A Mobile Application for Monitoring Inefficient and Unsafe Driving Behaviour

Adnan K. Shaout and Adam E. Bodenmiller
The Electrical and Computer Engineering Department
The University of Michigan-Dearborn
Dearborn, Michigan 48128
shaout@umich.edu; aebodenm@umd.umich.edu

Abstract—Many automobile drivers are aware of the driving behaviours and habits that can lead to inefficient and unsafe driving. However, it is often the case that these same drivers unknowingly exhibit these inefficient and unsafe driving behaviours in their everyday driving activity. This paper proposes a practical and economical way to capture, measure, and alert drivers of inefficient and unsafe driving. The proposed solution consists of a mobile application, running on a modern smartphone device, paired with a compatible OBD-II (On-board diagnostics II) reader.

Index Terms— Real Time Systems, Road Vehicles, Mobile Applications, Mobile Devices, Smartphone, Inefficient and Unsafe Driving, OBD-II reader

1. INTRODUCTION

There are several measurements, which can be used alone or in combination, that can help indicate if a driver is driving unsafe or inefficiently. A small subset of these measurements includes:

- Acceleration & Deceleration
- Vehicle Speed
- Detection of Faults in Safety or Mechanical Equipment
- Environment Conditions (e.g. Traffic & Weather)
- Rate of Fuel Consumption
- Engine RPM

Although these measurements can help determine the safety and efficiency of driving activity, there has not been a practical way to capture and display these data to the everyday driver in a way that can impact the driver’s behaviour in real time or in a manner in which it could be examined and reflected on in a historical sense.

1.1. Capturing Safety and Efficiency Measurements

In the past capturing these measurements has been impractical for the average driver. These types of measurements had to be captured in a controlled environment or through the use of expensive and specialized equipment.

Modern mobile devices, like smartphones, now possess many of the general features required by this specialized equipment. These features include a user interface to accept input and display output, computing power to run real time calculations, the ability to run specialized applications, and the ability to store and retrieve data both locally and remotely (e.g. Internet and GPS). Bringing these features to a mobile device has decoupled many of the features that made these specialized devices expensive or impractical for the everyday driver.

It is estimated that 23% of mobile consumers now have a smartphone [1]. Since many of these mobile platform producers (i.e. Apple, Google, etc.) have opened up their devices to accept third-party hardware and software. The power of these mobile devices can now be paired with less expensive specialized equipment that does not have to recreate or include the features that are available on the mobile device.

When the features of a modern mobile device are paired with an inexpensive OBD-II reader, the mobile device becomes a powerful tool that can directly communicate with the vehicle’s Engine Control Unit (ECU). Doing so allows the smartphone to capture, interpret, and display real time data that details the current state of the vehicle, as well as measures the driver’s interactions with the vehicle.

An OBD-II reader is a device used for troubleshooting problems with a vehicle or retrieve real time performance data by connecting directly to a vehicle’s ECU. In 1996 a law was passed that made it mandatory for all vehicles sold in the United States to support the OBD-II specification.

1.2. Recent Use of this Technology

Recently, in the mobile application market, several applications have emerged that pair the power of a mobile device with the information available through the use of an OBD-II reader. These applications tend to be directed toward auto enthusiasts, developing features that concentrate on measuring vehicle performance and troubleshoot mechanical issues [2]. Other applications are surfacing that focus on environmental concerns [3]. These applications concentrate on factors like measuring a driver’s carbon footprint and fuel consumption. Some of these applications
include features that can detect safety issue (e.g. issues with the vehicle’s stability control system). However, these features are focused on detecting mechanical issues with safety equipment, not on identifying real time concerns with the driver’s behaviour or environment.

1.3. Mobile Application Concept

This paper walks through the development process of a mobile application that’s primary objective is to capture, measure, and warn users of unsafe and inefficient driving.

“How’s The Ride?” is a software application that will monitor, record, and display real time vehicle data to the driver of an automobile. The application will provide real time feedback enabling the driver to adjust his or her driving style to be safer, smoother, and more efficient.

The application is a prototype. The end product, as not yet fully developed, will require more robust testing and feature tuning before it can be considered a marketable application. Instead, the prototype presented in this paper should be treated as a proof of concept determining if a mobile device can be robust enough to capture, measure, and alert drivers of unsafe and inefficient driving behaviour. In addition the equipment must be practical in price and ease of use.

2. Requirements

Although this application is a prototype, the application will have some general, yet strict, functional and non-functional requirements.

2.1. Functional and Non-Functional Requirements

1) Real Time View Safety and Efficiency Measurements: The application must provide a minimum of three of the following measurements to track safety and efficiency listed in Table 1.

2) Logging of Data: When an issue is detected, the application will log a summarized version of the issue on a five second interval.

3) Reflective View: The application will provide a reflective view that displays unsafe, inefficient, or questionable driving behaviour from the summarized data collected in requirement #2.

4) Platform: The application will run on a modern smartphone device.

5) Performance: In real time mode, vehicle measurements displayed to the user must be updated at a minimum of once per second.

6) Performance: In real time mode, all time critical measurements dependent on third-party devices (e.g. network communication) must attempt to update once every minute.

7) Usability: If a measurement requiring a third-party is not accessible, a message indicating that the measurement is not available should be displayed to the user.

8) Performance: When the application is monitoring the driver in real-time mode, the application should not require direct interaction with the touch screen.

3. Design

For this application, five distinct areas require design attention. First, hardware is selected. During this process special attention is given to ensure the hardware is capable of meeting the requirements that have been set. Second, based on the hardware selected and the requirements set forth, formulas are selected to calculate the measurements. Third, an intuitive user interface must be designed. Fourth, a high level architecture is designed to support the requirements, as well as allow the software to be maintainable. Finally, a test plan is developed around the requirements and the expected interactions with the application.

3.1. Hardware Selection

1) Selecting a Smartphone Device: Today there are five dominant smartphone devices on the market.

- Apple iPhone OS
- BlackBerry OS
• Microsoft Windows Mobile
• Android OS

For this application, the iPhone OS has been targeted. Although many of the other smartphone devices offer several of the same features as the iPhone, the number of applications available, the consistency of hardware between different versions of the device and the potential to run the application on non-phone mobile devices (e.g. iPod Touch & iPad) make the iPhone OS the most attractive.

2) Selecting an OBD-II Reader: There appear to be four popular devices on the market compatible with the iPhone. Of the devices researched they all connect to the iPhone in one of two ways:
• WiFi Connection – the OBD-II reader connects directly to the ECU via a wired connection. Then, the device creates a connection to the iPhone via a WiFi connection.
• Direct Connection – The OBD-II device connects directly to the ECU via a wired connection. Then, the iPhone connects directly to the OBD-II reader through a wired connection.

Although the use of a WiFi connection seems convenient, these devices were more expensive and had the potential to complicate the connection process for the end-user. In addition, a directly connected device had the added benefit of being able to charge the iPhone. Potentially this is very useful, as the mobile device’s interface will need to remain active while in real time mode and the iPhone’s network and GPS interactions quickly drain the battery. Ultimately, the goLINK OBD-II device was selected.

3.2. Measurement Selection

During the measurement selection process, the list of measurements declared in Table I were examined. These measurement requirements were matched against the capabilities of the hardware of the iPhone to determine if it had the means to capture the data needed for each measurement. This information is displayed in Table II.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>iPhone</th>
<th>OBD-II Reader</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleration</td>
<td>Accelerometer</td>
<td>Comparing OBD-II speed reading with previous reading and factoring in time.</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
<tr>
<td>Deceleration</td>
<td>Accelerometer</td>
<td>Comparing OBD-II speed reading with previous reading and factoring in time.</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
<tr>
<td>Speed of Vehicle</td>
<td>GPS &amp; Time</td>
<td>Reading from OBD-II Device.</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
<tr>
<td>Location</td>
<td>GPS</td>
<td>Reading from OBD-II Device.</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
<tr>
<td>Weather &amp; Traffic</td>
<td>3rd party</td>
<td>Reading from OBD-II Device.</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
<tr>
<td></td>
<td>Internet service</td>
<td>Direct Reading converted to MPH</td>
<td>Must be supported by vehicle ECU.</td>
</tr>
</tbody>
</table>

As indicated by Table II and Table III, in a few instances, either device could capture some measurements. In these cases, the device that had the potential to provide the most accurate data was selected. Note that the iPhone will perform the calculations required for each measurement even when the information to support the calculation is supplied by the OBD-II reader.

Table IV shows the measurements that were selected for this application and how the data for each will be collected and how the measurement will be calculated.
Further explanations of the formulas presented in Table IV are as followed:

Miles travelled = \((\text{VS} \times 88/5280) \times \text{SR}\)

\(\text{VS} = \) Vehicle Speed in MPH
\(\text{SR} = \) Sample OBD-II Read Rate in seconds

The formula for fuel consumption is proprietary but is calculated using direct readings from the OBD-II device. Many fuel consumption formulas that use data collected from OBD-II readers are publically available.

3.3. Interface Design

The interface is kept simple. Four separate views (states) are designed.

1) Info View: Figure 1 shows the Info View. The Info View provides basic instructions on how to use the application. It also provides access to the settings view where the sensitivity of the application can be adjusted.

2) Settings View: Here the thresholds of the measurements that alert users of unsafe or inefficient driving can be adjusted. Figure 2 shows the Settings View. This view is necessary because each vehicle is different. Although default settings are provided, this view gives a user the ability to start with less strict settings and decrease the threshold as their driving improves.

3) Real-Time View: Figure 3 shows the Real-Time View. This is the main view that will be displayed when a user is driving. This view requires no user interaction, as this would distract the driver. Real time data is captured and displayed to the user while this view is active. The three gauges at the top of the view display RPM, MPH and Acceleration in FPS². When one of these three measurements is met or surpassed, the reading for that measurement turns from yellow to red, and an alert is logged. A pin, representing the alert, is then dropped on the map to represent the location where the event occurred.

Average MPG and Instant MPG will also turn red if the vehicle reaches or surpasses the set threshold. However, these measures will not log alerts, as they are not strongly linked to location. Also, the fluctuation of instant MPG is too frequent for a summarized view of this information to be valuable.

Two other measurements are included in this view. These measurements are added because they are part of the MPG calculation. These measurements are Gallons Used and Miles Driven. Although they may not directly relate to measuring efficiency, they provide value to those who are tracking their fuel mileage. These two measurements are saved when the application is shutdown and restarted. This maintains the Average MPG measurement over time.

4) Historic View: Figure 4 shows the Historic View. It will be interacted with when the user is no longer driving. This view gives the user a way to reflect on their recent driving behaviours by providing a summary of the last 50 alerts where the thresholds for RPM, MPH, or acceleration were reached or surpassed. These alerts are plotted on the map in the locations where they occurred. When the alert pin is touched, the date, time, RPM reached, MPH reached and acceleration reached are displayed.

3.4. Software Design

The software has been designed to accommodate change. The high level architecture for this application is based on the Model-View-Controller (MVC) design pattern which is shown in Figure 5. In the MVC design pattern the model object encapsulates the application’s data. Views are the windows, controls, and other elements the user can see and interact with. The controller acts as the intermediary between the view and model, giving the views proper access to the model’s data while providing the application logic to
handle user interactions and coordinate the tasks of the application.

Fig.5. Model View Controller (MVC).

The architecture was based on MVC for two reasons. First, MVC decouples the models and views, reducing the complexity of the design and increasing the software's flexibility and maintainability. Second, MVC is relied upon heavily in the iPhone Software Development Kit (SDK) and many of the controls and features available in the iPhone SDK require that objects be implemented to play one of the roles defined by MVC.

Table V gives a very high level glimpse of the models, views, and controllers that make up this application.

### TABLE V
HIGH LEVEL MVC DESIGN

<table>
<thead>
<tr>
<th>Class</th>
<th>MVC Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrivingAlert</td>
<td>Model</td>
</tr>
<tr>
<td>DrivingAlertAnnotation</td>
<td>Model</td>
</tr>
<tr>
<td>OBDIIReading</td>
<td>Model</td>
</tr>
<tr>
<td>SavedSettings</td>
<td>Model</td>
</tr>
<tr>
<td>HistoricView</td>
<td>View</td>
</tr>
<tr>
<td>InfoView</td>
<td>View</td>
</tr>
<tr>
<td>RealTimeView</td>
<td>View</td>
</tr>
<tr>
<td>SettingsView</td>
<td>View</td>
</tr>
<tr>
<td>HistoricViewController</td>
<td>Controller</td>
</tr>
<tr>
<td>InfoViewController</td>
<td>Controller</td>
</tr>
<tr>
<td>OBDIIController</td>
<td>Controller</td>
</tr>
<tr>
<td>RealTimeViewController</td>
<td>Controller</td>
</tr>
<tr>
<td>SettingsController</td>
<td>Controller</td>
</tr>
<tr>
<td>SettingsViewController</td>
<td>Controller</td>
</tr>
</tbody>
</table>

3.5. Test Design

Two simple tests are designed for the application. Both tests are comprised of a basic checklist. The test shown in Table VI verifies that the application is registering measurements and working correctly in RealTime View. Also, accuracy ratings are assigned for each measurement:

- high (within 5% of actual)
- med (within 15% of actual)
- low (below 15% of actual)

The second test, not listed, ensures that all functional and non-functional requirements have been met.

### TABLE VI
MVC TESTING

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Available</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>MPH</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>ACC</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>Gallons Used</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>Average MPG</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>Instant MPG</td>
<td>x</td>
<td>high, med, low</td>
</tr>
<tr>
<td>Miles Driven</td>
<td>x</td>
<td>high, med, low</td>
</tr>
</tbody>
</table>

4. IMPLEMENTATION AND PROGRAMMING

The implementation and programming for the application was straightforward. However, there were two challenges. The first challenge involved capturing data from 3 different sources (i.e. the user, the OBD-II reader and the iPhone) while in the RealTime View. The second challenge was the difficulty involved in debugging the application, as this often meant having to connect the OBD-II reader to a running vehicle.

4.1. General Implementation Guidelines

The application was written in Objective-C. During the implementation process, special attention was given to how the OBD-II reader controller communicated with the remainder of application. The interface methods for this controller were kept very general (e.g. Connect, StartSession, RequestPID, Disconnect). Data returned from the OBD-II device was broadcasted by the OBDIIController to the rest of the application using a generic notification scheme. This allows for other OBD-II controllers to be written for other OBD-II devices without the need to modify the rest of the application code.

4.2. Real Time Interactions on Different Time Intervals

As explained in the design phase, the application has four high level states. These states are represented by the four high level views. The transitions between the states are show in Figure 6, a High Level State Chart of Application Views.
Three of the four states (Info View, Settings View and Historic View) are quite simple. The events handled in these states happen synchronously. However, the RealTime View is more complicated and is required to handle asynchronous events. A state chart has been included in Figure 7 to show the asynchronous events and states that occur in RealTime View.

![State Chart of RealTime View](image)

Fig.7. State Chart of RealTime View

Note that in RealTime View, the only direct interaction the user has with the mobile interface is the ability to change to a different view or exit the application. This intentional design element blocks the user from interacting with the mobile device while driving. In addition, this allows the bulk of processing to be dedicated to calculating and displaying the measurements collected from the OBD-II reader and GPS.

4.3. Debugging

Difficulty was often encountered when attempting to debug an issue. In order to receive the necessary data to exercise the features of the application, the OBD-II reader needed to be connected to the iPhone, the ECU, and a laptop computer. This forced several development hours to be spent inside of a running vehicle, wasting fuel and making the development environment uncomfortable.

5. Testing

Testing was performed on three separate vehicles:
- 1997, Chevy 1500 Light Truck
- 2003, Chevy Impala
- 2004, Ford Escape

Although the application was tested on all three vehicles, the majority of accuracy testing could only be performed on the 2004, Ford Escape due to limited time and resources. All measurements and views did function on all three vehicles. Therefore, these results were excluded from Table VII.

Accuracy tests were very basic. Actual comparisons for RPM, MPH, and Miles Driven were all pulled directly from the vehicle’s instrument panel. Average MPG was based on the miles reported by the vehicle’s odometer, divided by the gallons used to refuel the vehicle. Instant MPG and acceleration (ACC) could not be tested for accuracy with the resources available. The results of the remaining accuracy testing are listed in Table VII.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>1997 Chevy Light Truck (Accuracy)</th>
<th>2003 Chevy Impala (Accuracy)</th>
<th>2004 Ford Escape (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPM</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>MPH</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>ACC</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Gallons Used</td>
<td>?</td>
<td>?</td>
<td>med</td>
</tr>
<tr>
<td>Average MPG</td>
<td>?</td>
<td>?</td>
<td>med</td>
</tr>
<tr>
<td>Instant MPG</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Miles Driven</td>
<td>high</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

All functional and non-functional requirements were met or exceeded by the application.
The majority of testing went well. However, two defects were discovered during the use of the application. These defects were:

- Occasionally the connection between the application and the OBD-II reader was lost. The issue was quickly resolved by restarting the application.
- Sometimes a higher than expected measurement appeared for the acceleration when logged alerts were viewed in the Historic View.

6. MAINTENANCE

Although this application is a prototype, the maintenance phase for the application can be treated much like the maintenance phase of a released application. Instead of having the goal of releasing the next version of the application, the goal here is to move from a prototype to a marketable application.

6.1. Correct Defects and Improve Robustness

The defects discovered need to be addressed, either through a deferral process or by implementing corrections. Testing needs to be expanded to include more vehicles, and user acceptance testing should take place to measure ease of use and usefulness.

Weaknesses in functionality should also be explored. For example, testing showed the application’s Average MPG measurement to be within 15% of the actual Average MPG. Adding a setting that allows users to calibrate the MPG formula for their vehicle may be helpful.

6.2. Expansion of Supported Hardware Platforms

The flexible MVC architecture should be leveraged to add more hardware support for both mobile devices and OBD-II readers.

The iPod Touch runs the same iPhone based OS. However, the iPod lacks some of the functionality supported by the iPhone, like GPS. Adding a View that displays alerts in a table format, rather than on a map, would allow iPod Touch users the ability to view the data in a more meaningful way.

During the development of the prototype, debugging the application with the available resources proved difficult. Often, development had to occur while directly connected to a running vehicle. To improve efficiency in development, one of two options should be considered. One, create a mock (software based OBD-II) hardware device. This mock device would imitate a real OBD-II device. This would allow the core of the application to be developed and debugged without having to connect to a vehicle. Two, if available, use an OBD-II/ECU simulator to debug the application. Again, this avoids having to connect to an actual vehicle while developing and debugging.

Similarly, expanding the number of supported OBD-II reader devices (i.e. WiFi connection based) should be explored. Since the interface developed for the OBD-II controller was generically written, adding additional OBD-II hardware devices should require minimal changes.

7. RESULTS AND CONCLUSION

The initial results of the prototype are promising. The potential uses of software applications, like the one developed in this paper, are enormous. These applications can become very powerful tools when they are leveraged with the features of the smartphone and the data provided by the OBD-II reader.

7.1. Strengths

The applications performance exceeded expectations. As the test results show, the application provides high to medium accuracy for all measurements tested. Also, the simple interface, convenience of the smartphone, and relatively low cost of the OBD-II reader makes this application a viable solution to monitor and alert drivers of unsafe and inefficient driving.

7.2. Weakness

While the application shows a considerable amount of potential, there are weaknesses. The indentified weaknesses include:

1) Every reading available through the OBD-II standard is not available on every vehicle. This is not because the manufacturer is not compliant with the OBD-II standard, but due to the differences in vehicles. For example, some vehicles are not MAF equipped. The fuel consumption formula used in this application relies on this reading. Time and logic will need to be invested in determining which readings are available for a specific vehicle and new formulas may need to be developed.

2) Testing for this application was very limited and basic.

3) Usefulness was not tested. The question remains, “Can this application make one a safer or more efficient driver?”

7.3. Conclusion

As the tests results indicated, initial findings show the software is able to provide high to medium accuracy for all of the measurements tested. Though there were weaknesses identified, most can be mitigated by increasing the robustness of the code, performing more testing, and adding features that allow the user to calibrate the application to fit their vehicle and driving style.

The prototype was successful in providing a user-friendly method to make drivers aware of unsafe and inefficient driving practices. Also, the application provided a unique Historic View that allowed users to reflect on their own historical patterns of inefficient and unsafe driving. This application, combined with a smartphone and an OBD-II reader, is one of the few practical opportunities a driver may have to monitor, reflect on, and improve their overall driving behaviour.
ACKNOWLEDGMENTS

We would like to thank Brennan Hamilton, President of GoPoint Technology, USA, for donating his time and the use of his goLINK OBD-II reader. His example code and MPG formulas were of great value in allowing me to complete the prototype application.

REFERENCES