Diagnostic tool for trucks
-from idea to demonstrator

Qi Wang & Yinrong Ma

Bachelor thesis
15 credits

Halmstad University, June 2013
Diagnostic tool for trucks
-from idea to demonstrator

Bachelor’s Thesis in Computer Science and Engineering
2013 June

Author: Qi Wang & Yinrong Ma
Supervisor: Stefan Byttner
Examiner: Kenneth Nilsson
Description of cover page picture: Diagnostic tool for trucks
Preface

First of all, we would like to express our sincere appreciation to our supervisor Stefan Byttner for his support and feedback during the thesis work.

Furthermore, we are grateful to Volvo Group Trucks Technology and Viktoria Institute for giving us the chance to do this project, especially Magnus who gave us valuable suggestions and technical support.

Finally, we want to thank Halmstad University for offering us the opportunity to do this project. It was a good chance for us to learn more.

Qi Wang & Yinrong Ma
Abstract

Vehicles can end up in unplanned visits to workshops due to the driver not checking the vehicle status before using it in traffic. There are many factors not only caused by the environment but also due to the lack of tools that simplify or reminds about beforehand inspections. The purpose of this project was to introduce a smart-phone application that can display the health state (or related parameters) of a vehicle in a brief way and indicate if a part or function of the truck is not working properly. There are six functions in the application. Function status and function fault codes can display information about vehicles by giving two-dimensional plots about vehicle data, while function VSR displays some information in the form of text. Also, the user can submit their feedback through function comment. Function position is designed to give the users specific perspectives on an imported map based on their different user identity. Function check reminds about inspections that must be made before setting out on a driving mission. The application allows bus drivers and managers to continuously monitor different vehicle parameters with a statistical summary over time, as well as providing a method for following-up that drivers perform basic checks on the vehicle before it is taken into traffic.

Key words: smart-phone application, diagnostic vehicle tool, vehicle maintenance, visual data, service
Contents

1 Introduction....................................................................................................................... 1
  1.1 Project description ........................................................................................................ 1
  1.2 Project goal ................................................................................................................... 1
  1.3 Project scope ................................................................................................................. 2
  1.4 Outline ......................................................................................................................... 2

2 Related work ................................................................................................................... 3
  2.1 Navigation ..................................................................................................................... 3
  2.2 Diagnostic and maintenance ...................................................................................... 4

3 Methods ........................................................................................................................... 5
  3.1 Development environment ......................................................................................... 5
    3.1.1 Mobile platform ...................................................................................................... 5
    3.1.2 Functions and libraries ......................................................................................... 6
  3.2 Visualization of data .................................................................................................. 7
    3.2.1 UI components ..................................................................................................... 7
    3.2.2 Data representation ............................................................................................. 11
    3.2.3 Google maps ........................................................................................................ 13

4 Results ............................................................................................................................ 15
  4.1 Design of diagnostic tool ......................................................................................... 15
  4.2 Implementation of application ................................................................................ 16
    4.2.1 Login and main menu .......................................................................................... 16
    4.2.2 Function plotting ................................................................................................. 18
    4.2.3 Function position ................................................................................................. 25
    4.2.4 Function VSR ...................................................................................................... 26
    4.2.5 Function fault codes ......................................................................................... 28
    4.2.6 Function comments ....................................................................................... 30
    4.2.7 Function check ................................................................................................. 32
  4.3 Discussion .................................................................................................................. 35

5 Conclusion ....................................................................................................................... 37
  5.1 Summary ..................................................................................................................... 37
  5.2 Future work ................................................................................................................. 39
Figures

Figure 3-1 World-Wide Smartphone Sales ................................................................. 5
Figure 3-2 Summary of three chosen popular platforms ........................................ 5
Figure 3-3 Mobile - Server Model ......................................................................... 6
Figure 3-4 Relative layout ...................................................................................... 7
Figure 3-5 Drop down list ..................................................................................... 8
Figure 3-6 Tab .......................................................................................................... 8
Figure 3-7 Button .................................................................................................... 9
Figure 3-8 Text View ............................................................................................. 9
Figure 3-9 Checkboxes ......................................................................................... 10
Figure 3-10 Dialog windows .................................................................................. 10
Figure 3-11 Toasts ................................................................................................. 11
Figure 3-12 Radio Buttons .................................................................................... 11
Figure 3-13 Line chart .......................................................................................... 12
Figure 3-14 Bar chart ......................................................................................... 12
Figure 3-15 Pie chart ............................................................................................. 13
Figure 4-1 Flow of the application ........................................................................ 15
Figure 4-2 The login interface and the main menu ............................................... 16
Figure 4-3 The drawable folder ............................................................................. 17
Figure 4-4 Warning for incorrect password (in the left) and both UserID and Password (in the right) ... 18
Figure 4-5 Line chart and pie chart ...................................................................... 19
Figure 4-6 The string resource file ...................................................................... 19
Figure 4-7 Daily perspective ................................................................................ 20
Figure 4-8 Weekly perspective ............................................................................. 20
Figure 4-9 Initial settings of tabs .......................................................................... 21
Figure 4-10 Code about the back button .............................................................. 22
Figure 4-11 Histogram data of gear usage ............................................................ 22
Figure 4-12 Histogram of gear usage ................................................................. 23
1 Introduction

This chapter includes four sections, the original intention of the project is explained along with the specific goal and scope. The last section outlines the structure of the thesis.

1.1 Project description

As a common means of transportation, trucks and buses are widely used in the modern society. At the same time, there are a number of reports of failed inspections [1, 2] where it has been concluded that many vehicles are not fit for traffic. It has been found that it happens that drivers do not or cannot perform a sufficient inspection of the vehicle before taking it into traffic. It is a problem for the drivers, passengers and the environment if the vehicles are not kept in good condition. Furthermore, this can also cause disruption in supplies, add cost to the operator company and other stakeholders. Hence vehicles do need a check by drivers before driving as well as monitored by managers for service planning. However, it is hard to equip all types of vehicles with the same instruments and the dispersed arrangements of them are also potential hinder aspects.

When it comes to understanding large amounts of technical data quickly, rich background knowledge and experience are always indispensable. How to present sufficiently detailed information so that a driver or a manager can understand it is a challenging topic. Visualizing several different types of data is desirable, but can be hard to implement with the limited display capabilities of a driver dashboard.

Meanwhile, the maturing technologies of mobile devices like smartphones and tablet computers provide a promising opportunity. It is no problem to display data or plot diagrams on a mobile device now. These devices are usually battery-driven, portable, environment friendly, money-saving (compared to the cost of relevant truck instruments). From this point of view, it would be useful to have an ICT-based smart-phone application that can provide vehicle health related information (and other related information) that can be used by vehicle drivers and their managers.

1.2 Project goal

Our aim is to develop a smart-phone application (with a suitable interface) for displaying vehicle health related information. The intended users are not only vehicle drivers but also their managers. The application would be adapted to different users. To achieve the whole mission, the overall tasks are divided into two steps:

Primary Objectives

- Implement an application with a suitable interface by means of using the open source platform, as a mobile phone application.
- Select data from available onboard data, service records and fault codes, in
Introduction

discussions together with Volvo for the app to use. The data will then be read from a file for real-time display in the app.

- Presented all of the data in the visualization form.

Secondary Objectives

- In order to document that the drivers have seen and approved of the current status of the vehicle the signature of drivers need to be recorded.
- Driver notes of non-measured data – Some details need to be checked manually (existence of a fire-extinguisher and spare tire, the battery in fire alarm, outer mirror, etc.). It would be useful if drivers can make notes related to these issues in the app.
- Server streaming data – Since there will not be a physical vehicle that can stream data to the app, the communication protocol can be extended from reading the data from a file to streaming the data from a server.

1.3 Project scope

The project was limited in the following respects:

- There is limited availability of data from a real vehicle. The project does not have access to a real vehicle, only a few weeks of already collected onboard data and the vehicles service records.
- We have no information of user experience since it was not possible to perform this investigation within the timeframe of the project.
- Compared to other apps in the market, there is a less smooth design, such as click bars then gives feedback, operating by users gesture and choosing perspectives is quite inconvenient because users always need click three times to get a plot.

1.4 Outline

The remainder of this report is organized as follows. Chapter 2 includes information relevant to this work. Chapter 3 presents the main tools and methods used to develop this smart phone application and to get data. Chapter 4 gives the specific solution to each objective as well as discusses the results and make comparisons. Chapter 5 summarizes the work done in this project and describes future work.
Related work

2 Related work

After using “vehicle maintenance” and “navigation app” as key words to search in “the Google search engine”[3], we found an article named “Best Android apps for drivers, car owners, and car enthusiasts”[4] which lists the applications for cars that have both maintenance and navigation features. When using “bus maintenance and navigation app” or “truck maintenance and navigation app” as key words, we found that the applications provide either maintenance or navigation but not both. The conclusion was that diagnostic and maintenance features are common for smartphone apps (at least for the car market), but lacking some features that would be desirable for commercial vehicles (i.e. trucks and buses). This chapter describes some of the related work already on the market, as well as a comparison to the features of the application we have developed.

2.1 Navigation

When it comes to the handheld mobile devices that can be helpful to the vehicles which will be taken into traffic, navigation is one of the most common functions since guidance about direction, current position and route to the destination are all important for driving. Applications installed on smart phones could have advantages like more functions and prettier interfaces than special equipment such as Car Navigator. So a lot of smart phone applications provide navigation for the drivers contemporarily.

iPhone applications for trucks like Discover NavTruck [5], for example, is the first voice-guided smartphone truck GPS navigation. This one has special design about taking into account truck-restricted and prohibited roads to provide drivers’ safety and reliable navigation on truck-legal roads. It guarantees the drivers’ safety by providing the navigation of the most suitable routes which is widely used among the navigation applications and the design of its interface could be a reference.

For Android, Truck GPS Route Navigation [6] which also provides voice-guide as well as selecting suitable route by taking into account commercial truck restrictions is another truck navigation application example. Users can choose different map perspectives in this Android application.

However, the similar solutions are all different from ours. All of the related works are mainly solving the safe navigation problems of drivers by giving guidance and automatically avoid the problems caused by restrictions such as prohibited roads for vehicles during driving. In other words, they have not mention the maintenance of vehicles when they navigate at the same time. While our project is mainly aimed at combining both the vehicles’ maintenance and simple navigation in one, for a specific target audience, commercial drivers (namely bus drivers or truck drivers).
Related work

2.2 Diagnostic and maintenance

Being aware of the health condition of vehicles before driving is important so there are also many such applications on the market.

The first example is MineFleet Software [7] which is designed for commercial fleet owners and fleet management companies who want power of advanced predictive vehicle data mining for dramatically reducing the operating costs. This software offers onboard data stream mining for modeling, benchmarking, and monitoring of vehicle health, emissions, driver behavior, fuel-consumption, and fleet characteristics. Although this software is also available on many different handsets and smartphones, its functionality is limited. When users have done some inspections for their vehicles, they cannot take notes. Analyzing as well as offering reliable data is just a part of our project goal. We want to make a simple interaction between the application and the user so our application will display data and take feedback from the users.

The second one is called Rev iPhone App [8] which can display real-time vehicle data and its user interface is closed to the dashboards, but there is no display of e.g. vehicle service records. That is to say, users just get the gathered data but also cannot take records.

Another one is made for Android, there are applications such as AutoHealth [9], which claims to be able to tell you the health of a car before you buy it and displays details by text.

These applications are all third party applications, which mean they don’t have access to OEM (original equipment manufacturer) proprietary data. As mentioned in section 1.2, our target application would include displaying vehicle data such as onboard data, service records and fault codes which are provided by Volvo. After the users inspected their vehicles, they could make records as well as take notes about it and submit with their signature. Our goal is focused on an interaction between users and the application since it includes both display data and taking notes.
3 Methods

The main methods used in the project are described in this chapter. It includes development environment, and the tools we used for our project.

3.1 Development environment

3.1.1 Mobile platform

The Android mobile platform was chosen because of the large market share (see Figure 3-1) and the characteristics of the development environment (see Figure 3-2). [10, 11] In the meanwhile, JAVA is chosen as programing language and Eclipse as compiler environment. [12, 13]
Methods

After installing Java, we use the ADT Bundle provided by Google which includes the essential Android SDK components and a version of the Eclipse IDE with built-in ADT to streamline the Android app development.

3.1.2 Functions and libraries

- Data Access

Since there will not be a physical vehicle that can stream data to the app, we will be implementing streaming of data from a server. Thus the server-side scripting language --- PHP [14] is chosen, HTTP [15] is used as communication protocol and simultaneously JSON [16] used as data format.

As this is an operation to the server, we bundle the flags which need be sent to the server by using method BasicNameValuePair("flag", flag) to build post parameters in the java file. Then in php, we make a check that bundle actually exists by using the function isset($_POST['flag'])?$_POST['flag']:'' which tells the server to use the flag or set it null character.

It is essential for this operation to connect to internet and make HTTP requests. Before calling these methods, access to internet needs to be established. Then come to the steps of making requests. Firstly, a HTTP client is created by calling the method DefaultHttpClient() to create a new HttpClient object. Secondly,HttpPost() is called to create a new HTTP post object. Thirdly encoding post data by using httpPost.setEntity(new UrlEncodedFormEntity(nameValuePair)). Finally httpClient.execute(httpPost) is used to execute the post.
Methods

JSON is the syntax of the storage and exchange text messages. It is similar to XML and easier for decoding than XML. We simply call `json_encode()` in php file and use JSON Object to get the string then decode in java file.

- Imported Library

To fulfill the requirements of plotting in the application, importing an available library becomes the preferred solution. After our investigation, the most commonly used several plotting libraries for Android is called AChartEngine, androidplot and Google Chart Tools.[17, 18, 19] Google Chart Tools are powerful and simple to use and they can provide the most beautiful interfacial among the three. However, the network is indispensable for using this library. androidplot is also a good choice, but it does not support pie charts. Therefore, among these options we have chosen AChartEngine as our charting library. This library supports not only chart types which can contain and display multiple series, but also provide many other custom features. What is worth mentioning is that the charts which can be built either as a view which can be added to a view group or as an intent, such as being used to start an activity.

3.2 Visualization of data

3.2.1 UI components

Android provides large amounts of wonderful interface components. The ones we used mainly for our application are listed as follows.

- Layout

We use relative layout for our overall layout so that the position of its child views can be specified as relative to sibling elements or to the parent area.

![Relative Layout](image)

*Figure 3-4 Relative layout*

- Drop down list

The drop down list (called spinner in Android) is commonly used for providing a quick way to select one value from a set.
**Methods**

**Figure 3-5** Drop down list

- Tab

Taking advantage of the fixed tabs’ character --- display all items concurrently, they are used to navigate to different kinds of plots.

**Figure 3-6** Tab
Methods

- **Button**

Buttons are used for triggering the related actions which communicates by the text on it. ImageButton is used in our menu because it can display a button with an image instead of text by defining the image on the surface of the button either by the android:src attribute in the XML element or by using the setImageResource(int) method.

![Figure 3-7 Button](image)

- **Text View**

The input operation is essential and the component we chose is the textfields.

![Figure 3-8 Text View](image)

- **Checkboxes**

Clicking is easier than writing everything down, so the check box is the solution we used for things which need check before driving.
Methods

Among various kinds of dialog windows, the lightweight version, popups is chose to display related information to the user and they can either make a single selection advances the workflow or touching outside simply to dismissed it.

According to the principles of UI design, users do need a clear feedback about an operation. The toast is space saving since it only fills the amount of space required for the message and will disappear automatically.
Methods

Radio buttons allow the users to select one option from mutually exclusive optional sets.

![Radio Buttons](image)

**Figure 3-12 Radio Buttons**

### 3.2.2 Data representation

Display data by using charts is involved in the notion of data visualization wherein the visual manifestation of such data is defined as a certain profile to form extracted out of the information, including various attributes and variables of the corresponding information unit. [20] Data acquisition, data analysis, data governance, data management, data mining and data transforms are all the relative fields. Among the related fields, the field what our project involved is data analysis. This field includes exploratory data analysis (EDA) and qualitative data analysis (QDA) or qualitative research. Histogram is one of the typical graphical techniques that are used for EDA. And we also use other kinds of charts in the application. Charts often include an overlaid mathematical function depicting the best-fit of the scatter data. According to section 1.2, we use two-dimensional charts to display the onboard data.

- **Line chart**

According to the definition of line chart – it is a basic type of chart which is often used to visualize a trend in data over time series.
A histogram represents the distribution of data graphically. The height of a rectangle equals to the frequency density of the interval and the total area of the histogram equals the number of data.

A pie chart is a circular chart divided into sectors, illustrating numerical proportion. The arc length of each sector (and consequently its central angle and area), is proportional to the quantity it represents.
3.2.3 Google maps

Nowadays, Google provides a variety of versions of Google Map API for different platform applications which allows developers to offer interactive, feature-rich maps to users. In our application, Google Maps Android API v2 is used to fulfill the requirements of showing position. Before working with the API, we update the Google Play service, download, register for the API and obtain the key as a prerequisite. The following configuring steps in our application are demonstrated as below:

Add the key to application by inserting

```xml
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="your_api_key"/>
```

as a child of the `<application>` element in `AndroidManifest.xml`.

Add `<permission>`

```xml
<permission
    android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>

<uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
```

Finally do some specify settings in manifest. Set permissions of `INTERNET` (already done when using HTTP as communication protocol), `WRITE_EXTERNAL_STORAGE`, `READ_GSERVICES`, `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION` and add `<uses-
**Methods**

feature android:glEsVersion="0x00020000" android:required="true"/> because version 2 requires OpenGL ES version 2.

The `<fragment/>` element in XML file is used for the filling of the map segments. And in java file, create a SupportMapFragment object, then call `getSupportFragmentManager().findFragmentById()` to specify the fragments for filling and use `getMap()` to obtain. Objects of class `LatLng` is the key element which gives latitude and longitude.
4 Results

This chapter mainly presents the requirements, implementation and results of the project. The last part gives a discussion about the results.

4.1 Design of diagnostic tool

Figure 4-1 displays the main flow of the application and the related sections are also labeled in it. A login interface will be displayed first once the users start the application. After inputting the UserName and the PassWord, our application will judge and then give relevant feedback. If the inputted data matches an entry in the user database, the user can login successfully, otherwise, they will return to the beginning. After correct inputting, the application will present six functional modules to the user, namely: Status Plotting, Position, VSR (Vehicle Service Record), Fault Codes, Comments and Check List. The detailed description of the function and implementation of the login interface as well as each module will be stated in the following section.

![Figure 4-1 Flow of the application](image-url)
4.2 Implementation of application

4.2.1 Login and main menu

Users login the app with their own account. After checking if the username and password is correct or not, a warning will be displayed if there is a mismatch with the user database. If all things are correct, they will go into the welcome page. Before starting to use the tool, the user must first choose the truck type and vehicle number. When they return from functions, the right picture will be shown. If the user wants to change the truck type and the vehicle number, they can press the "change" button. If they want to log out, they can press the "log out" button. The used app widgets and the layout arrangement of this function interface are presented in Figure 4-2 as well as the results of switching between different pages. The interface in the left is the login page. And the following two are the two different types of main menu, one is when the user login at the first time and the other is the page which the menu would switch to when the user want to change the information of the vehicle. We customize what the widgets look like in the .xml files and manage what actions they need to accomplish in the related .java file.

![Figure 4-2 The login interface and the main menu](image)

- Specific Implementation

Button

The change button is set at the right-top corner. The `<Button/>` element is customized in the related .xml file. This button is created for providing the choice interface for the truck type and vehicle number. In order to declare the event handler programmatically, create an `View.OnClickListener` object and assign it to the button by calling `setOnClickListener(View.OnClickListener)`. These two functions enable the button icon clickable as well as programmable so that the users can see some actions after the clicked the button. In this case, the button’s response is to switch
Results

pages. To complete switching, an intent object is created in the onClickListener. It makes the current interface which looks like the screen shot on the right of Figure 4-2 change to the one on the right. The intent should be declared in the AndroidManifest.xml by giving the <activity android:name="ACTION ">/activity> element.

ImageButton

This one is very similar to Button, it can display a button with an image instead of text by defining the image on the surface of the button. All of the buttons on the menu are ImageButtons which are created for jumping into different function pages. This kind of button needs some specific settings. That is put the icon in the drawable folder (see Figure 4-3), then define android:src="@drawable/the icon name" to connect the button with this icon. The rest of the settings are the same as buttons which is described above.

![Figure 4-3 The drawable folder](image)

TextView and EditText

The TextView shows the truck type and vehicle number. When users finished choosing in the drop down list, the values will be stored in variable MainActivity.truck and MainActivity.vehicle. After jumping into other page we can set the TextView from those two parameters. For example, truck_no.setText("Truck: 
	" + MainActivity.truck). Then the related information will be displayed. The widget EditText is used to get the inputs from the users. Do edits to the <EditText/> element in the related .xml file and at the same time define the EditText object user_text = (EditText)findViewById(R.id.userid); and password_text = (EditText)findViewById(R.id.userpwd); Then the inputted information can be stored in these two variables and will be checked whether they are correct or not.

Tests
Results

In this section, the main test for the login interface is to make sure the users can use the application only when they provide right id and password. If either one of these two is wrong, the application will give a warning (see Figure 4-4). When it comes to testing the main menu, the result is that the pages switched successfully after the user pressed the change button (see Figure 4-2).

![Image](image.png)

**Figure 4-4** Warning for incorrect password (in the left) and both UserID and Password (in the right)

4.2.2 Function plotting

Three kinds of vehicle data are displayed as two dimensional plots in two different time perspective. These data are fuel rate, gear usage and boost pressure, which reference to the data calculated by calling related Matlab functions. As mentioned in section 3.2.2, bar charts, line charts and pie charts are decided to be displayed. We thought the progress of generating correct data for bar charts is representative so this progress will be illustrated in the following paragraphs and generating data for the rest two kinds of charts are skipped since they are similar to the progress of bar chart. We also simulated the three kinds of plot in Matlab so that the charts in the application can reference to these plots after they have been adjusted. In this function, Users choose the time and data type in the three drop down lists orderly, and then the three types of plot will be showed. The used app widgets and the layout arrangement of this function interface are presented in Figure 4-5. The image also shows the line chart and the pie chart through daily perspective. The arrangements of the widgets are edited in the .xml files and the actions they need to accomplish is assigned in the related .java file.
Specific Implementation

Drop down list

For the user, providing data from different time perspectives is more ideal. In view of this, the drop down list is designed as secondary linkage drop-down menu so that the user can choose the perspective first then the specific time.

Content of the drop down list is pre-determined and provided with a string array defined in a string resource file: res/values/plottingselectionarray.xml file. Figure 4-6 is what the resource file looks like.
Results

In the plotting java file, method findViewById point out the related XML element so that the code we wrote in .java file would be matched with the appointed widget. Using createFromResource() method to create an ArrayAdapter for the string array. The following are the steps used to fill in the adapter with the resource so that the choices have been connected with the adapter. The simple_spinner_item layout is provided by the platform. And the default layout is selected for the third argument for this method. Then call setDropDownViewResource(int) to specify the layout. The adapter should use to display the list of spinner choices simple_spinner_dropdown_item is another standard layout defined by the platform and selected for the argument for setDropDownViewResource(). Call setAdapter() to apply the adapter to the Spinner. Then what the adapter contains would become the choices of the drop down list. When the user selects an item from the drop-down, the Spinner object receives an on-item-selected event. Then what the adapter contains would become the choices of the drop down list. Implement the AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method to define the selection event handler for a spinner. Then specify the interface implementation by calling setOnItemSelectedListener(). To achieve the requirement of secondary linkage drop-down menu, the spinner that provides the perspective is designed as parent. Give the corresponding menu options according to the first level drop-down menu option position by initializing the specific time spinner in the parent’s setOnItemSelectedListener() method. The implementing results are displayed in Figure 4-7 and Figure 4-8.

![Figure 4-7 Daily perspective](image)

![Figure 4-8 Weekly perspective](image)

In the sub-spinner’s method setOnItemSelectedListener(), customized function plotting is called so the specific chart will display according to the clicked position of the list.

Tab

There are three types of plot offered by the plotting function according to 3.2.2. The three – click rule [21] of web site design principles could also be a reference of
Results

Android application, so the fixed tabs are adopted as a solution of this consideration. Each tab corresponds to a kind of chart and the charts are given concurrently when the user clicks the drop down list. Steps of setting the tabs are quite simple to understand: find the related widget and connect it with an tab adapter, give tab an specific id, decide what would be shown under this tab, give the tab title and told the adapter all was ready. Specific initial settings of tabs are showed in Figure 4-9 below. Find the TabHost first, then give the identifier to a tab, set the corresponding content layout, sets the indicator and add these settings to the tab at the end.

```
TabHost th = (TabHost) findViewById(R.id.tabhost);
th.setup();
TabSpec spec1 = th.newTabSpec("tag1");
spec1.setContent(R.id.linetab);
spec1.setIndicator("LineGraph");
th.addTab(spec1);
```

Figure 4-9 Initial settings of tabs

Plotting

This is the specific part of how to give the tab related content. At first, the function clearviews() is called. It is designed to find the tab content and remove the filled views in them, so the content of tabs can be refreshed. By using the library achartengine, the steps of plotting the three types of chart are quite similar. Create a graph object first by using BarGraph bg = new BarGraph(barstr); The library provides two ways of getting the charts. One is getChartIntent, the other is getChartView. Since the graphs need to be displayed in the layout of corresponding tab content, the getChartView is ideal. So the return types of all the Graph classes are set ChartFactory.getChartView(context, dataset, mRenderer, Type.DEFAULT); Then GraphicalView bggView = bg.getView(getApplicationContext()); gives the plot a GraphicalView object for filling. Finally use the findViewById method to assign the specific tab content. The details are pasted as follows: LinearLayout layout2 = (LinearLayout) findViewById(R.id.bartab); layout2.addView(bggView);

Button

A back button is created at the bottom of the interface by editing the <Button/> element. This button is created for controlling switching interfaces since it has the name back. We also need to implement the OnClickListener to active this icon. In order to declare the event handler programmatically, create an View.OnClickListener object and assign it to the button by calling setOnClickListener(View.OnClickListener). To complete switching, an intent object is created in the onClickListener (see Figure 4-10).
Results

Figure 4-10 Code about the back button

```java
final Button back = (Button) findViewById(R.id.back);
back.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        Intent intent = new Intent();
        intent.setClass(StatesPlotting.this, Welcome.class)
        startActivity(intent);
        StatesPlotting.this.finish();
    }
});
```

The intent should be declared in the AndroidManifest.xml by giving the `<activity android:name="ACTION ">
</activity>` element.

Signal histograms

We make the histogram be normalized displaying relative frequencies of different types of data and the proportion of cases that fall into each of several categories is showed, with the total area equaling 1. Detailed designs and implementation of histograms are presented below and the implementation of function plotting is expounded above.

In the selected data, the one of gear usage is quite different from others. The gear data can be represented using integers, while the other data is floating point value. (see Figure 4-11).

In Matlab, Function `hist()` is called to generate a histogram from the data. Then we get the histogram’s two needed axis values. In Figure 4-11, GX1 represents the abscissa axis values and GN1 represents the ordinate axis values. The height of the histogram equals to the frequency density of the interval.

```
>> [GN1,GX1] = hist ([5300,0:100:9000],7);
>> GX1

-1    1    3    5    7    9    11

>> GN1

GN1 =

506    0   318   126   184   490   1803
```

Figure 4-11 Histogram data of gear usage

In consideration of making the height of histograms represent the proportion of cases that fall into each of several categories, the ordinate axis values are treated by a simple mathematical calculation that is using the height of the bars divided by the number that represents the member in these categories.
Results

In the meanwhile, comparing two groups of data by plotting them in the same graph is more common and intuitive. So there are two types of data – current data and normal data. The current one is handled by using one group and the normal one is handled by averaging the data of the rest groups. The result is shown in Figure 4-12.

![Figure 4-12 Histogram of gear usage](image)

To extract current data from server, we need one side to post the data so HttpClient httpclient = new DefaultHttpClient(); is used to create a client in the app first. Secondly, build HttpPost httppost = new HttpPost("https://ideweb2.hh.se/~yinma12/first.php"); to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI("https://ideweb2.hh.se/~yinma12/first.php" in this case). Thirdly, use List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(4); to create a list to store HTTP variables and their values then nameValuePairs.add(new BasicNameValuePair("plot_graph_flag",plot_graph_flag)); and httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs)); to add the HTTP variables and value pairs. Finally, send the variable and value, in other words post, to the URL using HttpResponse response = httpclient.execute(httppost); str = EntityUtils.toString(response.getEntity()); used for obtaining data.

From the server side, the .php file receive the bundled flags by $plot_graph_flag = isset($_POST['plot_graph_flag'])?$_POST['plot_graph_flag']:'"; this code guarantee the .php file to get the posted flags, otherwise, set the variable in .php file as none.
then select data in a nesting switch case. The data is just stored in the form of arrays.

$x = array(-1,1,3,5,7,9,11);$ $y = array(0.1477,0.0.0928,0.0368,0.0537,0.1430,0.5261);$ 

At the end, give the value to variable $r_arr and output it through by means of $r_arr = array('DataX'=>x,'DataY'=>y); Function json_encode help the program to make the data in JSON form so that we could get and decode the data in JSON form which is a kind of lightweight data-interchange format that can make the application runs efficiently in the progress of exchanging data. So print json_encode($r_arr); The normal data is assumed stored in the app beforehand.

Since the source sent from server side is encoded as JSON format, JsonProcessor.java is created to deal with decoding at the client side. In this public class, the achieved source string is assigned to a list then split into two sub lists: listX and listY (see Figure 4-13).

```java
16 public static List ProcessJson(String str) throws JSONException {
17     JSONObject jsonObj = new JSONObject(str);
18     JSONArray dataX = jsonObj.getJSONArray("DataX");
19     JSONArray dataY = jsonObj.getJSONArray("DataY");
20     List listData = new ArrayList();
21     List<Double> listX = new ArrayList<Double>();
22     List<Double> listY = new ArrayList<Double>();
23     for(int i = 0; i<dataX.length(); i++){
24         listX.add(dataX.getDouble(i));
25         listY.add(dataY.getDouble(i));
26     }
27     listData.addAll(listX);
28     listData.addAll(listY);
29     Log.d("ListData", listData.toString());
30     return listData;
31 }
```

Figure 4-13 Main code of Json decoding in java file

In the BarGraph.java, the bar graphs are customized by several methods of XYMultipleSeriesRenderer. This class is used for handling the axis values of the charts. XYMultipleSeriesDataset dataset = new XYMultipleSeriesDataset(); and dataset.addSeries(series); are the main steps for adding plotting data. By using this object and the specific function, we can customize what the charts look like because we set the axis values which are the core factors of a two dimensional graph.

- Tests

In this section, the main test is performed to check the drop down list works. In the case of an available wireless network, the corresponding charts can be displayed in the corresponding tab after the user choosing the specific time (see Figure 4-5). For further details please refer to the Appendix A.
4.2.3 Function position

According to section 1.2, the target users are both managers and drivers. For the drivers, a navigation function can be useful while for the managers, we designed to show their in charged vehicles’ current positions. Since these requirements are both about the position of the vehicles, we designed an additional function for our application and named it position. This one is required to have two different versions. Figure 4-14 shows what will be shown depending on if the user login as a driver or as a manager the user login as a driver and a manager. The used app widgets and the layout arrangement of this function interface are also presented in this figure.

![Figure 4-14 Driver perspective (on the left) and manager perspective (on the right)](image)

As mentioned in 3.2.3, we use Google Maps Android API v2 for implementation. A flag which identifies the user was set when he login. Then in the public class Positions which extends FragmentActivity we use the if/else if-clause to check the flag and gives the corresponding action.

- Specific Implementation

In the case of driver perspective, we set the method setMyLocationEnable of GoogleMap object true to enable my-location layer. Then the layer continuously draws an indication of a user’s current location and bearing, and displays UI controls that allow a user to interact with their location. There are four types of GoogleMapset, Normal, Hybrid, Satellite, Terrain, as well as a None type for the option to have no map at all. In this project, it is set as normal by MapType(googlemap.MAP_TYPE_NORMAL). In addition, we show some related information in order to give hints to the user by using the method makeText of Toast.
Results

The other case is the managers’ perspective. Another method `addMarker` is used after giving the latitude and longitude of a point in the form of LatLng (double latitude, double longitude) since we need emphasize points in the map. Since the information would be displayed after the users clicked the marks, we also give an OnItemClickListener to achieve this. Customize the markers by setting their properties in `setOnMarkerClickListener(new OnMarkerClickListener())` so that markers can give proper action when they are clicked.

A back button is set at the bottom. It is similar to the one in section 4.2.2.

• Tests

The functionality is tested by performing a check of the map and that the settings work. Current position shows when the driver clicking the icon in the top right corner. The manager can see the information of the vehicle condition displayed in the Toast at the bottom of the interface and the vehicle numbers as well as their route number. Figure 4-14 shows the results.

4.2.4 Function VSR

Vehicle Service Record mainly displays the records by text. Figure 4-15 shows the result of clicking the last tab. The used app widgets and the layout arrangement of this function interface are also presented in this figure.

![Figure 4-15 Interface after pressing Tab MORE](image)

• Specific Implementation

Drop down list
Results

The main steps we have done is similar to what in section 4.2.2, create an adapter for the items, connect the adapter to the drop down list and finally enable its choices to become clickable and can response to the clicking, but at this time the source of the drop down list is given in an array directly:

```java
final String[] more_sp_t = {
    "Electric systems and instruments",
    "Power transmission",
    "Brakes",
    "Wheel suspension and steering",
    "Frame, springs, damping and wheels",
    "Body, cab and interior",
    "Miscellaneous"};
```

Then customize the spinner itself. Call `setDropDownViewResource(int)` to specify the layout which the adapter should use to display the list of spinner choices and another standard layout defined by the platform `simple_spinner_dropdown_item` used as parameter for `setDropDownViewResource()`. Then call `setAdapter()` to apply the adapter to the Spinner. Implement the `AdapterView.OnItemSelectedListener` and the corresponding `onItemSelected()` callback method to define the selection event handler for a spinner. Then specify the interface implementation by calling `setOnItemSelectedListener()`. Related actions are given according to the position of list items.

TextView

The same as mentioned in section 4.2.1. After setting in the .xml file, method `findViewById` points out the display element and use `setText()` to give content for showing.

- **Structure Design**

Figure 4-16 shows what the data file of VSR would look like.

![Figure 4-16 The data file of VSR](image)
There are ten aspects included and under each section comes several related sub-titles. Therefore, design a suitable data structure can be helpful, thus the data structure of VSR is worth mentioning (see Figure 4-17).

In this structure, if the operation code is LO-561, the first number is "5", which means it's a problem related with the brakes. The second number is "6" which means it is related to "compressed air brakes". The third number is "1" which means it's a compressor problem. This information is used to structure how the information is displayed. For example, when the user goes to the VSR function, the user could see ten different boxes, i.e. the ten top level function groups (the first ten numbers). If the user clicks on one of the boxes, he then sees the alternatives that are for the second number, etc. This could be a way for the user to see, for example, has there been any replacement of the brake discs lately? When the oil filter was last changed? etc. Tabs are used to play the role of boxes mentioned above so that the user can be given the options at once when he goes to the function. In view of the size of tabs, it is unsuitable to display ten tabs at the same time on the interface, so we only make three of them displayed directly and the rest was supplied by the drop down list under the tab "MORE".

4.2.5 Function fault codes

Fault codes are another indispensable parameter for diagnose of vehicle's current condition. In order to display the codes, this function also shows a histogram.
Results

Further information can be given after the user click the related buttons. Figure 4-18 shows the result of choosing one type of fault codes and clicking the related button. The used app widgets and the layout arrangement of this function interface are also presented in this figure.

![Function fault codes interfaces](image)

**Figure 4-18 Function fault codes interfaces**

- Specific Implementation

RadioButton

After create the xml element of RadioButton, actions need to be set in the related .java file, and since this kind of widget is an subclass under the class android.widget.Button we need to do some similar settings like we have done to the buttons. So we implement the method RadioGroup. OnCheckedChangeListener() by giving an if/else clause that check which information the user wants to know. Then in the clause gives the corresponding definition. Use the getId() method of the radio button to establish contact between clicked button and selected fault codes type. After building the connection, generate the related bar graphs(see Figure 4-19).
Results

30

Figure 4-19 Implementation of the method RadioGroup

AlertDialog

In the Button.OnClickListener() we give the content that need to be displayed, use the
AlertDialog.Builder APIs is used to create the dialog where we give the information
(see Figure 4-20).

public void onClick(View v) {
    // TODO Auto-generated method stub
    new AlertDialog.Builder(FaultCodes.this)
        .setTitle("C0040-Right Front Wheel Speed Circuit Malfunction")
        .setMessage("3/3/2013")
        .show();
}

Figure 4-20 Implementation of the AlertDialog

The structure of fault codes is quite similar to that of VSR and it is defined by OBD-II,
see e.g. [22]

• Tests

In this section, it is important to make sure when the user clicks the button, the
corresponding dialog box will pop up and display the related information. The result
is shown in Figure 4-18.

4.2.6 Function comments

Storing comments from the drivers is valuable because drivers can observe details
(noises, smoke, leakage etc.) which the onboard sensors may not be able to detect.
In this function, users can type their comments as well as their names. Then the
application would record the input and add current time automatically. Figure 4-22
shows the interface that provided to the users to comment. The used app widgets
Results

and the layout arrangement of this function interface are also presented in this figure.

Figure 4-21 Comment interface

• Specific Implementation

EditText

This component is mentioned in section 4.2.1 and the steps are the same as that one. Add an EditText to the UI by adding an XML element to the XML layout (see Figure 4-23).

Figure 4-22 The <EditText/> element

Find the element by method findViewById(), then use method getText() to get the content of this field and transfer it to string so that the comments could be pass to the TextView to show.

Calendar

Current date is necessary to show along with the comments and it is the highlighted part in this function. The implementation is quite simple. We use the fuction
Results

Calendar to read supply date. `Calendar.YEAR`, `Calendar.MONTH` and `Calendar.DAY_OF_MONTH` are used as parameters of this function.

- Tests

Users can write comment in the editable field. After clicking the submit button, a hint would be displayed to give feedback and the written information would be shown together with the current time. The result is shown in Figure 4-22.

4.2.7 Function check

For safety reasons, drivers should check the vehicles before they drive but some of the parts cannot be checked by equipment, such as lights, battery, tires and so on. If all the things are in the normal status, it’s fine to go into traffic. If not, they need to report the abnormal part in the comment function. In this function, users cannot press recheck button before they check. On the other hand, they cannot press check any more after they finish check. If they want to see the checklist again, they should press the recheck button. Figure 4-24 shows the interface that provided to the users to check. The one in the left is the interface displayed after the user press the function button. The middle one is the checking menu and the one in the right shows what the checking list look like. The used app widgets and the layout arrangement of this function interface are also presented in this figure.

![Figure 4-23 Function check interfaces](image)

- Flow chart
The processing flow of this function is described in Figure 4-25. The application will first judge whether the user has checked or not and set related button available. Then display the checking list to the user and wait for the next action which indicates that everything has been finished. According to the operation of the radio boxes, the application will judge if it is necessary to switch to the comment function. If comment is unnecessary, the whole process comes to the
Results

end. Otherwise, the comment interface will be provided for the user so that they can make comments.

Figure 4-24 The flow chart of function check

- Specific Implementation

Radio Button

The difference from checkbox is that users can make multiple choices in checkbox, but in radio button users choose only one choice. It is similar to the Radio button in section 4.2.5. If there is some problem on the vehicle, users should report it in the comment. So if users choose the one in the last two, it will jump into comment page. There is a flag to check if users make comments to report problem from checklist or normal comments, that when users press the back button, the app will show the correct page depending on the flag.

AlertDialog
Results

This AlertDialog has some difference to the one in section 4.2.5. It is shown as a list and the list could be different which depends on the one user choose. The methods `setTitle()`, `setItems()` and `show()` under `AlertDialog.Builder()` was used to give the information's title, content and enable it to be shown as shown in the java file. Items on the list are given as follows: new `String[]` {"Warning messages", "Lights and warningblinkers", "Brake lights are functioning properly"} and the switch case is used to set different items on the list. (See Figure 4-26)

```
checklist.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        // TODO Auto-generated method stub
        switch(arg2) {
            case 0: 
                new AlertDialog.Builder(DriverChecklist.this).setTitle("In driver seat").setItems(
                    new String[] { "Warning messages", "Lights and warningblinkers", "Brake lights are functioning properly" }, null).show();break;
            case 1: 
                new AlertDialog.Builder(DriverChecklist.this).setTitle("Go outside the bus").setItems(
                    new String[] { "Lamps", "Tires", "Hatchets closed", "Listen for air leakage", "Look for fluid leakage", "Wipers are not stuck to the windshield", "Damoses", "Engine, visual inspection" }, null).show();break;
            case 2: 
                new AlertDialog.Builder(DriverChecklist.this).setTitle("Go inside the bus").setItems(
                    new String[] { "Chairs, seat belts, emergency hammer", "Fire extinguisher and first aid kit" }, null).show();break;
            case 3: 
                new AlertDialog.Builder(DriverChecklist.this).setTitle("Go to the driverseat").setItems(
                    new String[] { "Check the bus maintains air pressure while brake pedal is pressed", "After driving the bus out of the parking slot, check for leakage on the ground" }, null).show();break;
            default:break;
        }
    }

    Figure 4-25 Specific setting of AlertDialog
```

- Tests

Users can get the check list after shown the check button if it is the first time for checking or get the list by pressing recheck button if it is not the first time. As figure 4-27 shows, there are three radio buttons at the bottom of the checking list. If the last two of them have been clicked (the green sentences in figure 4-27), the user should write comments about his inspect (we give a reminding sentence in red about this action) because these two means there are some faults happened. Once the user has chosen either one of the last two radio buttons, the application can change the pages automatically. That is to say, decisions about comments would also be made depending on which radio button is pressed. Pages will switch automatically between function check and function comment. What happens after changing into the comment page is the same as click comment button directly. A hint would be displayed to give feedback and the written information would be showed together with the current time. The result is shown in Figure 4-24.

34
4.3 Discussion

Compared with the initial idea, we made a little adjustment in design during implementation because of some reasons. The comparisons and conclusions are listed as follows:

- Data is assumed reading from files at first. Thus HTTP is initially decided to be introduced for extending from reading the data from a file to streaming the data from a server at first, but it is used as a communication protocol and the data is stored in server in this project.

- The original intention of the function Position is designed for the drivers to report his current position and gives the feedback to manager whether it is possible arrive the goal given by the manager under the present circumstances. However, it is more like navigation to the driver and a graphical and dynamic report on map now.

Compatibility design is an essential topic for mobile devices. In particular, horizontal screen adaptation processing mode is the most common. Managing screen sizes, resize for landscape mode will involve the concept of threads which is excluded in this project.
5 Conclusion

In this chapter, the achievements in the project are summarized and potential future work is outlined.

5.1 Summary

This thesis presents the development of a diagnostic smart phone application that can be used by both vehicle drivers and managers. The application is designed for the Android operating system and has a user-friendly interface and is designed for bus driver's needs.

According to the mentioned aspects in section 1.1, The original intention of the project was to develop a convenient and practical program to improve vehicle efficiency. AB Volvo provided us with suggestions on what to show and also gave some useful data such as onboard data, fault codes and VSR (vehicle service record).

Onboard data can reflect the status of vehicles in real-time. To know whether the vehicle is in good condition or not, comparison between real-time status and normal status cannot be avoided. Onboard data of this application can be divided into two meanings. One is real-time data and the other is recorded data. While fault codes represent the parts that have problems, for example, lamp broken, emission fault codes, low battery voltage, low cooling water level and so on. Users can know the actual error part through fault codes. And VSR, the one that is the abbreviation of vehicle service record, shows the history of vehicle maintenances. It could also be helpful for the users to understand the vehicle status.

After determining the basic functions, we discussed about what kind of information should be the feedback that is written by the users. Because interaction is necessary but all the data discussed above is just vehicle information. So we have designed comment and checking list as two additional functions to the application. Function comment provides a way for the users to be able to add their own notes. For example, during the driving, he went to change the tire and got refueling and the headlights and mirrors need recheck before next transportation. However, there are many things that need be checked manually and it is not always the case that drivers check that the bus is in a proper state before using it. In order to deal with that problem, we designed a check list in which we require checking different parts of the bus before driving.

In addition, the application’s target users are both the drivers and the managers, so we thought about reporting the vehicles’ current positions to them and plan to differentiate the displayed interfaces according to the users’ identity.

After that, the project goals are settled and presented in section 1.2. Finally there are six functions respectively called plotting (called status in the main menu), VSR, fault codes, comments, check and position in the application.
Conclusion

Regarding one of our project’s original intentions (see section 1.1), visualizing different types of data is required so the onboard data and fault codes are both presented in the form of a two-dimensional charts display in the application. In other words, function plotting and function fault codes are displaying charts. The users can choose to view the data according to different time perspectives, so that analyses and comparisons are more convenient and intuitive for the user. While the vehicle service record is presented in the form of text, so in the function VSR the users select the appropriate components to view the records on demand. Regarding the routine check before driving and the users need record related information the application provided a checklist for reference and record in function check. In addition, users can also submit a detailed comment with the function named comment. It is worth mentioning is that in function position the information displayed on the map is specified for different user identity.

The user class (driver or manager) is determined when they are using their own accounts to login so that some information of vehicles such as VSR (Vehicle Service Record), fault codes, checking lists and status data are showed as basic functions of the application. Drives can make comments on vehicle health, while managers can only browse the comments. In addition, the driver can be shown his current position on map and the manager can see the distribution as well as the health condition of all the vehicles.

Charts are used to display several different kinds of data. Some selected onboard data (fuel rate, gear usage and boost pressure) was read from a server and compared to the normal one which stored in the application by using two-dimensional charts. Different perspectives (daily perspective or weekly perspective) are available for the users. While the fault codes are displayed by a bar chart and related details could be shown after selecting the related button.

Some details need to be checked manually (existence of a fire-extinguisher and spare tire, the battery in fire alarm, outer mirror, etc.) are listed in a checking list in the application. Thus these non-measured data will not be missed. When it comes to essential comment, the application provides a brief interface that can document and submit what the drivers have seen and inspected of the current status of the vehicle as well as their signatures so that a kind of feedback could be recorded.

The vehicle service record is also an important aspect that both the drivers and the managers can view. Therefore, the application supplies ten groups of records and they are displayed in the form of text in responding to the user’s specific selections.

Compared with the project objectives, the function called position that displays the role of position navigation is a newly designed part in our application. Drivers can be shown their current positions while managers can be shown related information of a group of vehicles through this function.

In short, this application provides an easy and direct way of accessing some important vehicle information. Health monitoring becomes possible and convenient.
Conclusion

by displaying different kinds of onboard data and recorded repair history in the form of plots as well as texts. Streaming analyzed data from a server lightens the burden of smart phone and advances the before-hand data processing. It also optimizes the checking before the deliveries by giving detailed tips. Feedback after regular inspections can be stored and reported. Support for reporting the real-time status of the vehicles during their deliveries helps transportation monitoring.

5.2 Future work

The project introduced several limitations which leaves several possibilities for future work in the application. Features that could be implemented are e.g.

- Improving the Google map display to accommodate for information related to the goal of the driver (and the possibility for a manager to make changes to this in real-time).
- Integrate the application with the vehicle service record database.
- Implementing the function of warning if the driver has been driving too long without a stop for rest.
References


Appendix A – Different kinds of charts in the application

The charts of boost pressure in daily perspective

The charts of boost pressure in weekly perspective
The charts of fuel rate in daily perspective

The charts of fuel rate in weekly perspective
The charts of gear usage in daily perspective

The charts of gear usage in weekly perspective
Qi Wang
The author is a bachelor in Computer Science and Engineering.
E-mail: sinmo0523@gmail.com

Yinrong Ma
The author is a bachelor in Computer Science and Engineering.
E-mail: noelma1990@gmail.com