.2: The default settings of the logging module in the INI file

4.3.2. Serial Port Interface

Both the GPS receiver and the battery management system are accessed over a virtual serial port. This program uses the `QextSerialPort` class, which is an abstraction layer for the serial port interfaces of the different operating systems. This makes it possible to use the same code for all platforms, provides convenience functions for reading and writing the serial port and returns error and debug strings. Listing 4.3 shows how to open a serial port and, if this fails, how to read the error string.

```
bool GPS::init(QString portName)
{
    port->setPortName(portName);
    port->setBaudRate(BAUD115200);
    port->setFlowControl(FLOW_OFF);
    port->setParity(PAR_NONE);
    port->setDataBits(DATA_8);
    port->setStopBits(STOP_1);
    port->setTimeout(10);

    if (port->open(QIODevice::ReadWrite) == true){
        updateTimer->start(200); // read data every 200 ms
        emit debugMsg(QtDebugMsg, devName, "listening for data on " + port->portName());
    }
    else
        emit debugMsg(QtWarningMsg, devName, "failed to open " + port->portName() + " - "
            + port->errorString());

    return port->isOpen();
}
```

Listing 4.3: Initialisation of the serial port for the GPS receiver

4.3.3. Signals and Slots

The program takes advantage of the Qt concept of signals and slots. Every object can send and receive signals. These signals can not only trigger events but can also contain data. This mechanism makes the code leaner and more flexible. The first two lines in listing 4.4 show how signals are used to update the gps values in the user interface and the logger; the following lines demonstrate how signals are used to control the player. Figure A.1 gives an overview of the signals and the structure of the program.

```
connect(&gps, SIGNAL(updated(PositionData)), this, SLOT(update_ui_gps(PositionData)));
connect(&gps, SIGNAL(updated(PositionData)), &logger, SLOT(updatePosition(PositionData)));

connect(ui->toolButton_play, SIGNAL(clicked()), &player, SLOT(togglePlay()));
connect(ui->toolButton_stop, SIGNAL(clicked()), &player, SLOT(stop()));
connect(ui->toolButton_next, SIGNAL(clicked()), &player, SLOT(next()));
connect(ui->toolButton_prev, SIGNAL(clicked()), &player, SLOT(prev()));

connect(ui->checkBox_repeatThis, SIGNAL(toggled(bool)), &player,
    SLOT(setRepeatItem(bool)));
connect(ui->checkBox_repeatAll, SIGNAL(toggled(bool)), &player, SLOT(setRepeatAll(bool)));
```

Listing 4.4: Some examples for the usage of signals and slots in this work

4.3.4. Error Logging

Every class has the signal `debugMsg(WarnLevel, QString, QString)` to display and log messages. These messages can contain debug information, warnings and errors. An example of their usage can be seen in the listings 4.1 and 4.3. The messages are sent to an object of the `DebugLogger` class. Here they can be logged to a file, printed in the console or sent to the user interface (listing 4.5). The warn level allows filtering according to the importance of a message; the source string can be used to log only messages from a certain source. Furthermore, the current time and source are added to the message string for easier bug tracking. This structure makes it easy to add new modules to the program and to change the settings for processing, filtering and storage of the debug and error information.

```
void DebugLogger::logMsg(QtMsgType warnLevel, QString source, QString msg)
{
    if (!source.isEmpty())
        msg = "["+source+"] "+msg;

    msg = QTime::currentTime().toString("hh:mm:ss ") + msg;

    qDebug() << msg;

    if (warnLevel >= QtDebugMsg){
        if (logFile.isOpen())
            logStream << msg << endl;
        emit displayMsg(msg);
    }
}
```

Listing 4.5: All the debug messages are logged with this function

5. Backend

The separation of the data processing parts of a program from the interface is not only good practice, it also increases flexibility and makes it easier for the developer to locate problems. In this work, all the data processing is done by the backend modules, leaving the interface devoted to display of results and interaction with the user.

5.1. Battery

The `Battery` class receives data from the battery management system described in chapter 4.1.3. The data is read from the virtual serial port and comes in the format given in table 5.1. Each command follows the same structure: it starts with a capital letter, followed by a one byte value, a two byte value and another one byte value, all separated by commas. The first example gives the current voltage of cell number 17, which is 3.20 V, and the minimum voltage, which is 2.7 V. When fully charged, the last value for this command gives the bypass current for this cell (second example). In the third example, the overall battery voltage of the battery pack is 276 V. The last example gives the current of the battery, which in this case is 57.3 A consumption. For regenerative braking and charging, the first value is `1` to indicate the changed direction of the current. A full documentation of the commands that the BMS sends to the PC is given in table A.1.

command	1. value	2. value	3. value	example
V	cell number	cell voltage [V/100]	min. voltage [V/10]	V,17,320,27
V	cell number	cell voltage [V/100]	bypass current [A/10]	V,17,381,6
B	battery number	pack voltage [V/100]	(not used)	B,1,27600,0
A	current direction	current [A/10]	(not used)	A,0,573,0

Table 5.1.: Commands for the battery management system

The current capacity of the battery is not recorded by the battery management system and therefore must be calculated. This is done by integrating the values of the current. To calculate the remaining capacity of the battery, the system requires the capacity at the start of the program, which is saved in the settings (see table 5.4). To calculate energy consumption, the current is multiplied by the voltage and integrated over the time. Current in a positive (i.e. charging) direction is integrated separately to measure the effect of regenerative braking.

Additionally, to provide the driver with an estimate of the distance the car is able to travel, the remaining distance is calculated from the remaining capacity and the average consumption per kilometre.
All of these battery related values are stored in a structure of the type `BatteryData`. The values of this structure are listed in table 5.2. This structure is used to send the data to the user interface and the logger. When the object has received and processed new data from the battery management system, it emits an `updated()` signal to notify the user interface and the logger of the updated battery values.

Name	Type	Description
cellNumber	int	Number of battery cells
batteryType	BatteryType	Type of the battery cells
charging	bool	True if battery is charging
cellError	bool	True if a cell module reports an error
voltage	float	Voltage of the battery pack [V]
current	float	Current of the battery pack [A]
energyUsed	float	Energy consumed since last charging [kWh]
energyRegenerated	float	Regenerated energy since last charging [kWh]
capacityRemaining	float	Remaining battery capacity [Ah]
capacityPercent	float	Percentage of remaining capacity [Ah]
distanceRemaining	float	Estimation of the remaining distance [km]
cellVoltage	QList<float>	Current voltages of the cells [V]
cellVoltageMax	QList<float>	Maximum voltages since last reset [V]
cellVoltageMin	QList<float>	Minimum voltages since last reset [V]

Table 5.2.: Values of the `BatteryData` structure

In this structure, `cellNumber` is the number of cells in the battery and `cellVoltage`, `cellVoltageMin` and `cellVoltageMax` contain the voltages of the single cells. All other values refer to the battery pack. The voltages are given in [V], the current in [A], the capacities in [Ah] and the remaining distance in [km].
The type of battery cell defines some constants, such as maximum and minimum voltage, capacity and maximum current. To use the program on different cars with different types of batteries, this information is also saved in a structure. The values of this structure are illustrated in table 5.3; an example for a battery type is given in listing 5.1.

```
static const BatteryType TS_LYP60AHA = {
    3.8, 3.6, 3.1, 3.0, 2.6, 60.0, 180.0, 60.0 };
```

Listing 5.1: Example for a battery properties definition

The settings for the battery module are listed in table 5.4. The name of the device can be set here and is used for logging. This can be useful for evaluating the log files when

Name	Type	Description
cellVoltageOC	float	Overcharged cell voltage [V]
cellVoltageFull	float	Fully charged cell voltage [V]
cellVoltageNom	float	Nominal cell voltage [V]
cellVoltageWarn	float	Cell voltage warning level [V]
cellVoltageEmpty	float	Empty cell voltage [V]
cellCapacity	float	Battery cell capacity [Ah]
maxChargeCurrent	float	Maximum charge current [A]
maxDischargeCurrent	float	Maximum discharge current [A]

Table 5.3.: Values of the `BatteryType` structure

working with different battery management systems. The `port` and `batteryType` settings allow running of the same program on different cars without recompiling it.

Name	Type	Default	Description
deviceName	QString	Battery Module	Name displayed in the debug messages
port	QString	COM3	Serial port for the BMS
batteryType	BatteryType	TS_LYP60AHA	Type of the battery (see listing 5.1)
lastCapacity	float	60.0	Remaining capacity at last usage

Table 5.4.: Settings for the battery management system

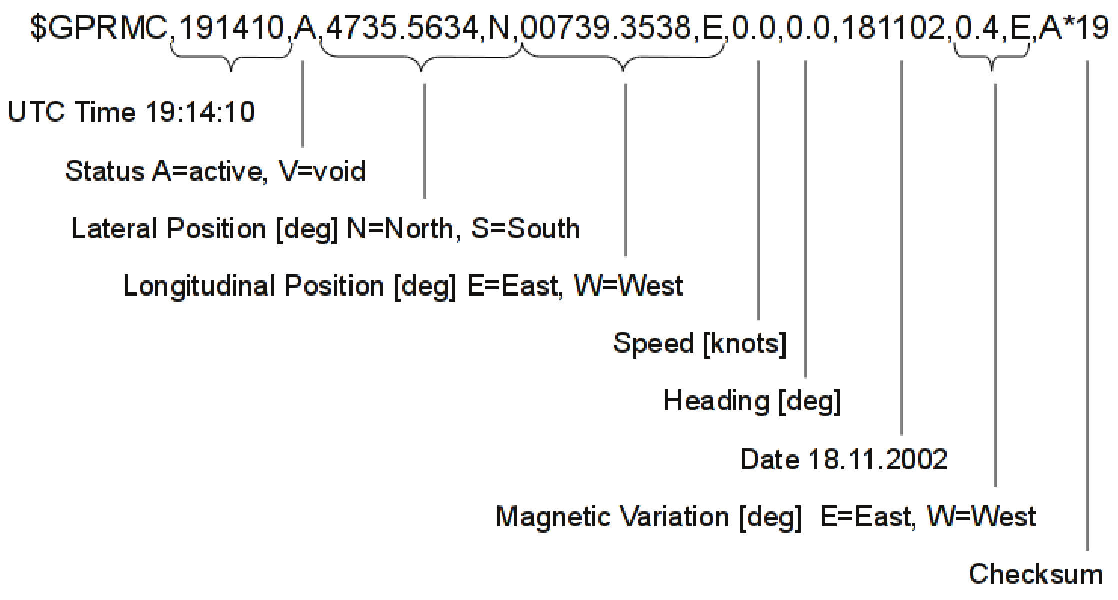

$GPRMC,191410,A,4735.5634,N,00739.3538,E,0.0,0.0,181102,0.4,E,A*19
UTC Time 19:14:10
Status A=active, V=void
Lateral Position [deg] N=North, S=South
Longitudinal Position [deg] E=East, W=West
Speed [knots]
Heading [deg]
Date 18.11.2002
Magnetic Variation [deg] E=East, W=West
Checksum

Figure 5.1.: The GPRMC sentence specification, part of the NMEA format

5.2. GPS

The `GPS` class has many similarities to the `Battery` class: it reads data from a virtual serial port, processes the information and emits a signal containing a data structure. The data comes from the GPS receiver introduced in chapter 4.1.2 and follows the NMEA standard. This standard is defined by the National Marine Electronics Association [nme10] and describes how the information from the GPS is encoded to ASCII strings. The most important parts of this definition are the GPRMC sentences. According to [nme01], they define the ″recommended minimum specific GPS/Transit data“. Figure 5.1 shows the structure of a GPRMC sentence. The GPGGA (″Global Positioning System Fix Data“) sentences are also read for retrieving information about the number of satellites the GPS can locate, the accuracy of the position and the current altitude.

Name	Type	Default	Description
deviceName	QString	QSTARZ GPS 818X	Name displayed in the debug messages
port	QString	COM2	Serial port for the GPS
totalDistance	float	0.0	Overall distance [km]

Table 5.5.: Settings for the GPS receiver

Similar to the capacity of the battery in chapter 5.1, the total distance is calculated by integrating the speed over the time. This value is stored in the settings (table 5.5) and loaded again at the start of the program.

Name	Type	Description
valid	bool	True if the GPS data is valid
UTCDateTime	QDateTime	UTC Time from the GPS
UTCDay	int	Day (UTC)
UTCMonth	int	Month (UTC)
UTCYear	int	Year (UTC)
UTCHour	int	Hours (UTC)
UTCMinute	int	Minutes (UTC)
UTCSecond	int	Seconds (UTC)
UTCMilliSec	int	Milliseconds (UTC)
latitude	float	Position latitude
longitude	float	Position longitude
altitude	float	Position altitude
hdop	float	Horizontal dilution of precision
numSat	int	Number of satellites in view
heading	float	Heading [deg]
speed	float	Speed [km/h]
totalDistance	float	Overall distance [km]

Table 5.6.: Values of the `PositionData` structure

The data from the GPS is written to a `PositionData` structure with the values listed in table 5.6. The UTC date and time is read from the GPS signal. Latitude, longitude and heading are given in degrees, altitude in metres, the total distance in kilometres and the speed in kilometres per hour.

5.3. Logger

The Logger class is responsible for collecting the vehicle data, calculating statistics about the current trip, storing this data on the hard disk and streaming the current status of the car to the team server. Similar to the PositionData and BatteryData, the information about the trip is stored in a structure called TripData (see table 5.7). This structure is emitted as a signal every time it is updated to display the new values in the user interface. All data is written in regular, user-defined intervals to a GPX and a CSV file and is transmitted to the server. The file names, the directory and the server address are saved in the configuration file (see table 5.8) and can therefore be changed by the user.

Name	Type	Description
time	int	trip time [ms]
drivingTime	int	driving time [ms]
distance	float	trip distance [km]
speedAvg	float	average speed [km/h]
speedMax	float	maximum speed [km/h]
accel60	float	best acceleration time 0..60 km/h [s]
accel100	float	best acceleration time 0..100 km/h [s]
voltageMin	float	minimum voltage [V]
voltageMax	float	maximum voltage [V]
currentAvg	float	average current [A]
currentMax	float	maximum current [A]
energyUsed	float	consumed energy [kWh]
energyRegen	float	regenerated energy [kWh]
energyPerKm	float	energy per km [kWh]

Table 5.7.: Values of the TripData structure

All values in table 5.7 refer to the current trip. The time value is the time elapsed since the start of logging, while drivingTime is the time the car has been in motion. The times are given in milliseconds, the distance in kilometres, and the average and maximum speed in kilometres per hour. accel60 and accel100 are the best acceleration times from zero to 60 and 100 kilometres per hour. The voltages, currents and energies are equivalent to the values in table 5.2.

Name	Type	Default	Description
logDir	QString	log_files	Name of the folder for the log files
logName	QString	yyyy-MM-dd_hh-mm	Naming scheme for the log files
logInterval	int	1000	Log interval [ms]
trkName	QString	Lotus Test Drive	Track name for the GPX file
trkNumber	int	0	Track number for the GPX file

Table 5.8.: Settings for the logging module

5.3.1. GPX File

The GPX file format for storing position data is an international standard based on the XML file format [xml03]. It supports the recording of tracks and single points, can include comments, and is extendable. The full definition can be found under [gpx04]. Creating GPX documents with Qt is facilitated by the built-in support for reading and writing XML documents. The `QXmlStreamWriter` class formats the file and ensures that the XML structure is valid. When logging starts, the header of the GPX file is created. It contains the version of the format, the track name and the track number. When logging is finished, all open XML elements are closed before closing the file. In listing 5.2, the code for saving a track point can be seen; listing 5.3 shows the corresponding part of the GPX file. These track points are saved at user defined intervals.

```
gpxWriter.writeStartElement("trkpt");
    gpxWriter.writeAttribute("lat", QString::number(positionData.latitude, 'f', 8));
    gpxWriter.writeAttribute("lon", QString::number(positionData.longitude, 'f', 8));
  gpxWriter.writeTextElement("ele", QString::number(positionData.altitude, 'f', 2));
  gpxWriter.writeTextElement("time",
      positionData.UTCDateTime.toString("yyyy-MM-dd'T'hh:mm:ss'Z'"));
  gpxWriter.writeTextElement("hdop", QString::number(positionData.hdop, 'f', 2));
  gpxWriter.writeStartElement("extensions");
    gpxWriter.writeTextElement("speed", QString::number(positionData.speed, 'f', 2));
  gpxWriter.writeEndElement();
gpxWriter.writeEndElement();
```

Listing 5.2: Code for logging a track point to the GPX file

The element `ele` saves the altitude, which is the height above mean sea level. The `hdop` value is the horizontal dilution of precision, which quantifies the imprecision of the position measurement. The `extension` element can store user defined values that are not part of the GPX standard, such as the speed of the vehicle. A complete GPX file is shown in listing A.1.

```
<trkpt lat="-31.94993000" lon="115.98387333">
  <ele>30.20</ele>
  <time>2010-09-23T04:07:12Z</time>
  <hdop>1.19</hdop>
  <extensions>
    <speed>19.63</speed>
  </extensions>
</trkpt>
```

Listing 5.3: Extract of the GPX file

5.3.2. CSV File

CSV files are a simple way of storing data as they are plain text and only have basic formatting. They can be written by micro controller programs and can be read by every common spreadsheet application. This makes them ideal for storing and exchanging data within this project, although they are not as flexible as XML based formats. The first line of a CSV file is the header that contains the type of values below. The values are saved in rows, and the columns are separated by semicolons. Listing 5.4 shows the code for writing a row; listing A.2 contains the first lines of a CSV file. The `csvStream` is an object of the class `QTextStream`, which offers convenience functions like automatic conversion of numbers to strings with variable precision.

```
    csvStream << (float)(tripData.time)/1000.0 << "; " << tripData.distance << "; " <<
            positionData.speed << "; " << positionData.altitude << "; " <<
            batteryData.voltage << "; " << batteryData.current << "; " <<
            batteryData.capacityRemaining << "; " <<
            tripData.energyUsed << "; " << tripData.energyRegen;
    for (int i=0; i<batteryData.cellVoltage.length(); i++)
        csvStream << "; " << batteryData.cellVoltage.value(i);
    csvStream << endl;
```

Listing 5.4: Code for adding a line to the CSV file

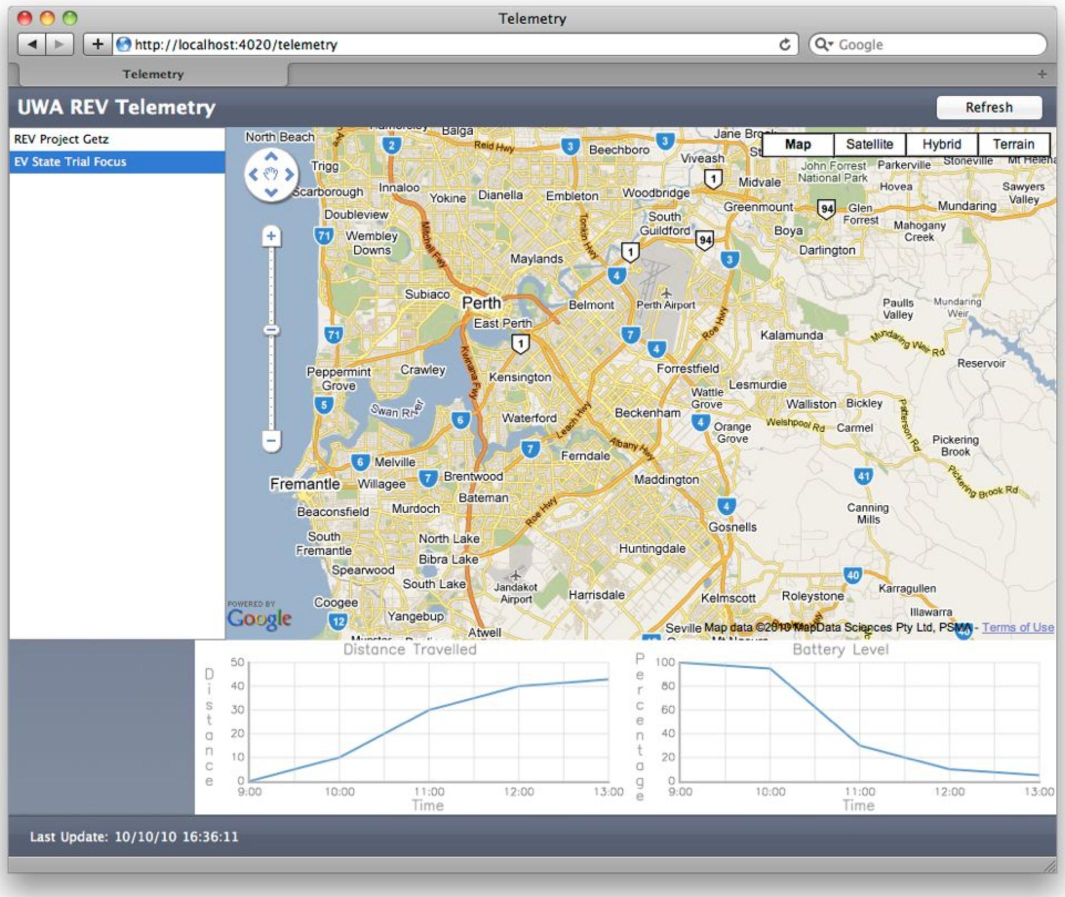


Figure 5.2.: The vehicle data on the web interface of the server [Pea10]

5.3.3. Server

When testing a car it is advantageous for members of the team to be able to access data as quickly as possible. With the UMTS module in the car, it is possible to send the status of the car to a server and enable team members to monitor the car's performance live over the internet. The software running on the server is part of the REV Project and is also used by the other cars in the project. The protocol is adjusted to the needs of micro controller based telemetry systems and optimized for low bandwidth consumption. A standard packet as shown in listing 5.9, with no digital or analogue inputs, has a size of only 33 Bytes. The status of the car is sent to the server every second.

The first byte of the packet signals what protocol is used. In this case, it is an ASCII 'R' as the protocol is designed for the REV Project. The bytes in position one and two give the length

Position	Field	Type	Description
0	Protocol ID	uint8	'R' for the REV Project
1	Packet Length	uint16	Total length of the message
3	Unit IMEI	uint64	Mobile broadband device IMEI (7 Bytes)
10	Latitude	int32	Latitude multiplied with 10^6
14	Longitude	int32	Longitude multiplied with 10^6
18	Julian Time	uint32	Seconds since January 6 1980, 00:00:00
22	Speed	uint8	Speed in km/h
23	Heading	uint8	Heading divided by 2
24	Altitude	uint8	Altitude divided by 20
25	Reason Code	uint16	Bit code indicating the reason for sending
27	Status Code	uint8	Bit code indicating the status of the car
28	DI Count	uint8	Number of digital inputs
28+	DI Values	uint8	Values of the digital inputs
29	AI Count	uint8	Number of analogue inputs
29+	AI Values	uint16	Values of the analogue inputs
30	Battery Level	uint8	Battery charge level percentage
31	Checksum	uint16	CRC-16-ANSI checksum

Table 5.9.: Protocol for the server communication

of the message, including the checksum. The following seven bytes contain the IMEI number [ime10] of the 3G/GPRS network device, a unique number that is used to identify the car. The latitude and longitude of the car's position are multiplied by 10^6 and transmitted as signed 32 Bit integers. The time is given in seconds from the seed date, which is January 6 1980 at midnight. The byte in position 22 is the speed of the car, followed by the heading divided by two, and the altitude divided by 20. The two bytes of the reason code indicate the reason for sending, and the status code byte indicates the status of the car. The meaning of the bits in these codes is given in table 5.10. Byte 28 gives the number of digital inputs. If this number is greater than zero, the following bits are used for transmitting the status of the inputs. If the number of digital inputs is not divisible by eight, these bits are filled up to the full byte. The next byte is the number of analogue inputs, followed by two bytes per analogue input. The last data byte is the battery charge level in percentage.
The last two bytes of the message are a CRC-16-ANSI checksum which allows detection of transmission errors [RG02].

The messages are created in defined time intervals and stored in objects of the type `QByteArray`. To ensure that no information is lost, every new message is appended to the `QList` object `messageList` before the call of the function `sendMessage`. Listing 5.5 shows the relevant part of the functions involved in server communication. Before sending a new message, the program checks if it is connected to the server, if there is another message being

Bit	Reason Code Low Byte	Reason Code High Byte	Status Code
0	Time interval	Heading changed	Ignition status
1	Distance interval	Low battery	Initial power on report
2	Sent on demand	Charging status changed	GPS timeout
3	Reserved	Reserved	Reserved
4	Panic switch activated	Reserved	Reserved
5	External input changed	Reserved	Reserved
6	Journey Start (Ignition On)	Reserved	Reserved
7	Journey Stop (Ignition Off)	Reserved	Reserved

Table 5.10.: Reason and status bytes for the server communication

processed at that moment, and if there are any messages in the `messageList`. After sending the message to the server, the function starts a timer.
The answer of the server is handled with the `receiveMessage` function. If the server has sent an <ACK> packet ('acknowledge', ANSI 0x06), the item is removed from the list and the next message can be sent. Otherwise, the `sendMessage` call will attempt to send the message again. If the server does not answer within a defined time, the timer calls the `messageTimeout` function to send the message again.

```
void Logger::sendMessage(){
    if (revServer.state()==QAbstractSocket::ConnectedState
    && !connectionTimer->isActive() && !messageList.isEmpty()){
        revServer.write(messageList.first());
        connectionTimer->start(); }}

void Logger::receiveMessage(){
    QByteArray answer = revServer.readAll();
    if (answer.endsWith(0x06) && !messageList.isEmpty())
      messageList.removeFirst();
    connectionTimer->stop();
    sendMessage(); }

void Logger::messageTimeout(){
    connectionTimer->stop();
    sendMessage(); }
```

Listing 5.5: Server communication functions

The server not only stores the data received from the car, it also displays it on the web interface. Figure 5.3 shows the mobile version of the page, which displays the last position of the car and the battery level. The server software and the web interface were developed by John Pearce [Pea10].

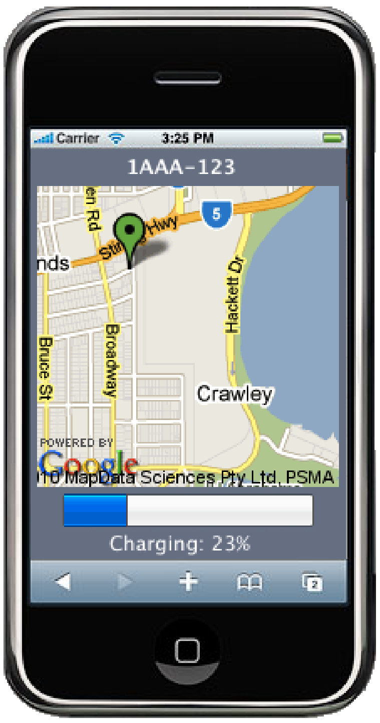


Figure 5.3.: The mobile version of the web interface [Pea10]

5.4. Map Widget

The map widget provides a so-called `Slippy Map` [sli10] for displaying the current position of the car. The map has multiple zoom levels, is continuously slidable and can be centred on the position of the car. The map widget draws the current position of the car and, if selected, the current route. It can also load a track from a GPX file to help the driver follow a predefined test route. The core parts of the widget are based on an example program from the Qt developers [Hid10] that has been modified in many aspects to fit the needs of the REV project.

Name	Type	Default	Description
latitude	float	-31.979283	Last latitude of the map
longitude	float	115.815852	Last longitude of the map
zoom	int	15	Last zoom of the map
myPosLat	float	-31.979283	Last latitude of the car
myPosLng	float	115.815852	Last longitude of the car
myDir	float	180	Last heading of the car
centered	bool	false	If true, map is centered on car position
invert	bool	false	If true, map is in night mode
myTrackVisible	bool	true	If true, the current track is visible
testTrackVisible	bool	false	If true, the test track is visible
testTrackCmtsVisible	bool	false	If true, comments to the test track are visible
trackFile	QString	testtrack.gpx	File name of the test track
tilesStoredLocally	bool	true	If true, tiles are stored on HDD
tilesStorageDirectory	QString	tiles	Folder for the tiles
baseURL	QString	http://tile.open streetmap.org	Server URL for downloading tiles
minZoom	int	1	Minimum zoom of the map
maxZoom	int	18	Maximum zoom of the map
maxWidth	int	1920	Maximum width for the widget
maxHeight	int	1200	Maximum height for the widget

Table 5.11.: Settings for the map widget

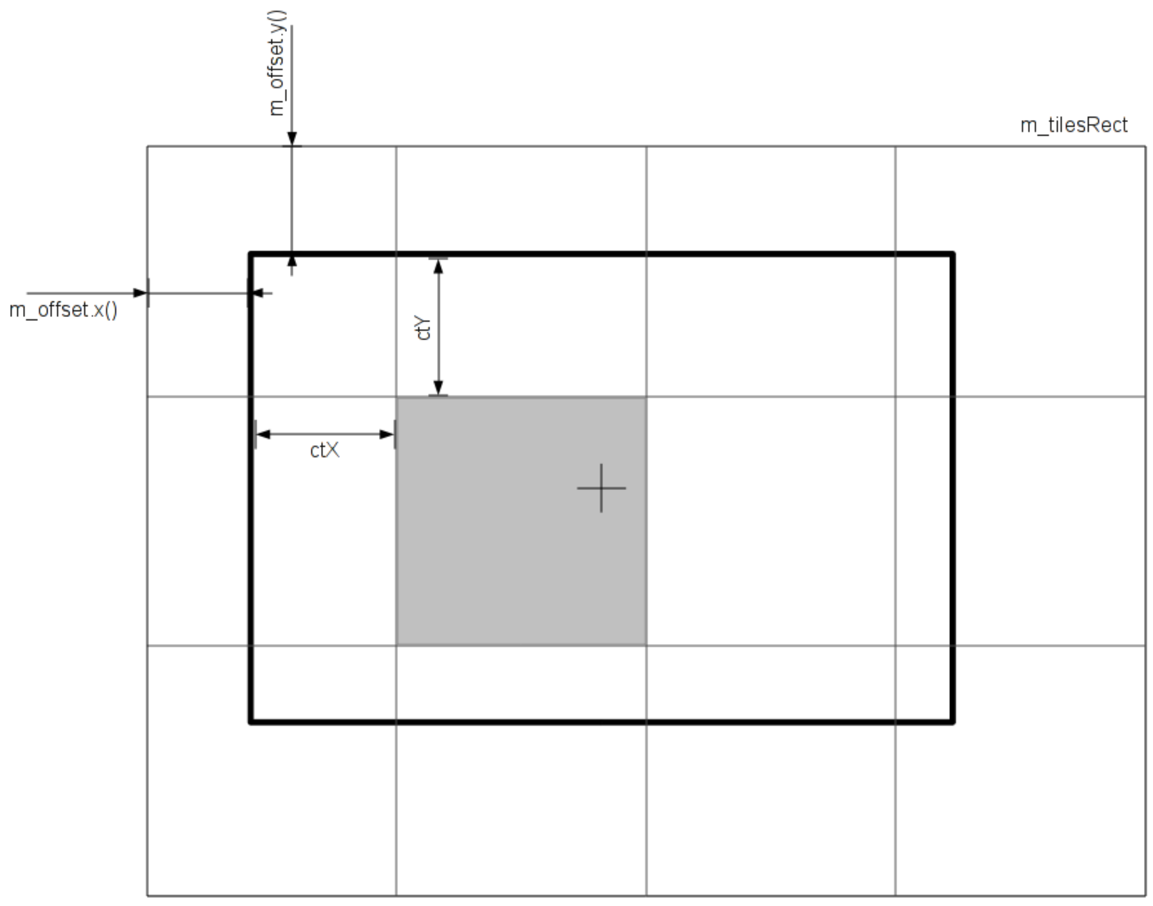


Figure 5.4.: The tiles in the Slippy Map

5.4.1. Slippy Map

The Slippy Map consists of several images, the so called tiles, which have a size of 256x256 pixels. The `SlippyMap`-Object selects the correct tiles according to the area and the zoom level that needs to be displayed. These tiles are put into position, missing tiles are downloaded, and tiles that are no longer needed are removed from memory.
Figure 5.4 shows the tiles (grey squares) and the map widget (black rectangle). The cross marks the centre of the map, and the grey square is the centre tile. The values `ctX` and `ctY` give the position of the centre tile, which is the distance between the top-left corner of the tile and the top-left corner of the widget. Listing 5.6 shows the calculation of the rectangle containing the tiles (`m_tilesRect`) and its position relative to the map widget (`m_offset`).

```
    QPointF centerTileF = tileForCoordinate(latitude, longitude, zoom);
    QPoint centerTile = QPoint((int)centerTileF.x(),(int)centerTileF.y());

    int ctX = width / 2 - (centerTileF.x() - centerTile.x()) * tdim;
    int ctY = height / 2 - (centerTileF.y() - centerTile.y()) * tdim;

    QPoint firstTile = centerTile - QPoint((tdim+ctX-1)/tdim, (tdim+ctY-1)/tdim);
    QPoint lastTile = centerTile + QPoint((width-ctX-1)/tdim,(height-ctY-1)/tdim);

    m_tilesRect = QRect(firstTile, lastTile);
    m_offset = QPoint(ctX, ctY) - (centerTile - firstTile) * tdim;
```

Listing 5.6: Generating the Slippy Map

5.4.2. Map Tiles

The map tiles for Slippy Map can be retrieved from different sources, dependent on how much disc space, computing power and bandwidth is available.

Downloading

Downloading the tiles from the official map server is the most flexible approach. It requires minimal disc space and the tiles are always up to date. Downloading one tile with UMTS usually takes between 400 and 800 ms, with a minimum below 10 ms for empty tiles (water, tile size 124 Bytes), and a maximum above 1000 ms (tile size up to 30 kB). Problems include the need for a constant Internet connection, and high traffic. Both issues are major disadvantages for mobile application.

Rendering

Instead of downloading the tiles from a server, they can also be rendered on the PC itself. The advantage of this option is the small amount of hard disc space required to cover large areas. For example, the map data for Australia is 1.7 GB, whereas the tiles of zoom level 16 already require several hundreds of gigabytes when rendered.
One disadvantage of this approach is the setup of the rendering engine. The most popular renderer Mapnik [map10] requires several other programs to run and is not trivial to configure. From the programmers point of view, there are problems with different versions of potentially changed behaviour, portability issues between the different operating systems, and problems with dependencies when shipping the program. Another important issue is the rendering time: one tile takes between 2.7 and 2.8 seconds on a 2 GHz dual core CPU; the

first tile after startup takes more than 10 seconds. The hardware in the car is less powerful than this CPU and therefore takes even longer, making the system unusable.

TileLite

TileLite [til10] is an alternative for including the Mapnik libraries into the program. It is a small tile server written in Python and using the Mapnik renderer. As it can be accessed in the same way as the official OSM tile server, the only change to the program required is using the `localhost` address instead of `openstreetmap.org`. This address is saved in the configuration file and can be changed by the user. The rendering takes between 2.35 and 2.60 seconds, which is slightly faster than direct usage of the libraries, but is still to slow for usage in the car.

Hard Disk

Loading tiles from the hard disk is very fast compared to rendering as it normally takes between 5 and 50 ms. Only when the hard disk is in power-saving mode, will it take up to 400 ms for the first tile to load. The whole city area of Perth in the zoom levels 12 to 16 consists of 36000 tiles with a size of 92 Megabytes. This is sufficient because the cars of the REV Project are limited in their range and unlikely to be used outside of the Perth city area. However, to increase flexibility, tiles that are not on the hard disk can be downloaded if the computer is connected to the internet. These tiles are then stored on the hard disk and do not need to be downloaded again.

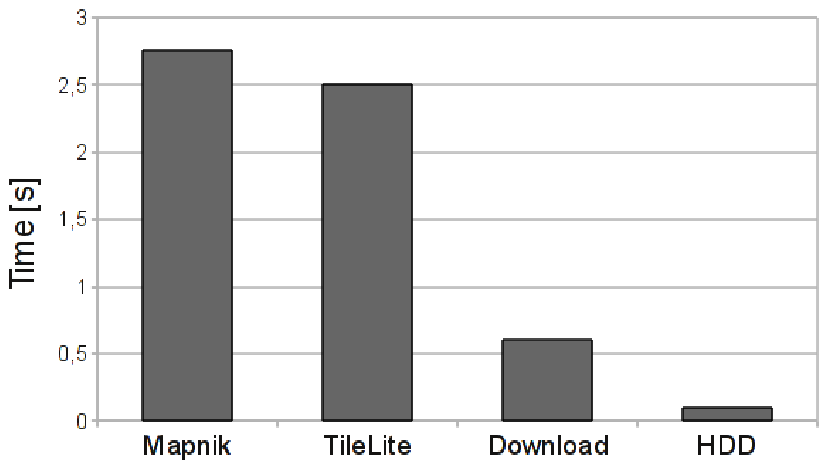


Figure 5.5.: Times for generating or loading one tile

5.5. Player

The `Player` class controls the playback of music files and manages the playlist. It uses the Phonon multimedia framework [pho10], which is part of Qt. This framework provides multimedia functions in a platform independent way, using the native backends of the targeted systems. The multimedia backends used are DirectShow on Windows, QuickTime on Mac OS X and GStreamer on Linux.
The main task of the class is maintaining the list of sources. The sources can be files or web streams. The class can add new sources, move existing sources within the list, select a source, remove a single source, or clear all sources from the list. For displaying the sources in the user interface, it creates a list containing the meta data. This list is updated every time the list of sources is changed, and the new meta data list is emitted as a signal. The class supports all common playback features, including skip to the next track, skip to a certain position, repeat the track and repeat the playlist. It saves the `repeatItem` and `repeatAll` setting as well as the current playlist, the selected item and the current volume (table 5.12).

Name	Type	Default	Description
repeatItem	bool	false	If true, the selected item is repeated
repeatAll	bool	true	If true, the playlist is repeated
mediaSources	QStringList	<empty>	The files in the playlist
selectedItem	int	0	Selected playlist item
volume	float	0.8	Sound volume of the player

Table 5.12.: Settings for the music player

6. User Interface

6.1. General Features

One of the main goals of the program is to achieve maximum reusability and flexibility. On the one hand, this is accomplished by a clean separation of the backend and the user interface; on the other hand, the user interface itself is designed to to comply with these requirements.

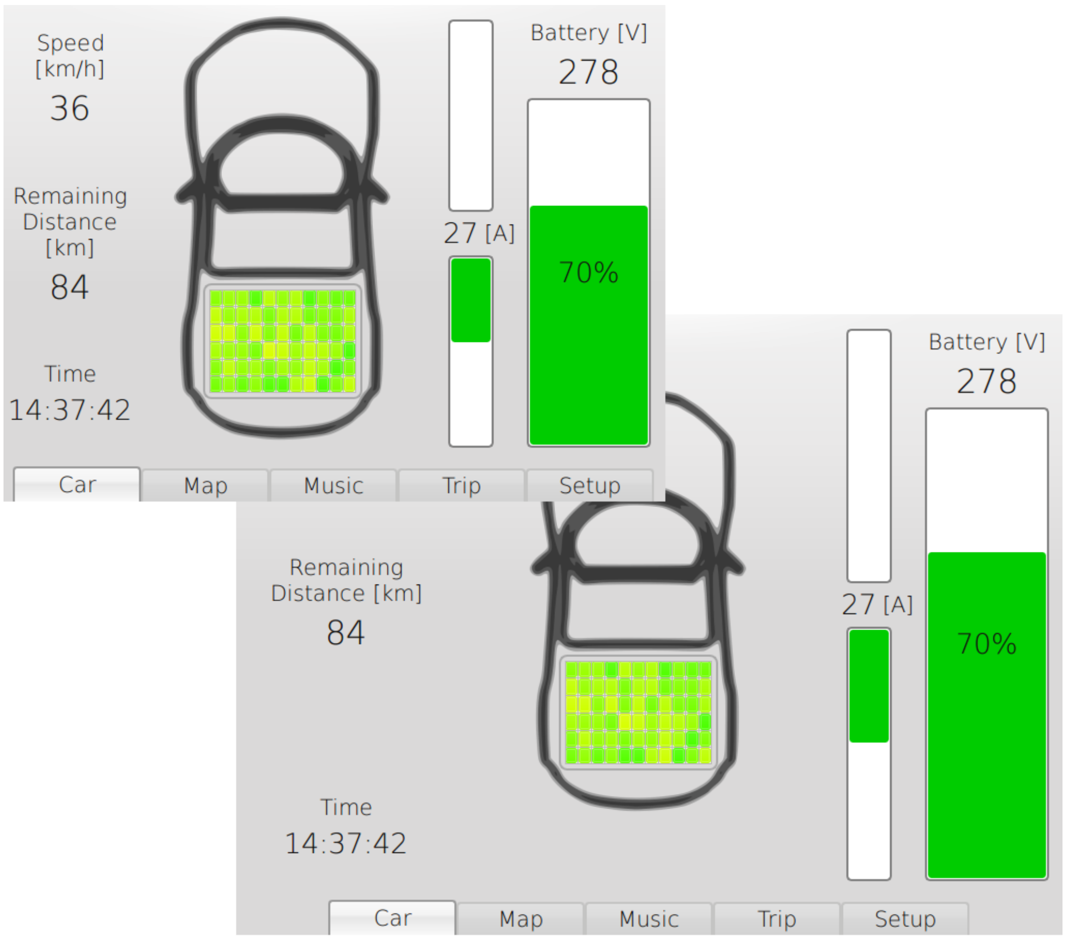


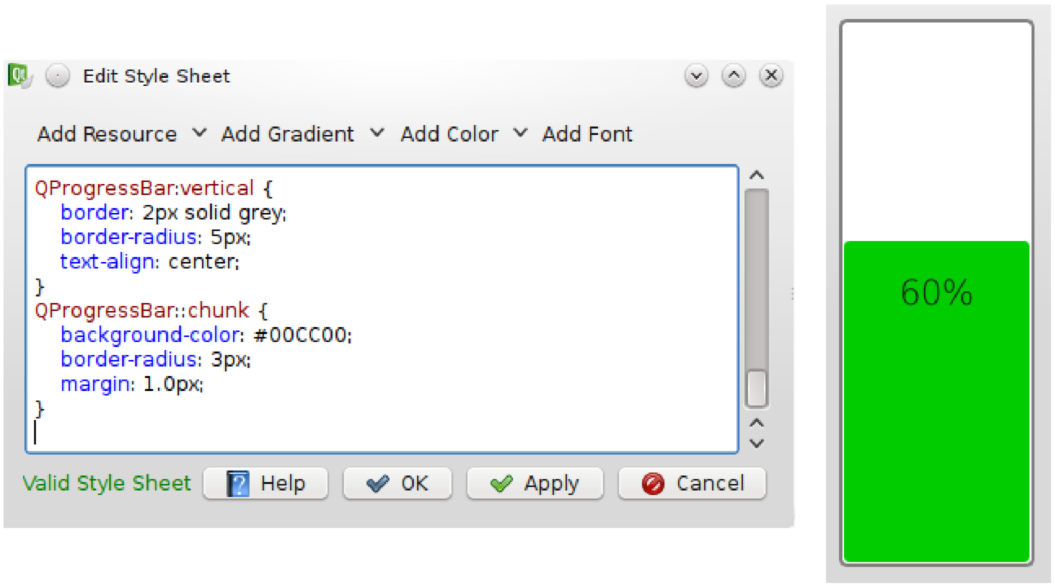
Figure 6.1.: Resizing the program from 640x480 to 800x600 pixels

6.1.1. Resizeability

The program is designed to run on a 720x480 resolution as this is the native resolution of the screen installed in the car. However, it supports resolutions down to 640x480. All higher screen resolutions and aspect ratios are also supported, although the program looks best on screens with up to 800x600 pixels (see figure 6.1). The resizeability of the program makes it easy to use on future cars with different hardware.

6.1.2. Style Sheets

To configure the appearance of the elements of the user interface, Qt Style Sheets are used. The concepts and syntax of the Qt Style Sheets are closely related to the HTML Cascading Style Sheets (CSS) [css10]. When applied to a layout, style sheets specify the appearance of all elements of a certain type and are inherited by objects on lower levels. Qt Style Sheets can be saved in files and loaded during runtime to allow customisation by the user. Figure 6.2 shows the configuration of the battery bar appearance. The first paragraph adds a border with rounded corners and centres the percentage label. This paragraph only refers to vertical bars, such as the battery bar and the current bars of the program. The second paragraph specifies the appearance of the chunk of all progress bars.

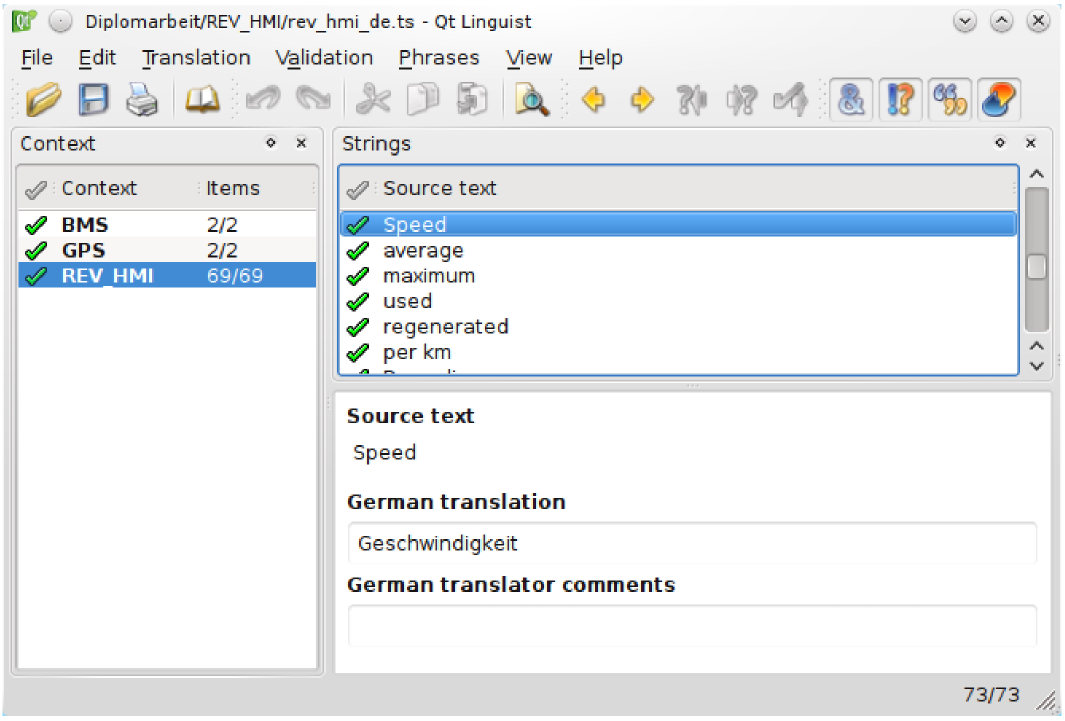


```
QProgressBar:vertical {
   border: 2px solid grey;
   border-radius: 5px;
   text-align: center;
}
QProgressBar::chunk {
   background-color: #00CC00;
   border-radius: 3px;
   margin: 1.0px;
}
```

Figure 6.2.: The style sheet editor for the battery bar

6.1.3. Translation

The Qt development framework includes powerful internationalization features that allow the language to be changed during runtime and new languages to be added to an existing program. The translation can be done entirely independent from the source code and is deployed in separate files. Fallback mechanisms exist for partial translations. Figure 6.3 shows the Qt Linguist program for editing the German translation of the program and in figure 6.4 the result can be seen.



Context	Items
BMS	2/2
GPS	2/2
REV_HMI	69/69

Source text
Speed
average
maximum
used
regenerated
per km

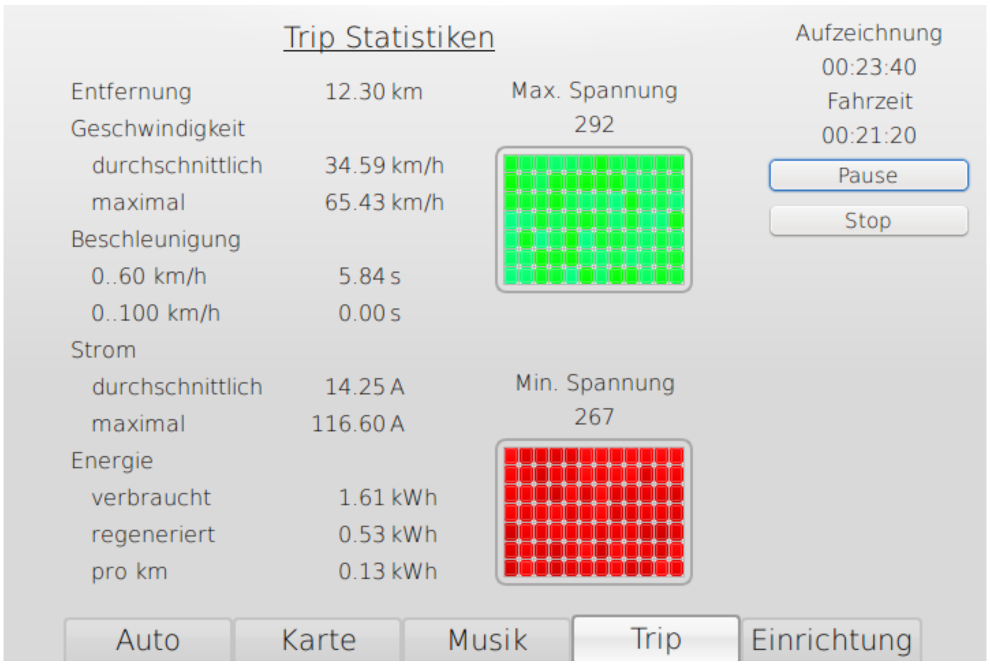
Source text
Speed

German translation
Geschwindigkeit

German translator comments

73/73

Figure 6.3.: The Qt Linguist translation program



Trip Statistiken

Entfernung	12.30 km	
Geschwindigkeit		
durchschnittlich	34.59 km/h	
maximal	65.43 km/h	
Beschleunigung		
0..60 km/h	5.84 s	
0..100 km/h	0.00 s	
Strom		
durchschnittlich	14.25 A	
maximal	116.60 A	
Energie		
verbraucht	1.61 kWh	
regeneriert	0.53 kWh	
pro km	0.13 kWh	

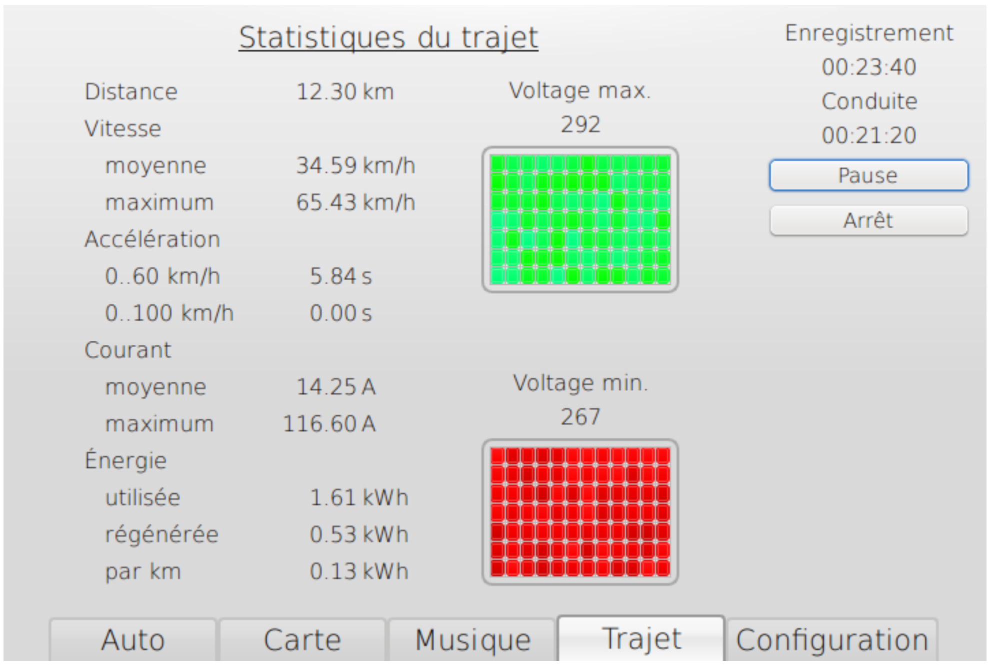
Max. Spannung 292

Min. Spannung 267

Aufzeichnung 00:23:40

Fahrzeit 00:21:20

Figure 6.4.: The German version of the program



Statistiques du trajet

Distance	12.30 km	
Vitesse		
moyenne	34.59 km/h	
maximum	65.43 km/h	
Accélération		
0..60 km/h	5.84 s	
0..100 km/h	0.00 s	
Courant		
moyenne	14.25 A	
maximum	116.60 A	
Énergie		
utilisée	1.61 kWh	
régénérée	0.53 kWh	
par km	0.13 kWh	

Voltage max. 292

Voltage min. 267

Enregistrement 00:23:40

Conduite 00:21:20

Figure 6.5.: The French version of the program

6.2. Car Tab

The car tab displays general information about the status of the car. On the upper left side is the current speed of the car, bellow which is the estimated remaining distance, and on the lower left side is the current time. Speed and time are from the GPS signal, and the remaining distance is calculated by the logging module. On the right side is the overall voltage of the battery, the charging bar that displays the percentage of charge, and the bars for the accelerating and regenerating current. The accelerating current points in the driving direction, whereas the regenerating current points backwards (see figures 6.6 and 6.7). The bars display the current relative to the maximum charging and discharging current defined for the battery cells. The precise value for the current is in between these bars, with negative values for a consumption of energy and positive values for regenerating or charging. At the centre of the car tab is the outline of the Lotus with a battery pack in the rear. Every box in this pack symbolizes a battery cell, and the colour changes continuously with the voltage of this cell. Fully charged cells are green, turn yellow during discharging and become red when they are empty. Completely discharged cells are dark red, overcharged cells are blue and cell icons that are unused are white.

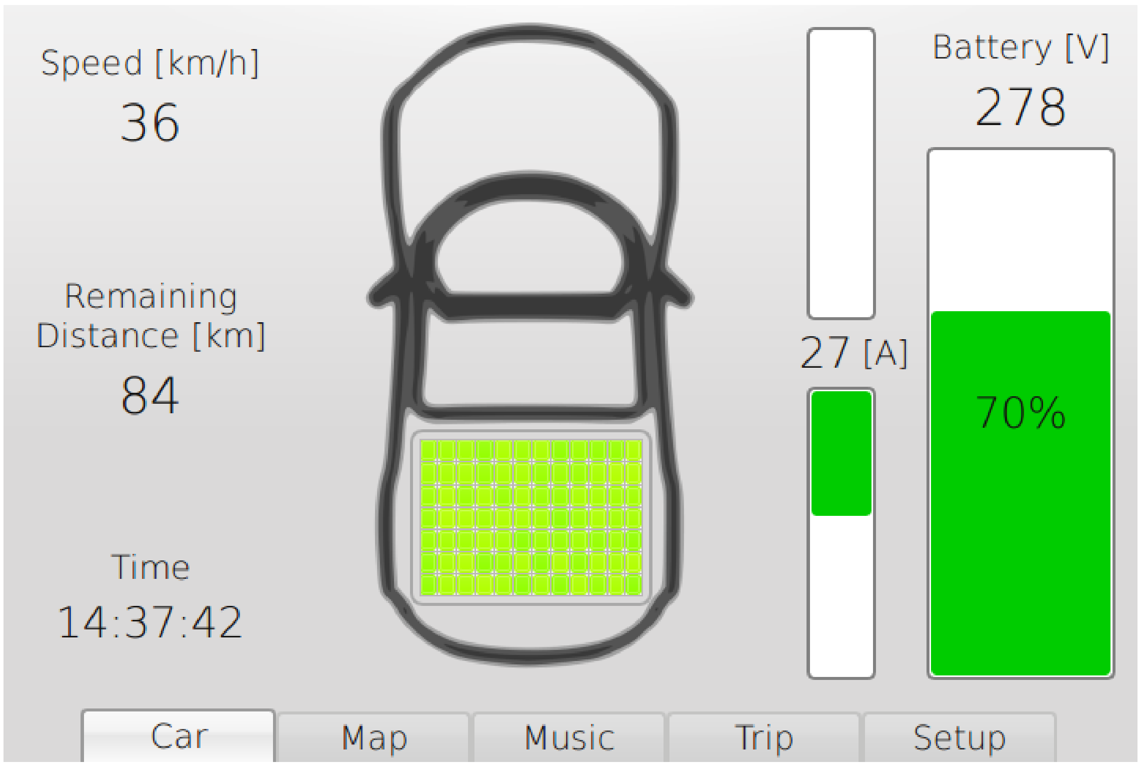


Figure 6.6.: The main tab of the program

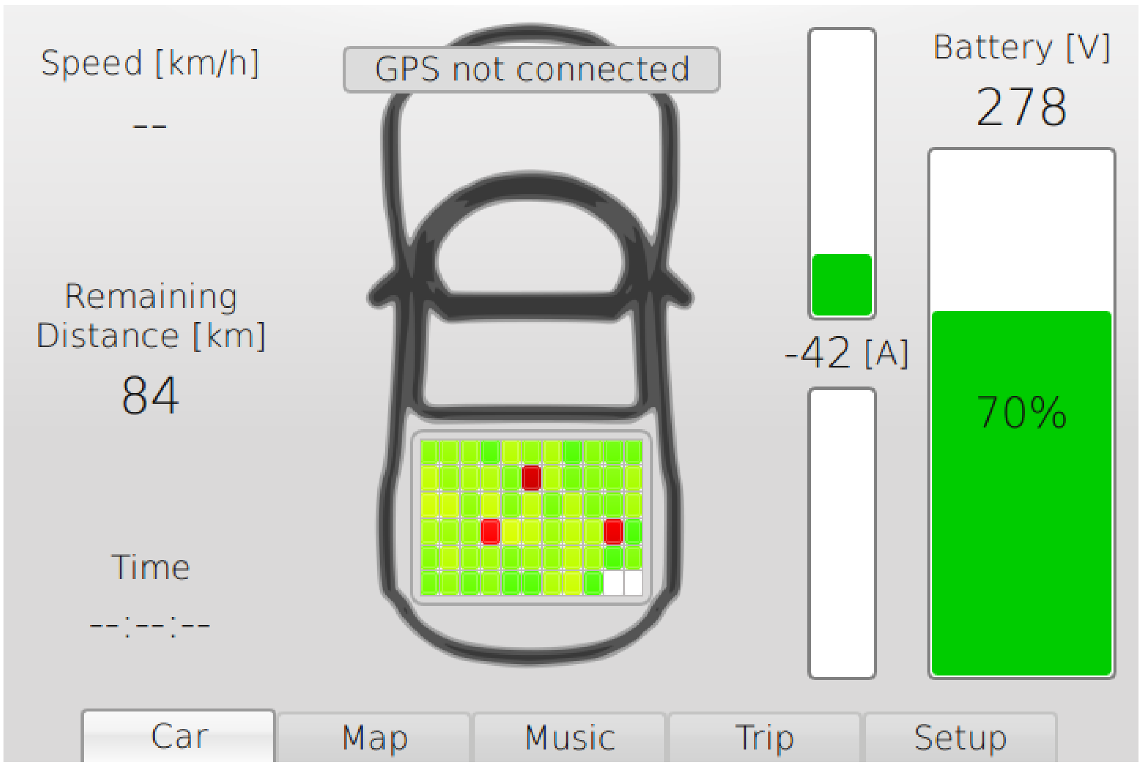


Figure 6.7.: The program with a disconnected GPS device and three faulty battery cells

In figure 6.7, the program displays a warning that the GPS device is not connected. It can also display warnings when the BMS is not connected or when one of the devices is not responding. The battery cell voltages are less balanced in this figure compared to figure 6.6, therefore the colours are in a wider range between green and yellow. Three cells have significantly lower voltages, which indicates damage. These cells should be checked and replaced if necessary.

6.3. Map Tab

This tab displays the position of the car on the map described in chapter 5.4. The buttons on the right control zooming in and out of the map; the first checkbox centres the map to the current position of the car. The next checkbox activates a night mode with inverted colours to improve ergonomics for nighttime driving. If the third box is checked, the current trip is shown on the map, and the boxes below activate the display of the test track and show comments if available. The button `Load Track` opens a file dialogue for the user to choose another track file. This file must be in GPX format [gpx04] and can be from a previous test drive or created by any other program. The first bar at the bottom of the screen displays the precision of the position measurement, which is calculated from the horizontal dilution of precision (HDOP). The HDOP rates the geometric inaccuracy of a GPS position due to a low number of visible satellites or unfavourable satellite positions. The minimum HDOP value is 1, higher values mean less accurate measurements. To make this number more intuitive, the precision displayed is the multiplicative inverse of the HDOP in percent. The second bar shows the number of visible satellites or, if the program has no GPS fix, a zero.

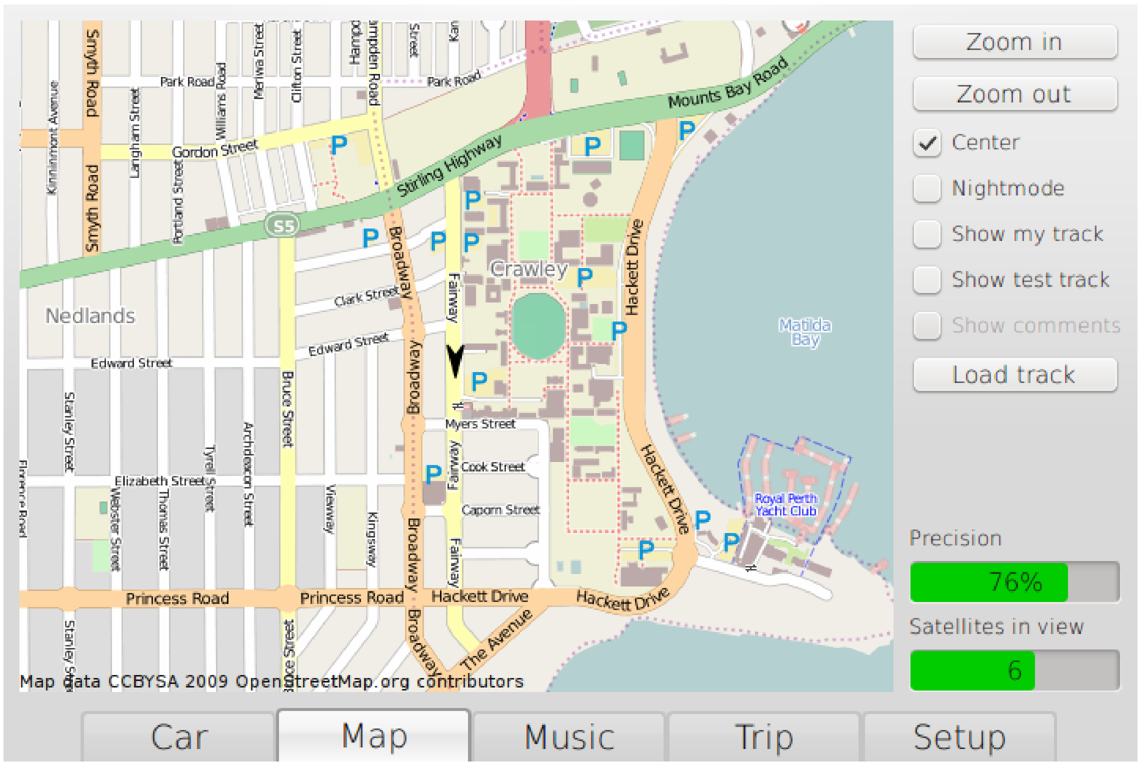
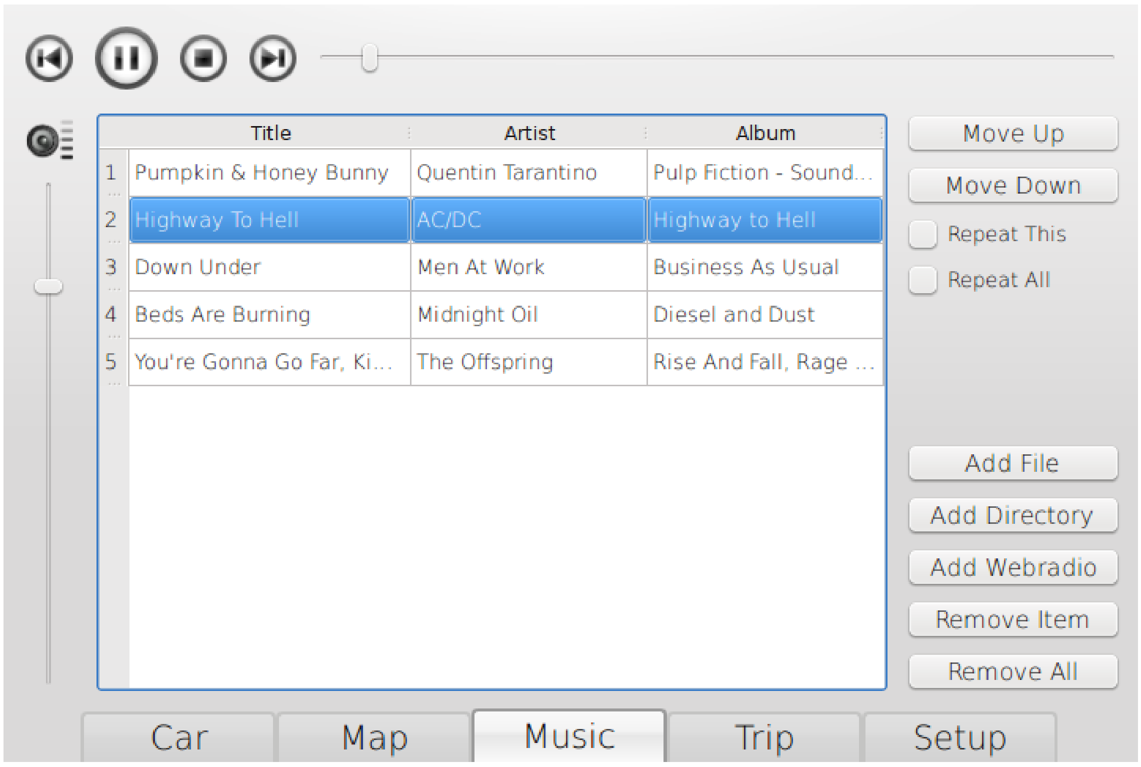


Figure 6.8.: The map view

6.4. Music Tab

The music tab controls the player functions of the program (figure 6.9). The buttons on the upper left side control the playback; the icons are the standard icons for play, pause, stop and skip. The vertical slider on the left adjusts the volume, and the button on top of it mutes the music. The horizontal slider displays the current position of the song and allows seeking. The central part of this tab is the playlist, which contains the songs that are currently being played. The song that is being played at the moment is highlighted. The first two buttons on the right move the current song up and down in the playlist. If the first checkbox is selected, the current song is infinitely repeated; if the second box is ticked the whole playlist is repeated.
The third button adds one or more files to the playlist, while the button below adds all files in a directory. The button `Add Websradio` adds a webstream from a radio station. The available webstreams are loaded from the configuration file to enable the user to add new radio stations. To avoid high traffic on the 3G connection, this feature is protected by a password. The last two buttons remove the selected item or all items from the playlist.

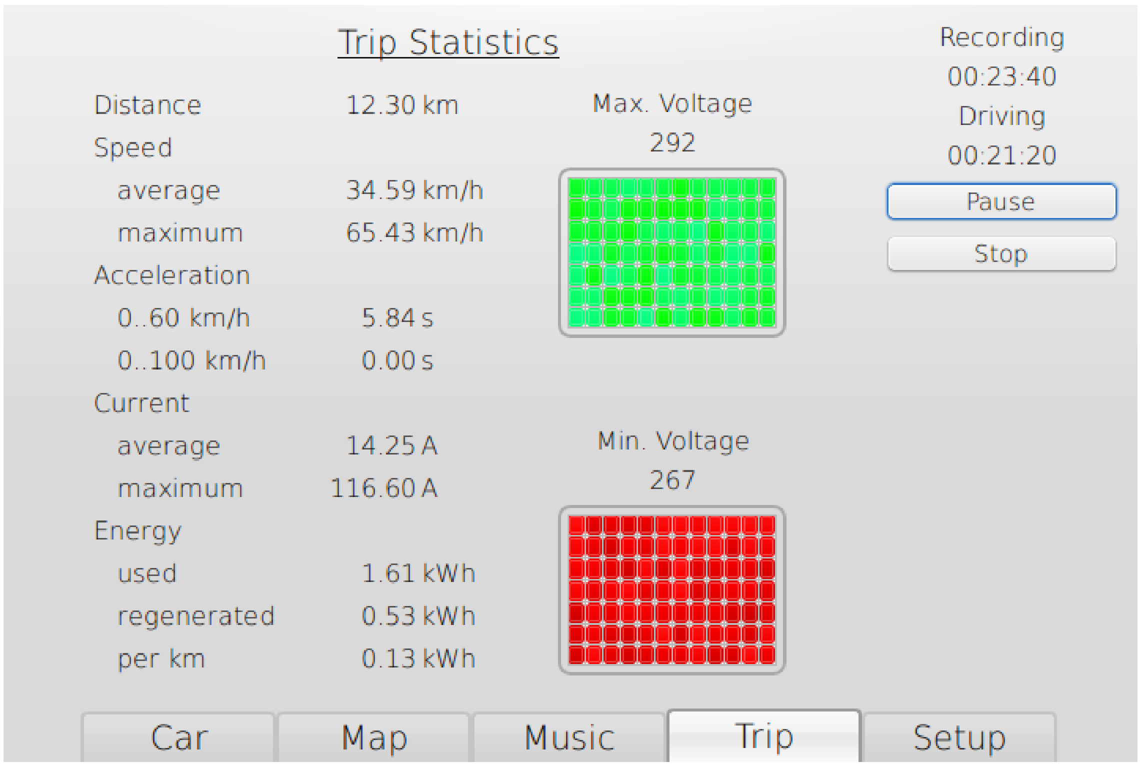


	Title	Artist	Album
1	Pumpkin & Honey Bunny	Quentin Tarantino	Pulp Fiction - Sound...
2	Highway To Hell	AC/DC	Highway to Hell
3	Down Under	Men At Work	Business As Usual
4	Beds Are Burning	Midnight Oil	Diesel and Dust
5	You're Gonna Go Far, Ki...	The Offspring	Rise And Fall, Rage ...

Figure 6.9.: The music player functions

6.5. Trip Tab

In this tab, all the information about the current trip is displayed. By default, the trip starts 10 seconds after the program is loaded. It can be paused and continued at any time with the first button. The second button stops recording and finishes the trip. Above the two buttons is the time elapsed since the start of the trip and the time the car was actually driving. The statistics start with the distance of the trip. The average speed is the distance divided by the driving time, the maximum speed is the fastest speed during the trip. The acceleration values are the times of the fastest acceleration from zero to 60 and 100 km/h, respectively. If the maximum speed is below those speeds, the acceleration times are zero. The energy used is the integrated current since the start of the trip, and the energy regenerated is only the integrated current in positive direction. The energy per km is the energy used divided by the trip distance. In the centre of this tab, the maximum and minimum voltage of the battery pack, as well as the maximum and minimum cell voltages, are displayed. The colours of the cell voltages are equivalent to those for the current cell voltages in the main tab.


Trip Statistics

Distance	12.30 km
Speed	
average	34.59 km/h
maximum	65.43 km/h
Acceleration	
0..60 km/h	5.84 s
0..100 km/h	0.00 s
Current	
average	14.25 A
maximum	116.60 A
Energy	
used	1.61 kWh
regenerated	0.53 kWh
per km	0.13 kWh

Max. Voltage
292

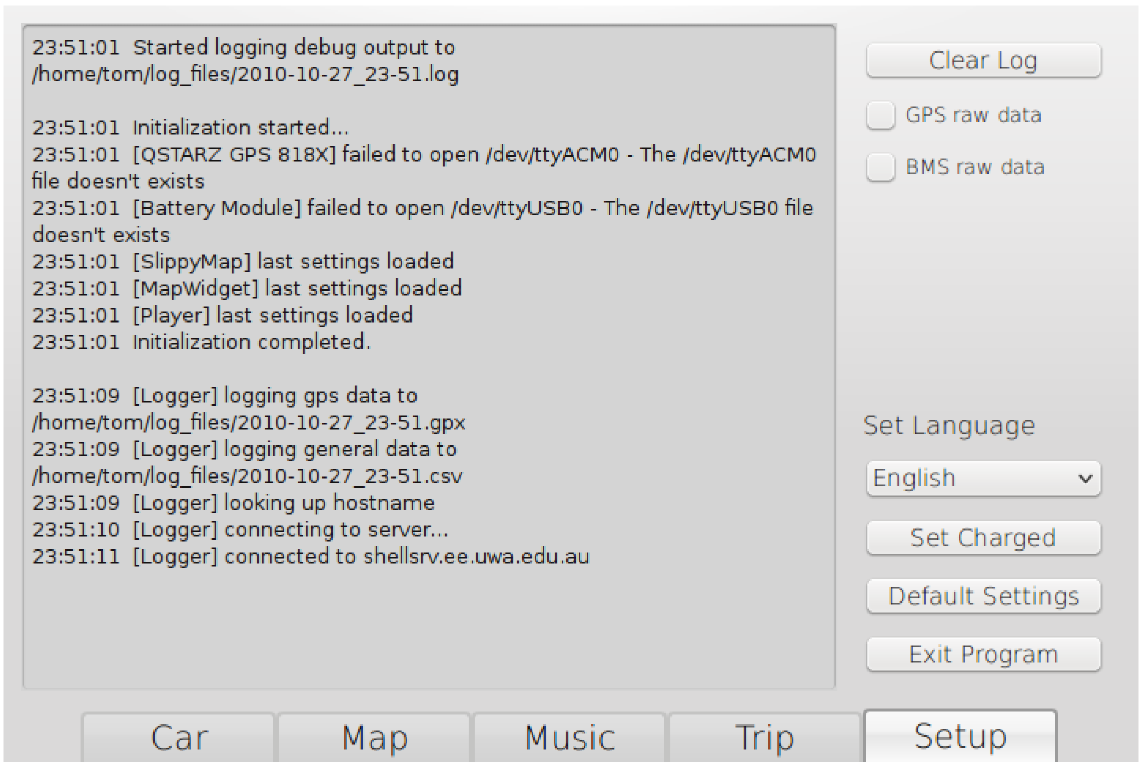
Min. Voltage
267

Recording
00:23:40
Driving
00:21:20

Figure 6.10.: The trip statistics

6.6. Setup Tab

The setup tab shows the debug and error messages to provide the user with all relevant information about the status of the program. This is especially helpful for the development of the program to trace back errors during test drives. The `Clear Log` button clears all messages in the user interface but leaves the log files untouched. The two checkboxes enable the logging of GPS and BMS raw data to trace back malfunctions in these modules. These boxes are unchecked by default as these settings increase the size of the log files substantially. The language selection drop-down list changes the language of the program. The available languages are generated from the translation files that are in the program folder. Therefore, installing a new language only requires copying the translation file to the program folder. The `Set Charged` button resets the battery capacity to the fully charged value specified for the battery type. The button `Default Values` resets all settings of the program to their default values. The last button exits the program when running in full-screen mode.



```
23:51:01  Started logging debug output to
/home/tom/log_files/2010-10-27_23-51.log

23:51:01  Initialization started...
23:51:01  [QSTARZ GPS 818X] failed to open /dev/ttyACM0 - The /dev/ttyACM0
file doesn't exists
23:51:01  [Battery Module] failed to open /dev/ttyUSB0 - The /dev/ttyUSB0 file
doesn't exists
23:51:01  [SlippyMap] last settings loaded
23:51:01  [MapWidget] last settings loaded
23:51:01  [Player] last settings loaded
23:51:01  Initialization completed.

23:51:09  [Logger] logging gps data to
/home/tom/log_files/2010-10-27_23-51.gpx
23:51:09  [Logger] logging general data to
/home/tom/log_files/2010-10-27_23-51.csv
23:51:09  [Logger] looking up hostname
23:51:10  [Logger] connecting to server...
23:51:11  [Logger] connected to shellsrv.ee.uwa.edu.au
```

Figure 6.11.: The administrative functions of the program

7. Evaluation

The two most important tasks of the program are the monitoring of the battery to prevent damages to the cells, and the recording of data during the test drives.

7.1. Battery Cell Monitoring

The monitoring of the battery cells has already been proven to be useful during the first test drives when one cell displayed an unusual behaviour. During charging, the voltage of this cell was notably higher and during driving the voltage dropped faster than all the other cells (see figure 7.1). This observation led to an inspection of this cell with the result that a loose connection was detected, which had caused a higher electric resistance. If unnoticed, this connection would have decreased the efficiency of the car and in the worst case, could have overheated and caused a fire.

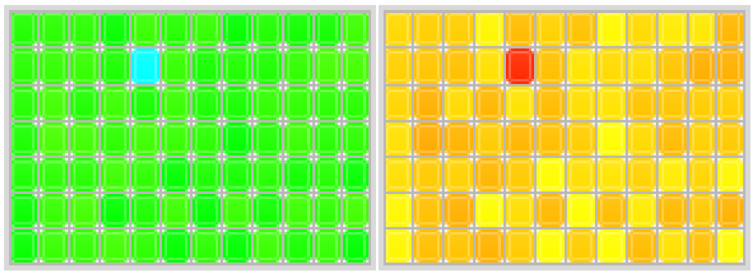


Figure 7.1.: Faulty Cell (left side charging, right side discharging)

7.2. Test Track Recording

During test drives, two files are created. The first file is in the GPX format described in chapter 5.3.1 and stores the test track as shown in figure 7.2. This information is needed for the evaluation to see if the test drive was in the city or on a highway, and to identify test drives that took the same route and can therefore be compared.

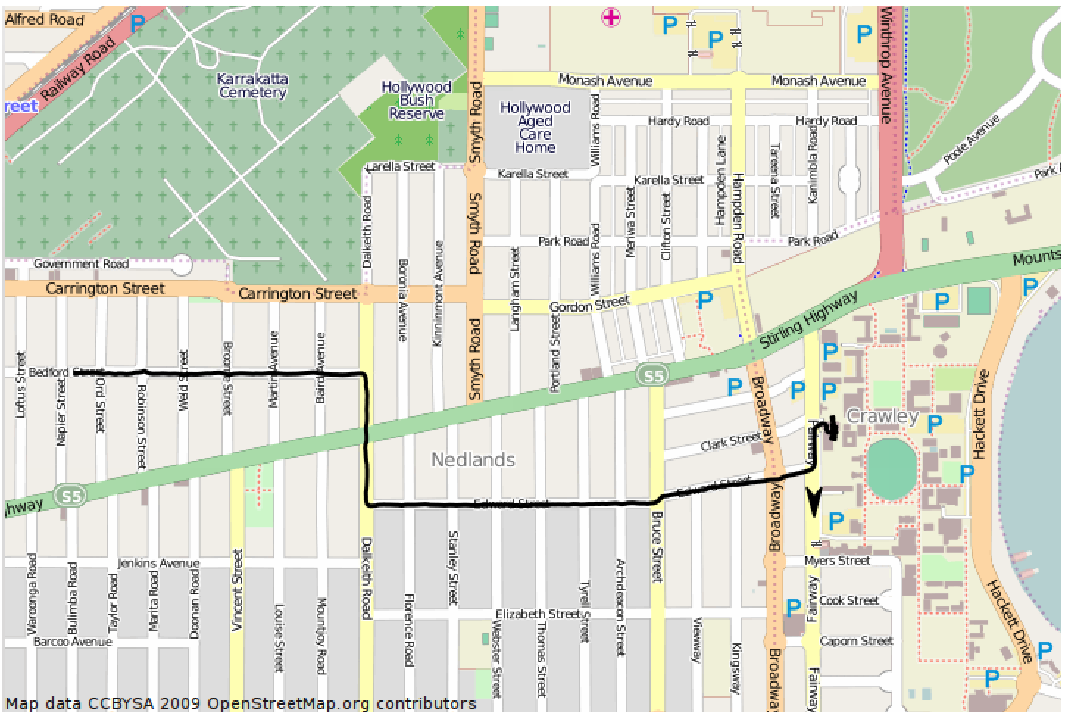


Figure 7.2.: Recording of a test drive

7.3. Vehicle Data Recording

The second file stores all vehicle data in the CSV file format (see chapter 5.3.2). This data includes the speed of the car, the time and distance of the trip, and all battery values. In figure 7.3 the voltages of the battery cells are displayed. As the voltages for each cell are updated only once every 15 seconds, the cell voltages show large differences. The real value is the voltage at the moment when the cell voltage is updated. To illustrate this, the black line gives the expected cell voltage. This line is calculated from the battery pack voltage, which is measured with a separate sensor and divided by the number of battery cells. In figure 7.3 this value is lower than expected for the first 10 to 15 seconds of the trip, which might be due to incorrect battery pack voltage values. This problem is subject to further inquiries that have not been completed at the time of completion of this thesis.

In figure 7.4 the relationships between speed, voltage and electric current are shown. Most interesting is the voltage drop during acceleration and the voltage increase during regenerative braking. Future work could analyse the influence of changes in altitude caused by driving up or down a hill, the usage of electricity consumers like heating or headlights, and the number of passengers in the car. Detailed evaluation of the data acquired during the test drives is subject to evaluation in other studies of the REV Project.

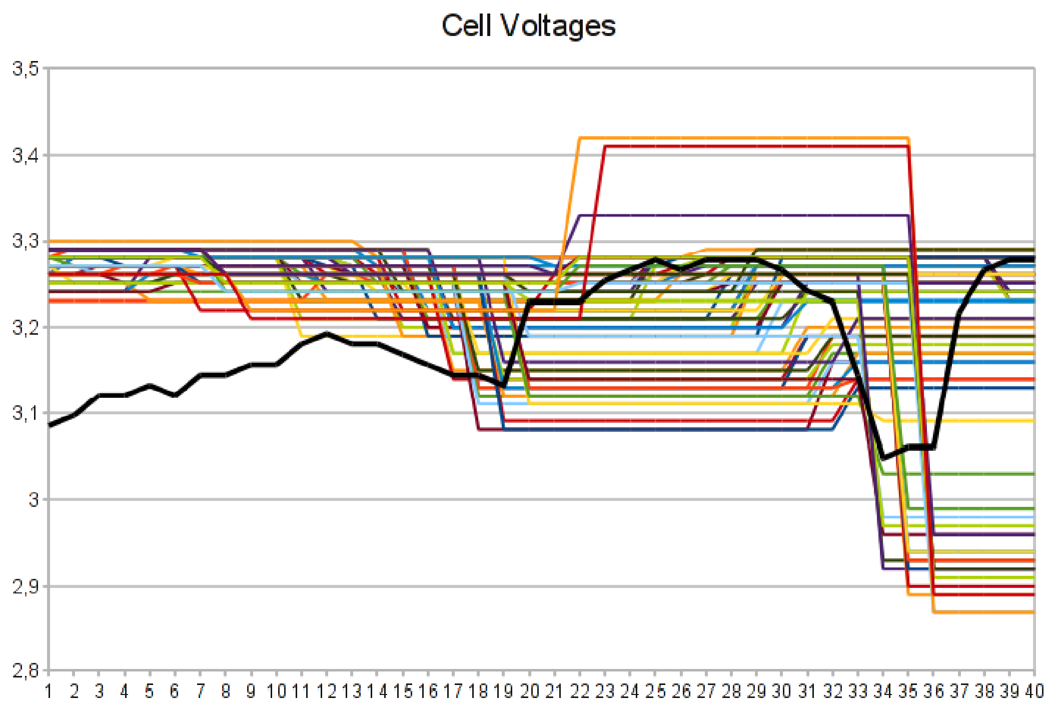


Figure 7.3.: Voltages of the battery cells

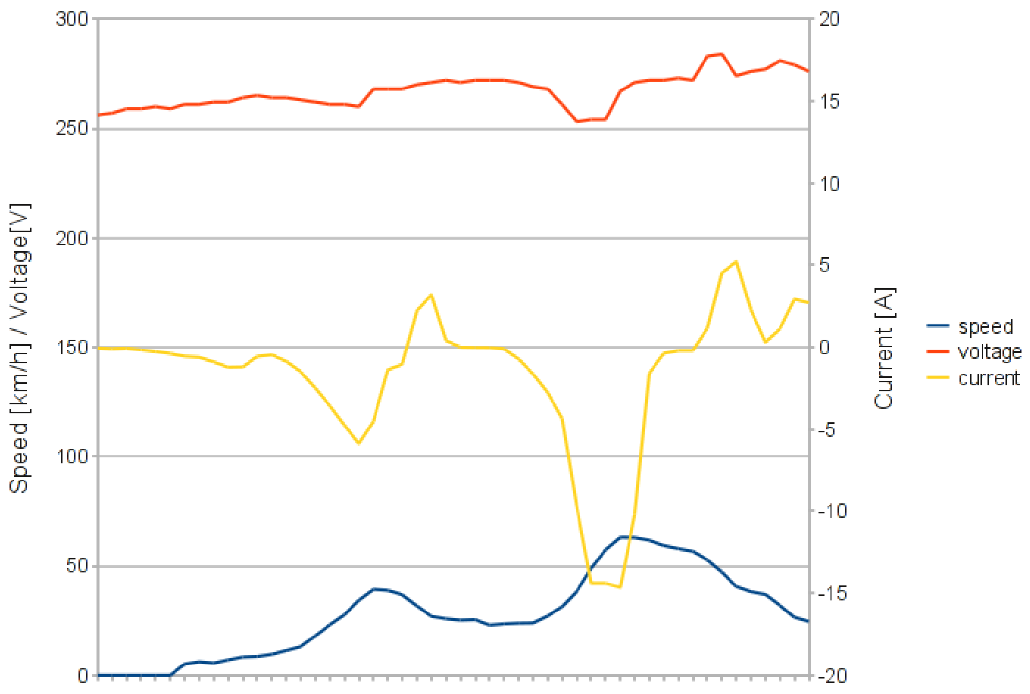


Figure 7.4.: Voltage and current changing with the speed of the car

8. Conclusion and Future Work

The adaptation of the driver information system to the requirements of electric vehicles proved to be necessary for the safety and convenience of the driver. The monitoring of the battery cells helps to prevent damage to the car and the estimation of the remaining distance with the current battery charge helps the driver to safely reach his/her destination. Beyond this, the REV Project can take advantage of the recording of vehicle data to analyse and optimise the performance of the car. Additional features, such as the live tracking of the car with the software developed by J. Pearce [Pea10], make it a powerful tool for the work of the project.

One of the main achievements of this work is the realisation of maximum reusability. The program runs on all common operation systems and hardware platforms, supports many different screen resolutions, and can be translated into any language. Furthermore, the program is designed to be easily extensible and contains a powerful error logging framework to trace back problems.

As part of the REV Project, this work will be subject to further development. This could include some of the following aspects:

- Switching to the latest version of Qt: when work on this thesis was started, Qt 4.6 was the most recent version. Since then, Qt 4.7 has been released, which has improved touchscreen support and a new audio API. It also comes with the new Qt Mobility which has, amongst other things, a new API for location based services.
- Integration of a full-fledged navigation system: this includes calculation of the route, re-calculation if the the driver leaves this route, and speech output of the directions.
- FM radio hardware integration: this has advantages over web radio as it does not need a constant internet connection and therefore also works outside cities, does not generate expensive internet traffic, and often has more local stations available.
- Usage of a tablet PC instead of the car PC plus touchscreen: this would have the advantage of perfect integration of all hardware, including GPS, WLAN, 3G internet, acceleration sensors, compass, sound and Bluetooth.
- Switching to an operating system that is designed for touchscreen usage: for example, MeeGo IVI (In Vehicle Infotainment) [mee10], a system developed by Intel and Nokia. It is based on Linux and has a front-end written in Qt. This operating system would

offer interesting features like navigation, speech recognition, Bluetooth integration and many others.

Finally, the author wishes to state that working on the REV Project and the development of this user interface has proven to be an interesting and challenging experience. The author hopes that this work will be helpful for all projects based on the data collected with this software.

A. Appendix

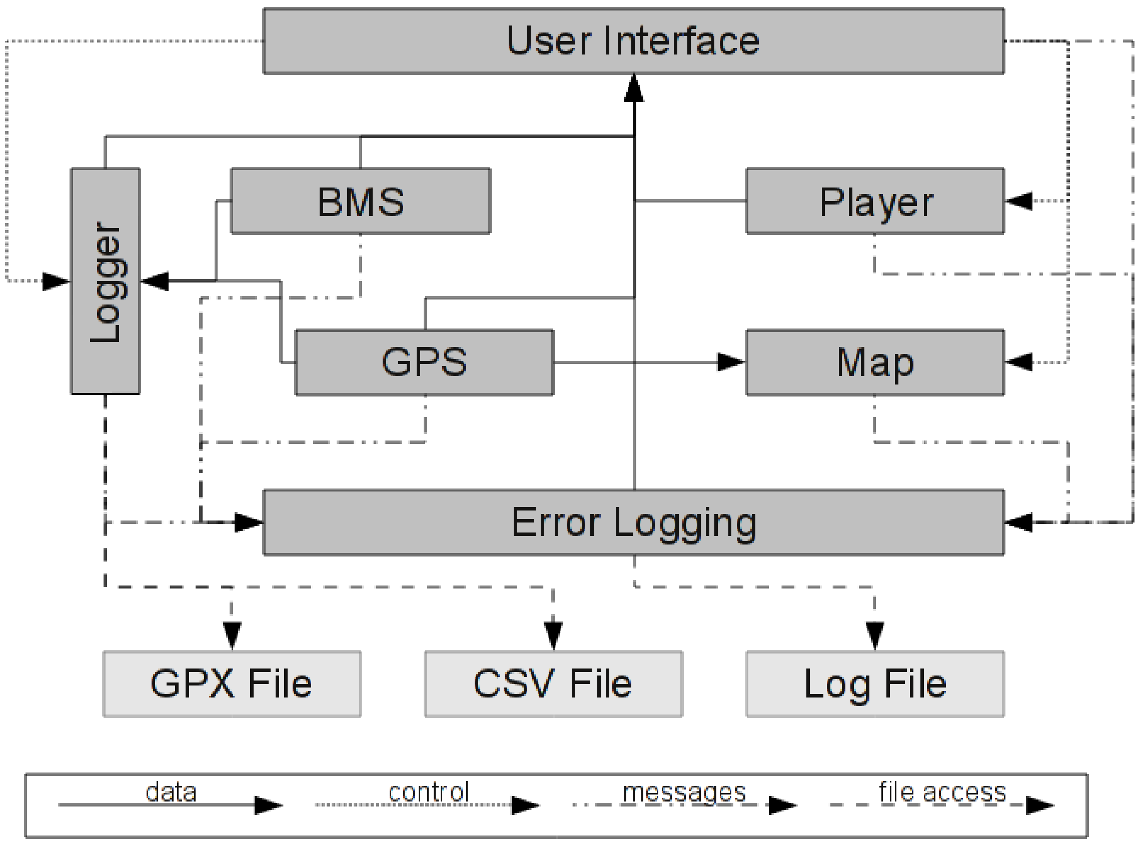


Figure A.1.: The structure of the program

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx version="1.1" creator="The REV Project" xmlns="http://www.topografix.com/GPX/1/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.topografix.com/GPX/1/1
    http://www.topografix.com/GPX/1/1/gpx.xsd">
    <trk>
        <name>Lotus Test Drive</name>
        <number>16</number>
        <trkseg>
            <trkpt lat="-31.94999000" lon="115.98373000">
                <ele>29.60</ele>
                <time>2010-09-09T03:43:42Z</time>
                <hdop>0.87</hdop>
                <extensions>
                    <speed>0.00</speed>
                </extensions>
            </trkpt>
            <trkpt lat="-31.94999167" lon="115.98373000">
                <ele>29.60</ele>
                <time>2010-09-09T03:43:43Z</time>
                <hdop>0.90</hdop>
                <extensions>
                    <speed>0.00</speed>
                </extensions>
            </trkpt>
            <trkpt lat="-31.94999167" lon="115.98373000">
                <ele>29.60</ele>
                <time>2010-09-09T03:43:44Z</time>
                <hdop>0.90</hdop>
                <extensions>
                    <speed>0.00</speed>
                </extensions>
            </trkpt>

            [...]

            <trkpt lat="-31.95000167" lon="115.98374667">
                <ele>21.90</ele>
                <time>2010-09-09T04:00:56Z</time>
                <hdop>0.89</hdop>
                <extensions>
                    <speed>0.00</speed>
                </extensions>
            </trkpt>
        </trkseg>
    </trk>
</gpx>
```

Listing A.1: GPX file example

```
time; distance; speed; altitude; voltage; current; capacity remaining; energy used; energy regen; V1; V2; V3; V4; V5
1.000; 0.000; 0.000; 31.200; 256.000; -0.070; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
2.000; 0.000; 0.000; 31.300; 257.000; -0.110; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
3.000; 0.000; 0.000; 31.400; 259.000; -0.080; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
4.000; 0.000; 0.000; 31.500; 259.000; -0.170; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
5.000; 0.000; 0.000; 31.600; 260.000; -0.270; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
6.000; 0.000; 0.000; 31.500; 259.000; -0.390; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
7.000; 0.001; 5.130; 31.600; 261.000; -0.560; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
8.000; 0.002; 6.167; 31.700; 261.000; -0.610; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
9.000; 0.004; 5.593; 31.900; 262.000; -0.900; 59.970; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
10.000; 0.005; 7.038; 32.000; 262.000; -1.230; 59.969; 0.000; 0.000; 3.280; 3.280; 3.280; 3.280; 3.280
11.000; 0.008; 8.353; 32.200; 264.000; -1.210; 59.969; 0.000; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
12.000; 0.010; 8.612; 32.300; 265.000; -0.570; 59.969; 0.000; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
13.000; 0.013; 9.612; 32.300; 264.000; -0.470; 59.969; 0.000; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
14.000; 0.016; 11.316; 32.300; 264.000; -0.860; 59.968; 0.001; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
15.000; 0.019; 13.112; 32.300; 263.000; -1.510; 59.968; 0.001; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
16.000; 0.024; 18.057; 32.300; 262.000; -2.500; 59.967; 0.001; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
17.000; 0.029; 23.039; 32.200; 261.000; -3.570; 59.966; 0.001; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
18.000; 0.035; 27.650; 32.200; 261.000; -4.760; 59.965; 0.001; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
19.000; 0.045; 34.281; 32.000; 260.000; -5.870; 59.964; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
20.000; 0.055; 39.244; 31.800; 268.000; -4.520; 59.962; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
21.000; 0.066; 38.781; 31.700; 268.000; -1.380; 59.962; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
22.000; 0.077; 36.762; 31.000; 268.000; -1.040; 59.961; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
23.000; 0.086; 31.669; 30.600; 270.000; 2.230; 59.962; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
24.000; 0.094; 26.947; 30.400; 271.000; 3.190; 59.962; 0.002; 0.000; 3.280; 3.280; 3.190; 3.260; 3.250
[...]
```

Listing A.2: CSV file example (5 battery cells)

```
[General]
language=English

[gps]
deviceName=QSTARZ GPS 818X
port=/dev/ttyACM0

[battery]
deviceName=Battery Module
port=/dev/ttyUSB0
lastCapacity=60
batteryType=TS_LYP60AHA

[map]
tilesStoredLocally=true
tilesStorageDirectory=tiles
maxWidth=1920
maxHeight=1200
latitude=-31.979283
longitude=115.815852
zoom=15
centered=false
invert=false
myTrackVisible=true
testTrackVisible=false
trackFile=testtrack.gpx
testTrackCommentsVisible=false
minZoom=1
maxZoom=18
baseURL=http://tile.openstreetmap.org
myPosLat=-31.979283
myPosLng=115.815852
myDir=0

[player]
repeatItem=false
repeatAll=false
selectedItem=-1
volume=0.8

[logging]
trkNumber=0
logDir=log_files
logName=yyyy-MM-dd_hh-mm
logInterval=1000
trkName=Lotus Test Drive
unitIMEI=123456789012345
serverName=shellsrv.ee.uwa.edu.au
serverPort=2001
```

Listing A.3: INI file with the default settings

Command	1. value (1 Byte)	2. value (2 Bytes)	3. value (1 Byte)	Example
A	current direction: 0 - discharging, 1- charging	battery current [A/10]	(not used)	A,1,573,0
B	battery number 1 - pack, 2 - auxiliary	battery voltage [V] or auxiliary voltage [V/100]	(not used)	B,1,276,0
C	charging mode: 0 - manual, 1- condition 1, 2 - condition 2	(not used)	(not used)	C,1,0,0
H	maximum setpoint [V/10]	(not used)	cell count	H,38,0,83
L	minimum setpoint [V/10]	(not used)	cell count	L,26,0,83
M	cell number	min. cell voltage [V/10]	max. cell voltage [V/10]	M,17,28,36
O	bms status: 0 - offline, 1- online	(not used)	(not used)	O,1,0,0
V	cell number	cell voltage [V/100]	min. cell voltage [V/10] if above max. setpoint: shunting current [A/10]	V,17,320,27

Table A.1.: Command list for the battery management system

Bibliography

[bmw10] BMW Group PressClub, 2010. URL https://www.press.bmwgroup.com/. (Cited on page 23)

[che10] 2011 Volt - Electric Car - Chevrolet.com, 2010. URL http://www.chevrolet.com/future-vehicles/volt/. (Cited on page 25)

[css10] Cascading Style Sheets, 2010. URL http://www.w3.org/Style/CSS/. (Cited on page 58)

[dai10] Daimler Global Media Site, 2010. URL http://media.daimler.com/. (Cited on page 24)

[eye08] Eyebot - Online Documentation, 2008. URL http://robotics.ee.uwa.edu.au/eyebot/. (Cited on page 19)

[for10] SAE Collegiate Design Series, 2010. URL http://students.sae.org/competitions/formulaseries/. (Cited on page 16)

[gpx04] GPX 1.1 Schema Documentation, 2004. URL http://www.topografix.com/gpx/1/1/. (Cited on pages 45 and 63)

[Her09] T. Herzog. World greenhouse gas emissions in 2005. *World Resources Institute*, 2009. (Cited on page 15)

[Hid10] A. Hidaya. OpenStreetMap and Qt and (S60), 2010. URL http://labs.trolltech.com/blogs/2009/08/04/openstreetmap-and-qt-and-s60/. (Cited on page 51)

[ime10] International Mobile Equipment Identity, 2010. URL http://en.wikipedia.org/wiki/International_Mobile_Equipment_Identity. (Cited on page 48)

[map10] Mapnik C++/Python GIS Toolkit, 2010. URL http://mapnik.org/. (Cited on page 53)

[mee10] MeeGo In-Vehicle, 2010. URL http://meego.com/devices/in-vehicle. (Cited on page 71)

[min10] VoomPC-2 Enclosure for Car PC Applications, 2010. URL http://www.mini-box.com/VoomPC-Enclosure-2?sc=8&category=87. (Cited on page 31)

[nme01] GPS - NMEA sentence information, 2001. URL http://aprs.gids.nl/nmea/. (Cited on page 42)

[nme10] NMEA 0183 Standard, 2010. URL http://www.nmea.org/content/nmea_standards/nmea_083_v_400.asp. (Cited on pages 32 and 42)

[Pea10] J. Pearce. Electric Vehicle Telemetry. 2010. (Cited on pages 47, 50 and 71)

[pho10] Phonon Overview, 2010. URL http://doc.trolltech.com/4.6/phonon-overview.html. (Cited on page 55)

[qst10] QStarz BT-Q818X GPS Receiver, 2010. URL http://www.qstarz.com/Products/GPS%20Products/BT-Q818X-F.htm. (Cited on page 32)

[RG02] T. Ramabadran, S. Gaitonde. A tutorial on CRC computations. *Micro, IEEE*, 8(4):62–75, 2002. (Cited on page 48)

[robo6] RoBIOS Library Functions, 2006. URL http://robotics.ee.uwa.edu.au/eyebot/doc/API/library.html. (Cited on page 19)

[sli10] Slippy Map - OpenStreetMap Wiki, 2010. URL http://wiki.openstreetmap.org/wiki/Slippy_Map. (Cited on page 51)

[tes10] Tesla Motors, 2010. URL http://www.teslamotors.com/. (Cited on pages 21 and 22)

[til10] TileLite, 2010. URL http://bitbucket.org/springmeyer/tilelite. (Cited on page 54)

[Var09] D. Varma. Renewable Energy Vehicle Instrumentation: Graphical User Interface and Black Box. 2009. (Cited on pages 19 and 20)

[xml03] Extensible Markup Language (XML), 2003. URL http://www.w3.org/XML/. (Cited on page 45)

All links were last followed on November 1, 2010.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Thomas Walter)