THE UNIVERSITY OF WESTERN AUSTRALIA

FINAL YEAR PROJECT

21776987

Renewable Energy Vehicle's User Interface

Author:

Fangpeng Li

Supervisor:

Prof. Thomas BRÄUNL

Thesis submitted as part of the B.E. degree in the School of Electrical, Electronic and Computer Engineering, University of Western Australia

Glossary of Terms

API	Application Programming Interface
BMS	Battery Management System
DP	Douglas Peucker Algorithm
EV	Electric Vehicle
GPS	Global Positioning System
GUI	Graphical User Interface
HMI	Human Machine Interface
IDE	Integrated Development Environment
QT	QT Application Framework
REV	Renewable Energy Vehicle Project
RPi	Raspberry Pi
RPM	Revolutions Per Minute
SDK	Software Development Invironment

Abstract

The Renewable Energy Vehicle project (REV) is dedicated to design and develop environmentally sustainable vehicles, initiatively formed by the University of Western Australia. Among the numerous achievements of REV, the REV Eco is a road-licensed plug-in electric commuter car based on a Hyundai Getz, and the REV Racer is an electric sports car based on a Lotus Elise.

Embedded controllers for both vehicles were originally designed for replacing vehicle instruments like speedometer and tachometer, and displaying GPS information. After gradual development, modules like media player, battery management system (BMS), and route tracker were integrated into the controller.

This thesis focuses on upgrading both software and hardware that concern the User Interfaces (UI) on both vehicles to improve the performance and promote the capability. The last modification of the UI was conducted in 2014. Hence the first thing is to upgrade the development environment for the base program, and the operating system allocated on the controllers. During this process, a fair amount of necessary alters and changes was applied to the pre-existing code to eliminate the incompatibility between the previous UI program and the newly software environment. Similarly, a hardware of the controllers was changed from Raspberry Pi (RPi) Model B to RPi 3 Model B, the latest product of the Raspberry Pi series of single computer board. These upgrades are supposed to make the UI adapt to recent techniques and benefit its development in the future.

Contents

Introduction

1.1 Electric Vehicles

Today climate change is no longer a topic only for environmentalists but is now world-widely discussed by the general society. There is a common agreement that climate change is mainly due to greenhouse gas (GHG) emissions, especially the carbon dioxide (CO_2) emissions. As conventional internal combustion engine (ICE) vehicles contribute a significant part of the CO_2 emissions, it is increasingly necessary to find alternative transportation approaches. Therefore, electric vehicles (EV) which produce no GHG itselves have been developed as a suitable alternative to conventional vehicles using fossil fuel.

EV in this thesis refers to a battery-powered vehicle. There are usually one or more electric motors which use electrical energy stored in rechargeable batteries and propel the car[1]. Unlike a lot of people think that EV is a modern invention, it has a long history began in the early 19th century. In 1837, chemist Robert Davidson of Aberdeen invented the first EV. After decades of development, EVs were put into practical usage and once popular during the period from the late 19th century to early 20th century. Due to advances in ICEs, EVs were soon replaced by cheaper gasoline vehicles. However,

Modern commercial use of EVs were firstly found in 1987 in the USA. Those EVs used as taxies which served in New York were manufactured by the Philadelphian Electric Carriage and Wagon company[1]. In contrast to ICE vehicles, they produced less noise and needed no gear changes. Although the development and use of EVs were very limited due to restriction of battery costs and low fuel prices at the time, the interest in EVs gradually increased due to the increasing concerns about the growing fuel prices and the global warning crisis, and eventually met its burst brought by the Tesla Motors, a small up-start California car company who invented an EV sports car aka the Telsa Roadster in 2008. The success of this vehicle caught the eyes of the public and renewed the interest of the tycoons of the vehicle industry in EVs. Following this, more and more practical EVs, like Telsa Model S, Nissan Leaf, BMW i3 and Fiat 500e, were put into the market. According to the International Energy Agency (IEA)[1], the number of registered electric cars in 2016 exceeded 750 thousand all over the world, which was the highest number ever. The increasing demands of EVs indicates a bright prospect of researches in related fields.

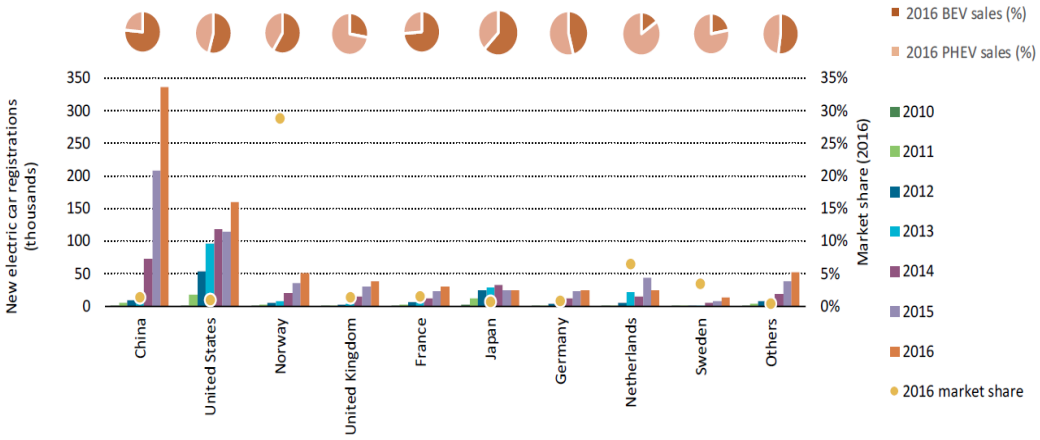


Figure 1.1: Electric car sales, market share, and BEV and PHEV sales shares in selected countries, 2010-2016 (Sources: IEA analysis based on EVI country submissions, complemented by EAFO (2017a), IHS Polk (2016), MarkLines (2017), ACEA (2017a, 2017b) and EEA (2017).)[2]

1.2 The REV Project

The UWA Renewable Energy Vehicle (REV) project is a project conducted by the University of Western Australia (UWA). It began as a restart of ealier research in to hydrogen fuelled vehicles, which had been discarded primarily due to the costs. The REV aims to prove the viability and 'revolutionise personal transport' by building zero emission vehicles, charged from renewable sources. As a part of this project, two REV plug-in electric conversion project were started, providing road-legal fully EVs for experimenting.

The REV Eco was the first plug-in electric conversion project that was started and completed in 2008. It is a five-seater commuter vehicle, based on a Hyundai Getz 2008 model, converted to electric drive by replacing the ICE with an electric drive motor, batteries wiring and instrumentation. The second conversion project conducted by UWA as well is the REV Racer. This car is two-seated high-performance car, based on a Lotus Elise S2, 2002 model. The conversion work of the REV Eco (aka Getz) and Racer (aka Lotus) were completed in 2008 and 2009, respectively.

Figure 1.2: REV Eco - The Converted Hyundai Getz. (Source: http://therevproject.com/)

Figure 1.3: REV Racer - The Converted Lotus Elise. (Source: http://therevproject.com/)

An important part of the REV is the development of the human machine interface (HMI) between users, especially drivers, and the aforementioned two EVs. Ideally, the HMI should provide a logging and user interface (UI) system handled by a device embedded into a targeted EV. Since originally raised in 2008 and firstly implemented by Ovens, this system has undergone several modifications and even re-implementations. The last update was carried out by Marcus in the year of 2013, resulting in upgrades of both software and hardware, and a complete rewrite of the graphic user interface (GUI).

System Analysis

2.1 REV Getz

It is basic demand that the driver should be informed the state of the car and get warned whenever the car is in an inappropriate condition that could harm their safety. To satisfy this need, it was the first time that the HMI system for the REV EVs (Getz and Lotus) was developed in 2008 by Ovens. This system was deployed to an EyeBot M6, an embedded controller developed at the University of Western Australia (UWA)[6]. It was primarily used to display important vehicle instruments including battery status, engine RPM (revolutions per minute) and speed, and allow data logging for analysis at the moment. As the EyeBot M6 consists of a LCD touch screen and an embedded controller, it offered the HMI system for the Getz the abilities of collecting a wide range of system parameters and displaying a customized user-friendly interface.

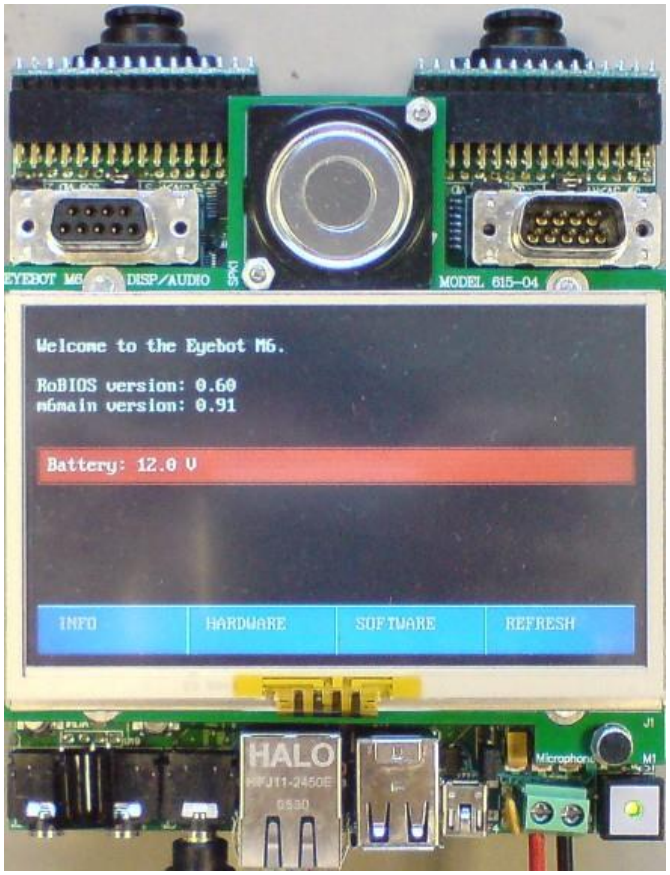


Figure 2.1: The Eyebot M6, the controller to control instrumentation in The REV Eco up until 2013.

In the year of 2009, due to the installation of EyeBot controller, the system was introduced more features like GPS tab and warning tab. After the modification done by Varma, the system was able to display a map centralized at the location of the car when switched to the GPS tab, and display warning information when switched to the warning tab. The warning information informed the driver something like low battery, the driver's door open or the driver's seatbelt unlatched. Additionally, a similar system was designed for the REV Lotus and installed on a VoomPC.

In 2010, Walter, who was responsible for developing the REV HMI system at that time, introduced the Nokia Qt C++ development framework to the project due to the consideration of the program portability and system performance. On one hand, programs developed with Qt framework can be deployed to multiple platforms including Windows, Linux, Mac OS and most importantly, some embedded systems. On the another hand, although C++ is not the most suitable programming language in the software development for embedded device, it provides a relatively better performance (compared with Java, Python, and etc.) while providing a better programming experience than C language. Nevertheless, Qt provides an open source version and thereby this change saved the potential cost of using a commercial integrated development environment (IDE). Another critical modification done by Walter was to introduce the music module, which was used to play audio files.

Before this project, the last system installed on the REV Getz was implemented by Marcus in 2013. The modifications generally consist of the upgrade of both software and hardware, and fixing existing defects in the program. The program was developed and compiled with Qt 5.1 (the latest Qt version at the moment), and then installed in the Raspberry Pi, a single board computer. Based on the recommendation raised by Gabriel in 2013, Raspberry Pi were used as embedded controllers for the REV Getz and REV Lotus, taking place of previous EyeBot M6 and VoomPC. This change remarkably improved the system performance.

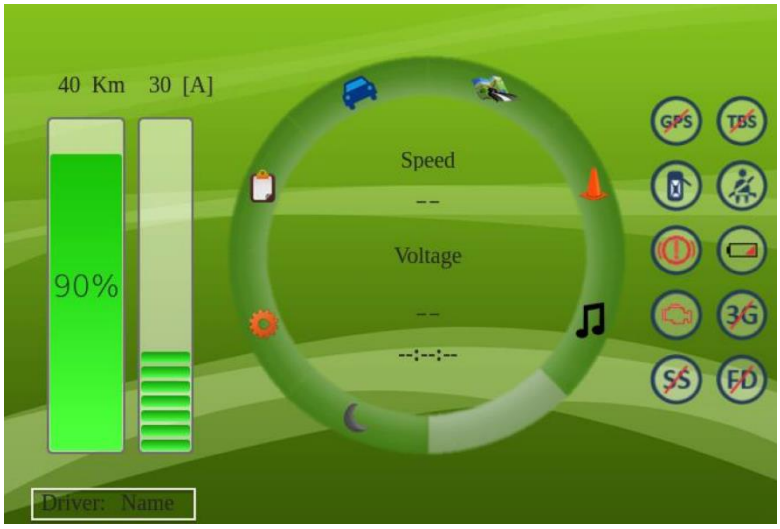


Figure 2.2: The GUI design by Gabriel Feng. (2013)

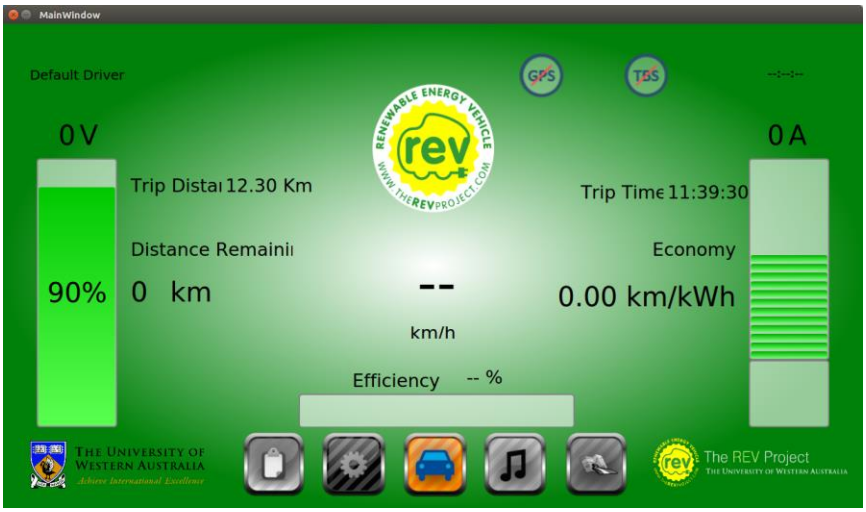


Figure 2.3: The GUI Design by Macus Pham. (2013)

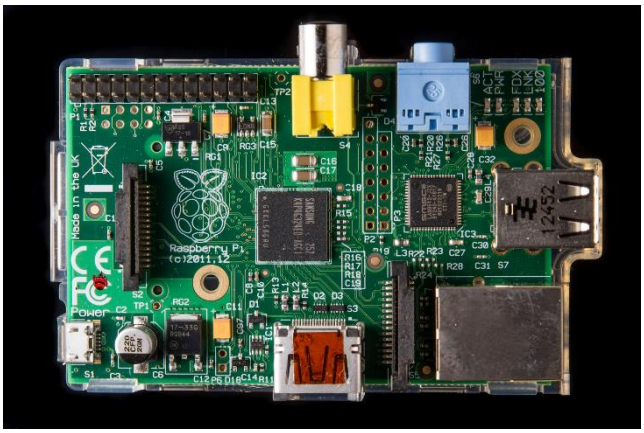


Figure 2.4: Raspberry Pi 1 Model B

2.2 REV Lotus

Since Varma firstly implemented a GUI system used for the Lotus, the REV Lotus system had always had a different design to that of the Getz until the year of 2013. Figure xx shows the different desgins for the two EVs in 2010. Despite the differences in GUI, these two systems essentially had similar functionalities and structures. Therefore, in the previous modification made to the REV Lotus system, it was designed with a same GUI of the REV Getz system. However, it is worth mentioning that the control methods for the two systems were entirely different. For the REV Lotus, the screen is a touch screen and hence user can navigate to any of the five tabs by a single click to corresponding button. As for the REV Getz, the navigation to different buttons was done through the Griffin Powermate (a rotating knob). This difference required different implementation measures and led to slight difference in program structures. Table 1 presents the key presses while testing the program in desktop.

Action	Keyboard Key	Powermate KeyPress
Left on navigation button	Q	Rotation left
Right on navigation button	W	Rotation right
Selection on navigation button	E	Press down
Navigation on buttons in windows	TAB	Rotation left/right
Selection of buttons in windows	SPACE	Press down
Exit a window	ESC	Hold down
Exit a menu	ESC	Hold down

Table 1: Key navigation for the GUI of REV Getz in destop and in vehicle

Figure 2.5: The VoomPC. (2013)

2.3 Tesla Model S

Telsa Model S is the second model developed by the Telsa Motors and was first on the market in 2012. In the Figure xx which shows the its interior, it can be found that the in-vehicle system is made up of two parts, a display behind the steering wheel and a big screen next to the driver's seat. The display is used to show speed, battery status and other crucial information (it can also display a navigation map if necessary) while driving. The toolbar above the map widget in the big screen (Figure xx) indicates that it provides more functionalities including media player, navigation map, battery status, web browser, camera management and phone call. Besides, there is a status bar at the top, showing a brief status of the car. Moreover, it can be noticed that the screen is divided into two separated windows, which allow the driver to view the map and control the media player simultaneously.

The left image in Figure xx is the version 5.0 of the main screen UI design from the Telsa Motors and the right one is the version 7.0. An obvious difference between these two UI designs is the style. From v5.0 to v7.0, Telsa used a flat style to replace the previous 3D button style. Despite the argument of whether flat styles are easier to understand, flat look is most effective on small displays and has recently taken over Windows, Mac, Android and other applications.

Figure 2.6: The interior design of Tesla Model S.

Figure 2.7: Main window UI v5.0 (left); Main window UI v7.0 (right).

2.4 Objectives Identification

After review of previous work on this project, it can be concluded that the main purpose of ultilizing the system in the REV EVs is to provide virtualized driving information. Additionally, other useful modules like data logger, GPS map and media player were added into the system. As this project is to develop an in-vehicle system, it should concentrate on assisting the driver while driving and not distracting he or she too much. A common risk in the driving process was pointed out by Goncalves[5] that the driver spends too much attention on the secondary tasks like manipulating a driver information system (DIS) or reading data from a display, and neglect the primary one, which is driving safely. This may lead to a decrease in driving performance and safety. Therefore, the consideration of providing a safe driving environment has the most priority in the modification work to the existing GUI.

Before the modification work to the previous system can be conducted, it is an important procedure to upgrade software, hardware and eliminate defects in the previous GUI program. Marcus has proven the necessary of this step when he took over this project from Gabriel Feng. As the hardware change of using RPi to replace both EyeBot M6 and VoomPC had already implemented by Gabriel, the upgrade work at that time basically included upgrading the operating system for the Rpi and using QT 5 instead of QT 4 for program compilation. Additional work like setting up the connection of CAN bus to Rpi in the REV Lotus and installing a Hall-Effect Sensor to enable tachometer were also carried out to keep the controller able to receive and process driving information from the vehicle.

To sum up, the primary objectives of this project are as listed below:

1. Upgrades in software and hardware. It can be expected that some problem would occur and impact the execution of the program due to the incompatibility between different software environments and hardware configurations.
2. Redesign of the existing GUI to enhance user experience under the premise of minimizing the distraction to the driver while driving.
3. Re-implement the music player. Since a built-in media player class is provided in the latest Qt library, the performance and stability can be improved through the re-implementation of the music module by using the new Qt class.
4. Re-implement the map. The previous map module used a customized class to read pre-downloaded map tile images and render them to the window. The latest Qt library provides several

plugins to implement same functionality, providing significantly better performance and fluency.

Once above changes have been completed, it can be expected to deliver an improved in-vehicle HMI system with better usability and functionality.

Implementation

3.1 Hardware Upgrade

Rpi 3 Model B is the latest product of the Rpi series. It was published as the third-generation Rpi in February 2016 (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/) and replaced the Rpi 2 Model B. In contrast to Rpi Model B, it provides wireless LAN, Bluetooth Low Energy (BLE), 2 more UB ports, and extremely more advanced processor. Besides these, it is worthy to mention that a micro SD port has took the place of SD port for loading operating system and storing data since Rpi 2. These advantages should be able to make it worthy to take the increased cost. Till the publish date of this thesis, the price of Rpi 3 Model B is $67.95 AUD (https://raspberry.piaustralia.com.au/) and that of Rpi Model B is $39.95 (https://www.adafruit.com).

The first problem faced during the process of hardware upgrade was incompatibility between the operating system installed on the previous controller, namely Rpi Model B, and the new Rpi. This is because of the backward compatibility of the Rpi series products, which means older Rpi is supported by later Raspbian operating systems while older Raspbian is not compatible with later Rpi products. This factor additionally verified the necessary of upgrading the operating system.

The installation process of new operating system was relatively simple. An arguable point may be about the choice of desktop environment (DE). The default DE of the official Raspbian distribution is PIXEL, a branch of LXDE, which is a light weighted DE and recommended to be used to decrease memory usage. In this case, the operating system stays with the default DE.

3.1 Software Upgrade

As aforementioned, Qt as a cross-platform application framework[8] was firstly introduced to the software development process in this project by Walter in 2010. After almost a decade of development, the latest version, Qt 5.9.2, has integrated numerous built-in classes and plugins, offering developers great convenience and efficiency. Two most significant changes in this project also benefitted from this, which will be introduced later. By upgrade the Qt framework, it means to compile the previous program with a new Qt IDE which is up to date. Qt itself provides both forward and backward compatibilities and thereby the most part of code in the existing GUI program worked fine while compiled. Problems were raised while compiling the external libraries.

During the evolution process of the REV HMI systems, a number of third party libraries were used in the program and finally several libraries were still left in the system, which are listed in Table xx.

Library	Utility
Qcustomplot	Plot tracks on the map.
Gizmod Daemon	Enable the use of the rotating knob in Raspbian and the other Linux OS.
Qextserialport	Enable the system to receive data through USB ports.
ALSA (Advanced Linux Sound Architecture)	Audio driver to enable voice control in Raspbian and the other Linux OS.

Table 3: External libraries used in the previous sytems

In general, these external libraries excluding ALSA still work appropriately. As a touch screen is going to be installed in the REV Getz as well, it will be no longer necessary to use the Gizmod Daemon library. Besides, the utilities of both Qcustomplot and Qextserialport libraries can be replaced by the built-in plugins or classes in Qt library. As for ALSA, the previously used library seems to crash with current Qt framework as a more up to date ALSA library has been integrated as a part of it. Therefore, the references to this library in the source code were removed and a built-in function of QMediaplayer class is used to control the system volume.

3.3 GUI Design

In the previous GUI system, there are 5 tabs for the REV Getz and 6 tabs for the REV Lotus (the one extra tab is the battery tab). In this section, some significant defects in the GUI design are pointed out and a new design scenario is introduced.

3.3.1 Home Tab

The home tab is designed to display some crucial driving information, including the voltage, current, trip data, warning informs, and especially the speed, etc. On the basis of previous GUI design, the first edition of modified Home Tab UI is as shown in Figure xx. The modification mainly consisted of a new background, new button icons and warning icons, and a much more striking speed number. This new background looks more comfortable and makes it easier to distinguish the UI elements. Moreover, a flat style design was adapted to the changes of button and warning icons. In compliance with the core idea to minimize the distraction to the driver, a much bigger font size was used to the speed text. This design allows the driver easily gets the current speed and less dedicated to look at the driving

information display while driving. However, a fatal error of this design was that it was designed with a screen size of 1920×1080, while the resolution of the touch screen is 1024×768. Therefore, the GUI design was redesigned to fit the small display. In the new GUI, the older background is no longer suitable as it may look too crowd in a much smaller screen. The button icons have the same problems. Besides additional changes to the background image and button icons, there is also a useful improvement that the warning icons were separated from the home tab, which makes them visible when the user navigates to other tabs. This modification was made due to the importance of warning information.

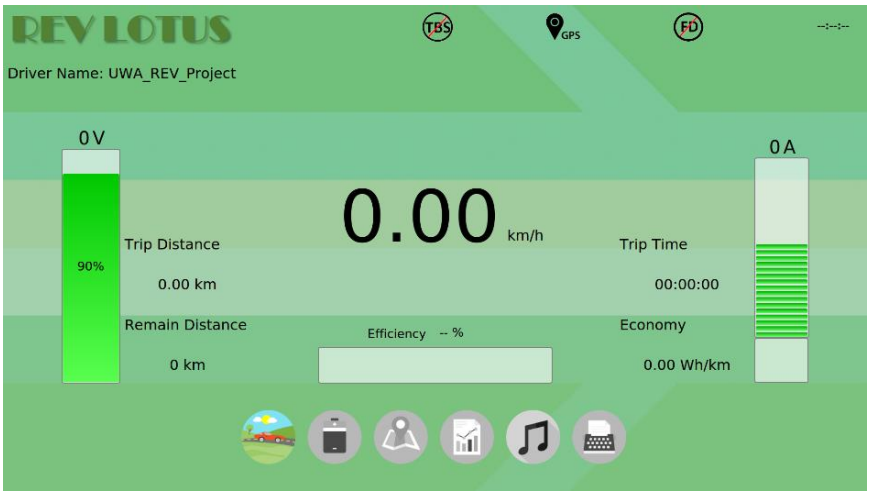


Figure 3.1: Original GUI design

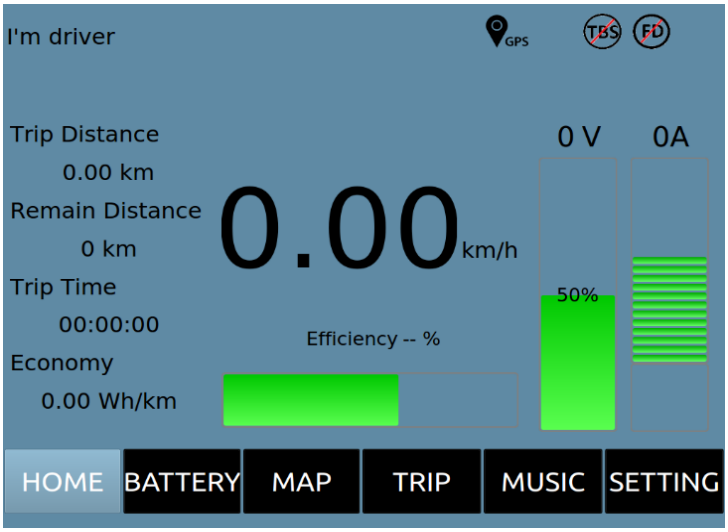


Figure 3.2: The current GUI design

3.3.2 Battery Tab

The battery tab plays a crucial role in the design of EV[10]. In this project, difference in battery tabs of two REV EV systems is due to the different design of their battery management system (BMS). In the REV Getz, a TBS monitor was used to access the battery information, while in the REV Lotus, the Hardware Black Box system developed by Daniel Kingdom is used in the BMS[7]. It has a USB interface with FTDI drivers for an emulated serial port and provides serialized information from the BMS - battery pack voltage, battery pack current and individual cell voltages and currents. It also transmits information about the state of the digital inputs and analogue inputs - temperature of the battery packs, headlights, door sensor and seatbelt sensor via serial.

This tab is only available in the REV Lotus system, used to display the battery status and relevant statistical data. The widget at the right upper area can display the voltage of all available batteries the informs driver the position of those batteries in uncommon conditions. It can be noticed that at the top of the window, the warning icons and driver name label are still displayed on the screen.

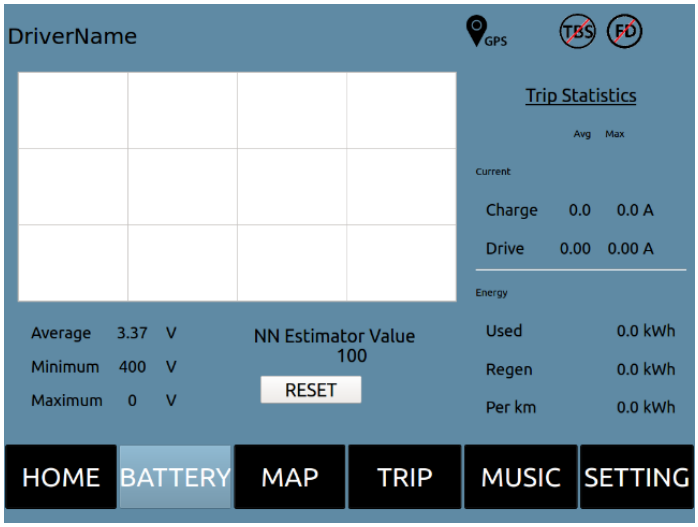


Figure 3.3: The battery tab

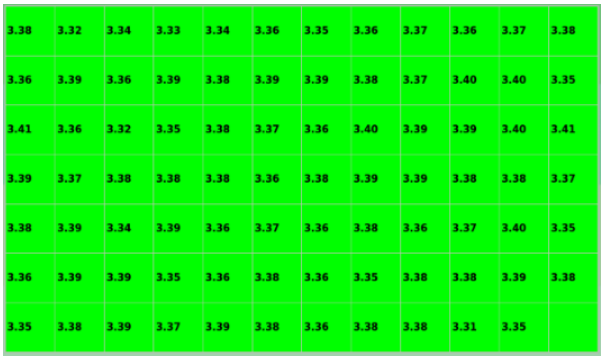


Figure 3.4: The battery table in work

3.3.3 Map Tab

From the aspect of GUI, the new design of the map tab maximized the size of the map on the premise that the menu buttons and warning icons are always available or visible. At the top right area of the map widget, the zoom level number is provided. The new map was re-implemented by the Qt Mapbox plugin and the zoom level is a metric to quantify the scaling extent and the bigger the zoom level number is, the clearer the objects on the map are. In this program, the range of zoom level is set from 12 to 20, in order to optimize the size of map cache. This map was implemented by the Qt Quick module with mapbox plugin for QML.

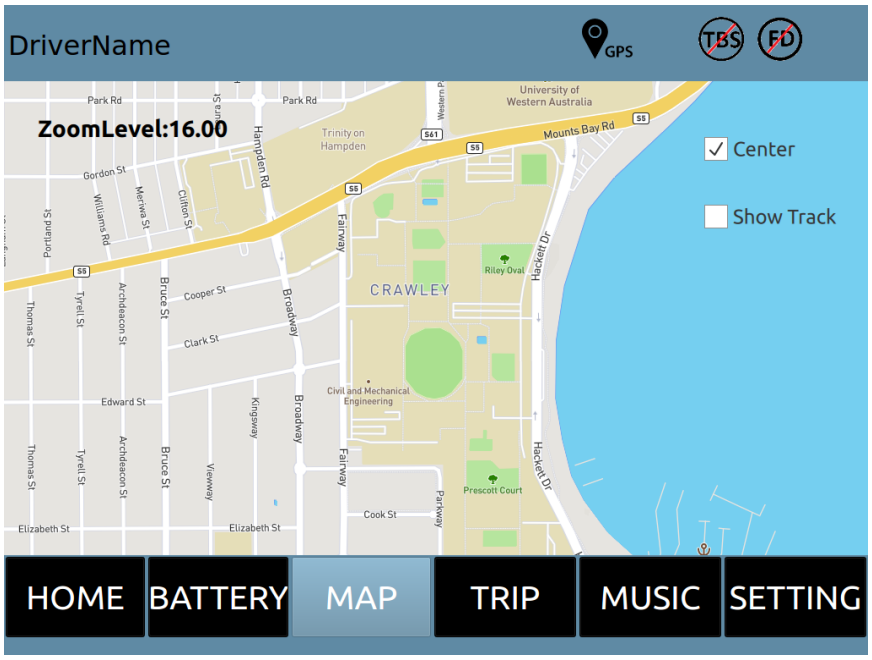


Figure 3.5: The map tab

QML is a declarative language commonly used to design UI-centric applications. Qt Quick is a SDK which Nokia created for writing QML applications. Providing a standard library for QML. A mapbox map can be created by writing below code lines in a *.qml file:

```
Plugin {
    id: mapPlugin
    name: "mapboxgl" // "osm", "esri", ...

     PluginParameter {
         name: "mapboxgl.access_token"
         value: "pk.eyJ**********************"
     }
```

a)

```
Map {
    id: map
    anchors.fill: parent
    plugin: mapPlugin
    center: QtPositioning.coordinate(-31.979283, 115.815852) // UWA
    minimumZoomLevel: 12
    maximumZoomLevel: 20
    zoomLevel: 16
    copyrightsVisible: false
}
```

b)

Figure 3.6: a) Choose a map plugin; b) Create a map.

Plugin is a QML type used to specify the name of desired plugin and define necessary parameters. Then a Map type can cite the id of the plugin and then use it to display the map images fetched from the specified map provider. Here, the defined plugin parameter, the access_token, is a unique token used by developers to access any of the Mapbox's tools, APIs and SDKs. Another parameter defined in this program is the cache directory. By specifying the value of this parameter, the fetched map tile images will be automatically stored in the specified directory for offline use. As aforementioned, the new map module can benefit from the advanced map rendering algorithm, which provides a much faster rendering speed than the algorithm used in the previous system, which was barely improved after it was firstly implemented by Varma in 2009. Nevertheless, the new mapbox map also allows user to zoom in or zoom out the map by tap of 2 fingers, and hence the "Zoom In" and "Zoom Out" buttons were no longer needed and removed.

3.3.4 Trip Tab

Similar to the battery tab, the trip tab was not changed too much as it is used to present driving information. This information including both motivation data and battery data are useful to help the driver find out his or her driving habit and monitor the correlation between current and speed. Besides adjusting the UI elements to fit the size of the touch screen, there is also a minor change that the driver name label used to change the name of driver was moved to the setting tab, which it is more conform to. The

warning icon tab at the top is still available in this tab, as well as in the other tabs.

3.3.5 Music Tab

This is another tab which was entirely rewritten in this project. The first one is the map tab. From the aspect of GUI, a customized style sheet was applied to the vertical slider bar and the horizontal slider bar to make the GUI design more conform to a flat style. As for the functionality, the new music player automatically loads all files of *.wma, *.mp3 or *.mp4 format in pre-specified directory. The user can either directly choose music in the table or use the two buttons at the right side to navigate in the table. The green button is used to play music when there is no music being played. Once any music is played, the icon is instantly switched to a pause sign and by pressing it the music player will be temporarily paused. Pressing the button again will play the music from the position it was paused. The red button is the stop button. Different with the pause button, after the stop button was pressed, playing music again will start at the beginning rather than from the historical position. The shuffle button is the play mode button. Similar to the green button, it is used to switch among three different play modes, which are shuffle, repeat and loop mode, respectively.

Figure 3.7: The music tab

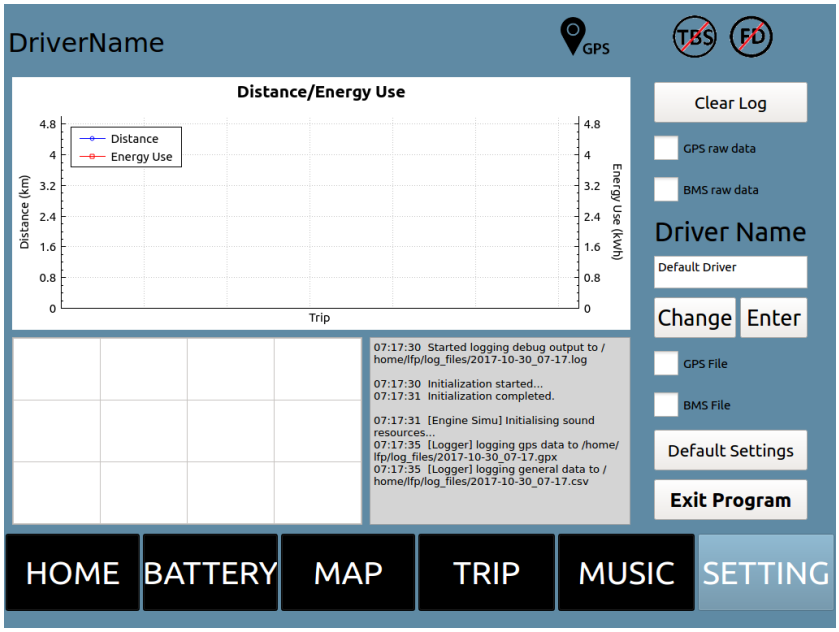
```
//initialise mediaplayer
playlist = new QMediaPlaylist(this);
playlist->setPlaybackMode(QMediaPlaylist::Loop);
player = new QMediaPlayer(this);
player->setPlaylist(playlist);
connect(player, SIGNAL(positionChanged(qint64)), this, SLOT(setProgSlider(qint64)));
```

Figure 3.8: Set up the music player

An object of QmediaPlayer is created to play music. This class provides many useful of APIs like play, pause, and stop music. Additionally, it can also emit signals to inform other module that its status has changed. For example, positionChanged(qint64) is a pre-defined signal of Qmediaplayer and emitted every 2 seconds by default if playing progress is changed. The setProgSlider(qint64) slot, or function, will be called every time it receives the signal and used to set the handle of slider bar to corresponding position.

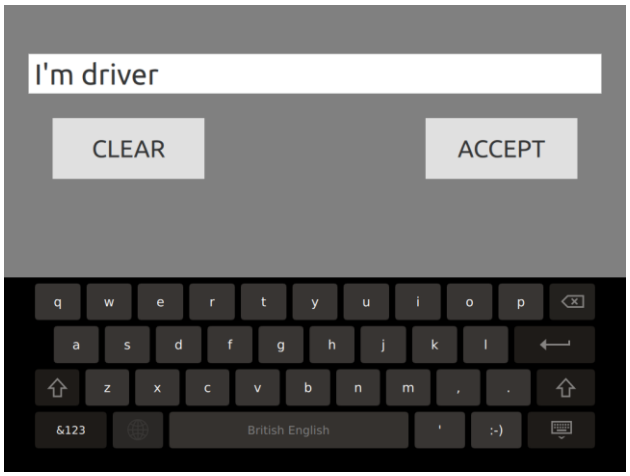
3.3.6 Setting Tab

It was mentioned that the driver name label was moved into this tab. Although a text input widget is provided for type in the user name, there is no valid input method while using the system in the vehicle, which means the driver is not able to set or change the driver name without plug in a keyboard. This is apparently impractical. Therefore, an input panel was provided in the renewed system and will be called once user clicked

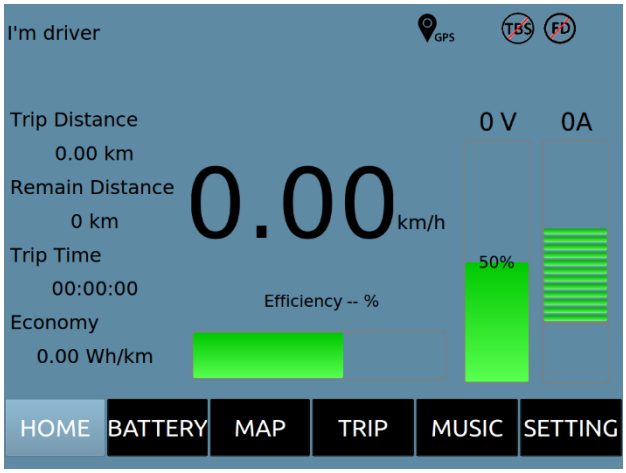


the text field. Considering the limited screen area, when the input panel is called, it occupies the entire window and is hidden when user clicks the accept button.

Figure 3.9: The setting tab



a)



b)

Figure 3.10: a) The input panel; b) The driver name was changed.

This input panel is implemented by the virtual keyboard plugin provided in Qt Quick module. There are three procedures to use the plugin in a QML

application. Firstly, the reference to the virtual keyboard plugin should be declared in the main.cpp, and then the corresponding header should be imported to a *.qml where virtual keyboard is used. Lastly, the InputPanel type should be configured. Figure xx shows a typical configuration of this type. By adding appropriate compile commands while compiling the virtual keyboard library, it also provides numerous input methods from different locale. For example, by configuring command

CONFIG += "lang-de_DE"

the input panel can be switched to the layout of German locale. This new feature can be not only used for type in the driver name, but also used to type input for a web browser or phone numbers for mobile calls if these modules are implemented in the future.

Methodology

4.1 Integration of C++ and QML

Qt is once a framework dedicated to software development using C++ language. As a result, although QML is highly differentiated from C++ in both structure and syntax, there are still many ways to easily integrate C++ code into QML applications or extend C++ applications by QML files. In this project, this technique was utilized in two approaches. The first one is to render a QQuickView in a QWidget-based application, and the second one is to emit a signal in C++ file and trigger a function in QML.

The first approach is relatively simpler. It was used to display the map window and the input page using virtual keyboard, both of which are implemented in QML files. As the previous GUI system was a purely QWidget based application, the QML files cannot be directly rendered to the main window. Alternatively, the GUI created in the QML files are put into a QWidget first, using

createWidgetContainer(QQuickView mapView);*

function. Once this widget is used as a container, any configurations excluding geometry information of this widget will be overridden by the QQuickView, which is the mapView in this case. The geometry information decides the size and the absolute position of the widget in the main window.

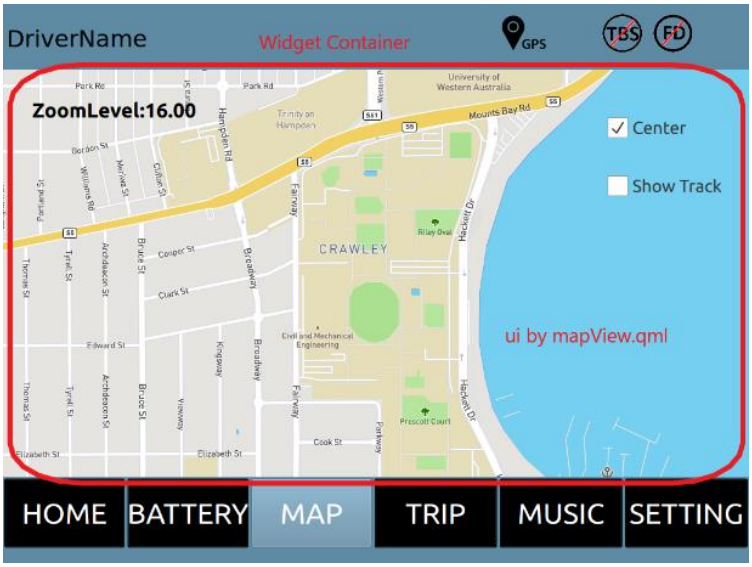


Figure 4.1: example of using widget to render a QQuickView

Another application of integrating C++ with QML is to connect a signal in C++ file and a slot in QML file. In the current program, position data including latitudes and longitudes were read by C++ code, using the *qextserialport* library. In order to send the position data to the QML file and draw the track on the map, a feasible solution is to use a signal created in C++ to trigger a slot in QML and transfer the data as a parameter. Firstly, use *rootObject()* function to get the access to attributes in *mapView.qml*, then use *findChild<>()* function to find the type in QML with specified object name. Now, the objects in both C++ and QML are aware of each other. After connect the pre-defined signal and slot, C++ can inform the QML end to update the track by emitting the signal.

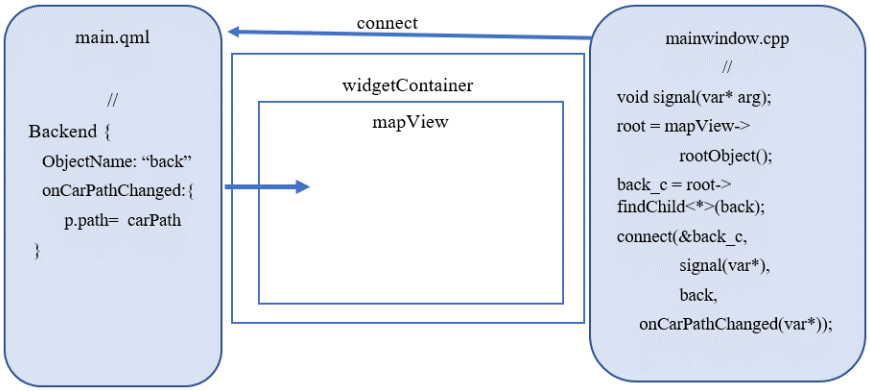


Figure 4.2: connect QML and C++ with signal and slot

4.2 Douglas Peucker Algorithm

In the existing HMI system, the GPS module faithfully records the position of the EV after every fixed time interval. The detailed track records can help drivers remind their driving behavior and realize their driving habits. But it is also a common situation that we want a smoothen track which shows the general route other than describing even minor changes in direction. Besides, the lost in the accuracy of GPS receiver may also produce noises in the track[9]. In this project, the Douglas Peucker algorithm (DP) is used on demand to simplify the route drawn on the map.

DP, also known as the Ramer-Douglas-Peucker algorithem, is commonly used to simplify a curve consists of line segments into a similar curve with fewer points[3]. The simplification process should start with a curve with a set of sequential points or lines. Assume C is a set of ordered points ***Pi***

$$C=\{Pi\ (xi\ ,\ yi\ )|i=1,\ 2,...,\ n\}$$

Where n is the number of points in the set and ***xi*** *,* ***yi*** are the coordinates of the point with index ***i***. Then the solution of DP can be described like this[4]:

- Select the point with the greatest perpendicular distance to the line segment between the first and the last point in the curve;
- If the distance between the point and the primary line is greater than a specified tolerance, then divide the primary line into two line-segments based on the position of the point chosen in the first step;
- Repeat above two steps until the convergence goal is achieved.

This is a simple but efficient method widely used in simplification work for map service providers. A potential situation which may benefit of applying this algorithm is when the track is very long and composes of a large number of nodes. In this case, it can be expected that DP will help eliminate those inessential points and improve the system performance.

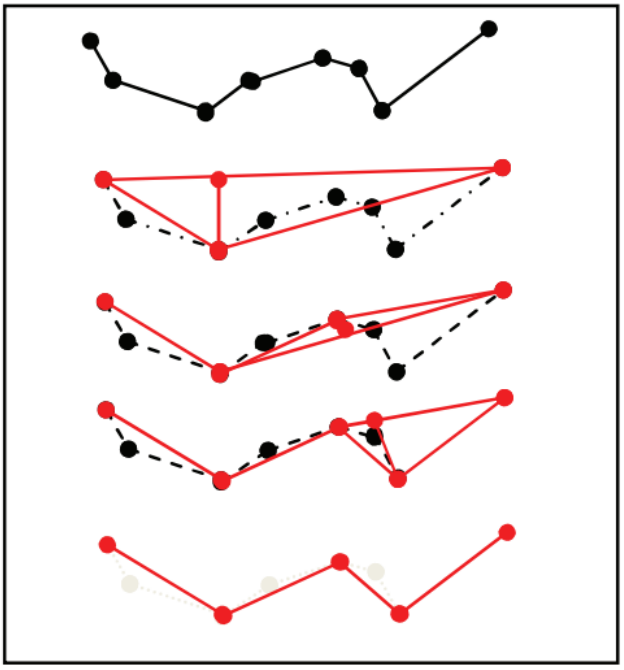


Figure 4.3: Simplify a piecewise linear curve with DP

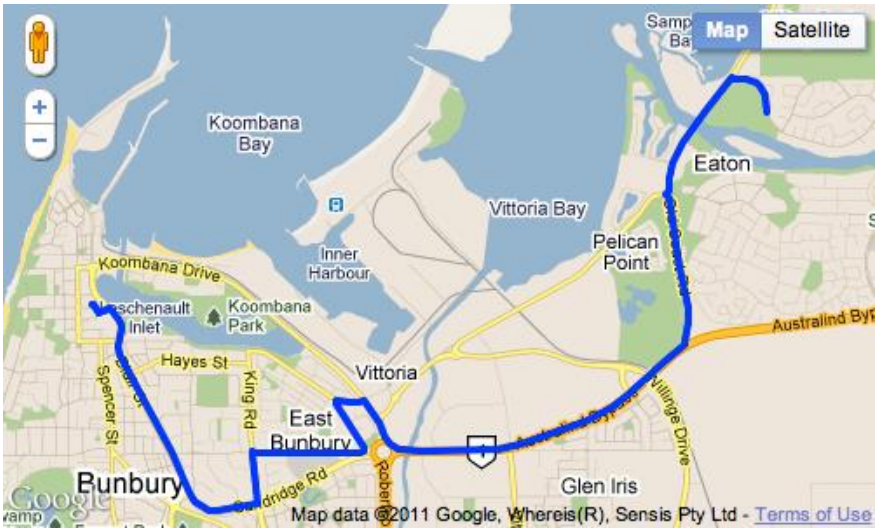


Figure 4.4: Track smoothened by DP (Source: http://blog.transittimesapp.com/category/gtfs/)

Problem & Future work

In this project, a considerable amount of effort was spent to investigate Qt APIs as Qt is used as the development framework. After countless read of Qt Documentation, the official handbook provided by the Qt company, it was found out that a much wider range of ready-to-use APIs and plugins are available in QML while those APIs in C++ libraries are usually used to access or process detailed data. Taking Qt Location as an example, many C++ APIs are provided to offer developers the access to almost all necessary geocoding and navigation information, and most of QML APIs are used to render these information on a map. Therefore, an optimized strategy for software development in the Qt framework might be to design the UI by QML and use C++ to implement the backend like reading and processing data. Such development scheme can make the program benefit not only the abundant interactive functionality in QML application like touch gestures, layer overlays, and so on, but also the precise and modular structure design and higher performance in C++ programming.

In order to make use of the various plugins provided for QML application, the map and virtual keyboard module were implemented in QML while other GUI modules were kept with C++, considering the numerous connections between the backend and the GUI. However, the aforementioned method of integrating C++ application with QML files were turned out to be infeasible on RPi. As a cross-platform tool, Qt provides a platform plugin for displaying the visualized elements in Qt application and this is the only available choice on RPi devices. There is a strict restriction in this plugin that it only renders either a QWidget window (C++) or a QQuick window (QML) at the same time. Any attempt to display two different type of windows or a hybrid window makes the plugin terminate the Qt program. The hybrid window includes using a QWidget as the container to render the UI in a QML file. This makes it impossible to separately implement a part of the project in QML and then integrated it into the original program.

Therefore, in order to make use of the modules implemented in QML to provide better performance and additional functionally, an important work following this report would be re-implement the entire GUI part in QML and thereby there is only one main window in the system to be render, which is the QQuick Window. As all of these new features require the interactivity with screen, upgrading the REV Getz HMI system with touchscreen is also considered as a part of future work.

Conclusion

It was found that the previous systems can still work appropriately in the REV EVs. Therefore, this project mainly concentrated on improve the existing systems to deliver an enhanced in-vehicle HMI system which provides better user experience and more useful functionality. The new map module and input method on the basis of QML APIs can enrich the interaction with touchscreens, which is a common method in the modern technologies to enhance the usability of HMI systems[12]. Another achievement in the project would be the improvement of the GUI. Although there was not a specified specification used in the GUI design, it is admittedly an improvement that the new system offers the driver a better user experience to some extent. The navigation map with flexible touchscreen control and the clear driving information display provides driver not only convenience but also safety, as they produce less distraction to the driver.

The program has been well best tested on desktop, but the crucial new features are still not able to be delivered to the REV EVs due to the restriction of display Qt application on embedded device. Therefore, the re-written of the GUI system is still undergoing and will be hopefully bring the latest features provided by Qt to the REV EV systems.

1. En.wikipedia.org. (2017). Electric car. [online] Available at: https://en.wikipedia.org/wiki/Electric_car [Accessed 28 Oct. 2017].

2. Global EV Outlook 2017. (2017). [S.l.]: OECD, p.12.

3. En.wikipedia.org. (2017). Electric car. [online] Available at: https://en.wikipedia.org/wiki/Electric_car [Accessed 30 Oct. 2017].

4. Li, L. and Jiang, W. (2010). An improved Douglas-Peucker algorithm for fast curve approximation. 2010 3rd International Congress on Image and Signal Processing.

5. Goncalves, J., Rossetti, R. and Olaverri-Monreal, C. (2012). IC-DEEP: A serious games based application to assess the ergonomics of in-vehicle information systems. 2012 15th International IEEE Conference on Intelligent Transportation Systems.

6. Pham, M. (2017). Renewable Energy Vehicles' User Interface. Perth: University of Western Australia, p.4.

7. Varma, D. (2009). Renewable Energy Vehicle Instrumentation: Graphical User Interface and Black Box. Perth: University of Western Australia, p.71.

8. Kang, P., Wei, Y. and Wei, Z. (2017). Control system for granary ventilation based on embedded networking and Qt technology. 2017 29th Chinese Control And Decision Conference (CCDC).

9. Shahid, M., Shahzad, M., Bukhari, S. and Abasi, M. (2016). Autonomous vehicle using GPS and magnetometer with HMI on LabVIEW. 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS).

10. Sixto, V., lopez, P., Sanchez, F., Jones, S., Kural, E., Parrilla, A. and LeRhun, F. (2014). Advanced co-simulation HMI environment for fully Electric Vehicles. 2014 IEEE International Electric Vehicle Conference (IEVC).

11. Kang, P., Wei, Y. and Wei, Z. (2017). Control system for granary ventilation based on embedded networking and Qt

technology. 2017 29th Chinese Control And Decision Conference (CCDC).

12. Rozhdestvenskiy, D. and Bouchner, P. (2017). Human machine interface for future cars. Changes needed. 2017 Smart City Symposium Prague (SCSP).