

Copyright

by

Paul Adam Chandler

2012

**The Report Committee for Paul Adam Chandler
Certifies that this is the approved version of the following report:**

MileageAIDE: A System for Business Mileage Tracking

**APPROVED BY
SUPERVISING COMMITTEE:**

Supervisor:

Adnan Aziz

Mark McDermott

MileageAIDE: A System for Business Mileage Tracking

by

Paul Adam Chandler, BSEE

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2012

Abstract

MileageAIDE: A System for Business Mileage Tracking

Paul Adam Chandler, MSE

The University of Texas at Austin, 2012

Supervisor: Adnan Aziz

This report describes the MileageAIDE system for recording the business mileage of a vehicle. The system consists of a mobile application for an Android phone and a hardware device connected to the vehicle's On-Board Diagnostic port (OBD-II). The two interact via Bluetooth to record the time and odometer value for every trip in the vehicle and remind the user to classify the trip as business or personal. The result is a system that is easy to install, easy to use, inexpensive and keeps detailed and accurate records that are acceptable for reporting to an employer for reimbursement or as business expenses on a tax return.

Table of Contents

List of Tables	vii
List of Figures	viii
Chapter 1: Introduction.....	1
1.1 Motivation.....	1
User Story 1	2
User Story 2	2
User Story 3	3
User Story 4	3
1.2 Report Organization.....	3
Chapter 2: Goals and Requirements	4
2.1 OBD-II Review	5
2.2 OBD-II Design Constraints.....	6
Chapter 3: Automobile Information Device (AIDE): Hardware and Firmware.....	8
3.1 Hardware.....	8
3.2 Firmware and Operation	10
3.2.1 Power Up	10
3.2.2 Bluetooth Communication.....	11
3.2.3 Communication with Vehicle	12
3.2.4 Low Power Standby.....	15
Chapter 4: Business Miles Mobile App.....	17
4.1 BusinessMiles Activity	17
4.2 TripEdit Activity.....	21
4.3 Database Setup and Operation.....	22
Chapter 5: Testing and Evaluation.....	23
5.1 Development.....	23
5.2 Accuracy	24

5.3 Timeline and Code Size	25
Chapter 6 Conclusion.....	28
6.1 Future Work.....	28
Bibliography	29

List of Tables

Table 1:	Lines of Code in Business Miles App.....	25
----------	--	----

List of Figures

Figure 1:	Female OBD-II Connector Pin Diagram	5
Figure 2:	Olimex TMS470-P256 Development Board	8
Figure 3:	Olimex MOD-BT Bluetooth Extension Board	9
Figure 4:	ELM327 RS-232 to CAN Interface Board	10
Figure 5:	Bluetooth Connection Flow-chart.....	12
Figure 6:	ELM327 Message Processing Flow-chart	14
Figure 7:	Top Level AIDE Operation	16
Figure 8:	BusinessMiles Main Screen.....	18
Figure 9:	Set Odometer Dialog Box.....	19
Figure 10:	Trip Purpose Alert Dialog.....	20
Figure 11:	Trip Edit Screen and Trip Purpose Pull-down Menu.....	21
Figure 12:	AIDE Prototype Connected to Test Vehicle.....	26
Figure 13:	BusinessMiles Main Screen on LG Optimus S Smart Phone	27

Chapter 1: Introduction

This report focuses on the MileageAIDE system for recording the business miles driven in a personal vehicle. The system consists of:

1. The Automobile Information DEvice (AIDE), a universal OBD-II (On-board Diagnostic port) to Bluetooth adapter and odometer tracker.
2. The BusinessMiles Android Mobile app.

1.1 MOTIVATION

The Internal Revenue Service requires adequate records to deduct business expenses. An adequate record is a detailed record prepared at or near the time the business expense is incurred [1]. The details required vary slightly based on the nature of an expense. For business use of a car the details required include the cost of the car; the date it was put into business use; the mileage, date, destination and purpose of each business use; and the total miles placed on the vehicle for the year. The cost of the car and date it was put into service for the business are one-time events that are easily recorded with documentary evidence. The other items are often difficult for the individual self-employed person to keep timely and accurate records. During a busy day, it is easy to forget to check and record an odometer value.

Currently, there are many organizational tools available on the Android Market used to enter and store the details of a business trip [2]. Some of the most easy to use apps only need the user to press one button to indicate the start and stop of a trip and the app calculates and records the distance using GPS information. However, all of these require the user to remember to start and stop the trip logger. Also, GPS places an extra strain on the phone's power and processing limitations and any signal interference results in inaccurate routing and distance calculations.

There are also many mileage tracking devices on the market that use GPS, OBD-II information, or both [3]. These simply record every trip in the vehicle without discriminating between personal and business. They are better suited for small businesses with a vehicle used exclusively for business. An individual using these systems on a personal vehicle used for business would have to download the trips and attempt to remember the purpose of each trip.

I have created a more effective solution for individuals who use their family car for both business and personal and must create complete and accurate mileage reports for their business.

User Story 1

A real estate agent uses the family car for her business and will be showing a couple several houses during the course of a day. When she starts the car, the permanently mounted hardware detects the start and sends a notice to her Android phone. The phone alerts the real estate agent to categorize this trip as personal, business, or a continuation of any business she may already be engaged in. She selects “New Business” and enters an identifier for the trip manually or selects it from a list of appointments or contacts on her phone. The odometer value received from the AIDE along with the starting time is recorded for the trip in a database on the phone. The real estate agent meets the couple at the first house and shuts down the engine. The AIDE detects the engine stop and notifies the smart phone before returning to sleep mode. The ending odometer value and ending time are recorded for the trip on the smart phone.

User Story 2

For individuals concerned with the environment or gas prices a combination of AIDE and smart phone can improve their driving habits. The AIDE can emit an

unpleasant noise through the device itself or the smart phone if excessive acceleration is detected.

User Story 3

A family concerned for the safety of their newly licensed teenager can record his driving habits on the AIDE. Also, notifications can be sent to them from his phone if he is driving recklessly or any malfunctions are detected including those caused by a wreck [4]. The AIDE can notify the smart phone when the vehicle is in motion and the phone will limit the use of calls and texts.

User Story 4

A car enthusiast can download error codes, monitor vehicle information [5], potentially alter the engine performance, or control various vehicle systems from his smart phone [6].

1.2 REPORT ORGANIZATION

First, I describe the goals and requirements of a mileage tracking system in Chapter 2. I describe the hardware, firmware and operation of the AIDE in Chapter 3. In Chapter 4, I describe the software and operation of the BusinessMiles Android Mobile App. The issues encounter during development and the accuracy of the system are covered in Chapter 5. I summarize the project and provide ideas for future work in Chapter 6.

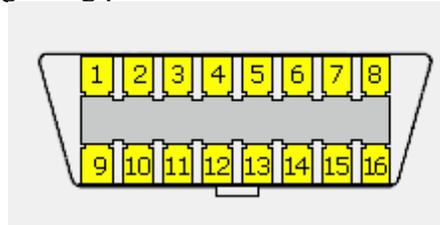
Chapter 2: Goals and Requirements

The mileage reports must record the actual odometer reading during the start and stop of each business trip, the start and stop time, and the business purpose and destination. To accomplish this, the start and stop of each trip will be based on engine start and stop events. The business purpose can only be known by the user, so in order to record that information in a timely manner we will alert the user with a notification icon and alarm. This will remind the user to manually enter the business purpose and destination. All of this information will be recorded on the user's smart phone. Since the smart phone is a personal device we can use its presence during an engine start event to determine if a trip should be considered personal or business. If another family member is using the car and the business user is not present than the trip can be considered personal and will not even need to be recorded. So, the device to be placed in the vehicle will be required to determine the odometer reading at each engine start and stop event, search for the business user's smart phone, and interact with that smart phone to provide the odometer reading and engine status information. The Android smart phone will provide the user interface to the entire system, interact with the AIDE and keep a detailed record of all the trips taken by the business user.

The AIDE should be small, inexpensive, easily installed and capable of interacting with a broad range of vehicles and smart phones. Fortunately, most smart phones are enabled with Bluetooth wireless technology. Also, as of 2008, all cars and light trucks sold in the United States are required to have an On-Board Diagnostics port (OBD-II) utilizing the Controller Area Network (CAN) protocol.

2.1 OBD-II REVIEW

The OBD-II standard is an extension of previous vehicle diagnostic and emission related features [7]. It specifies the standard connector to use, the pin placement on the connector (Figure 1), the signaling protocols used and the messaging format.



Pin	Signal
2	J1850 Bus+
4	Chassis GND
5	Signal GND
6	CAN High
7	ISO 9141-2 K-LINE
10	J1850 Bus-
14	CAN Low
15	ISO 9141-2 L-LINE
16	+12v

Figure 1: Female OBD-II Connector Pin Diagram

CAN is a multi-master broadcast serial bus standard for connecting microcontrollers [8]. Each node on the bus is able to send and receive messages using message priority bus arbitration. If two or more nodes begin to transmit simultaneously, the node sending the most dominant message ID will overwrite the less dominant IDs. The nodes sending the less dominant IDs will detect this overwrite and stop transmitting their message until a later time. Thus, only the most dominant message is fully

transmitted. If the bus is free, any node may begin to transmit and all other nodes will receive the message. Each message contains a destination ID so only the correct receiving node will act on the message.

2.2 OBD-II DESIGN CONSTRAINTS

The OBD-II standard provides many standardized diagnostic codes, but none for reading the odometer value. However, there is a standard OBD-II diagnostic code for reading the vehicle's current speed. With that we can query the vehicle's speed at regular intervals, calculate the distance it has traveled and maintain an odometer value on the AIDE that closely matches the actual vehicle odometer value. To meet user expectations for accuracy, I decided it will need to maintain an odometer value within 1 mile of the vehicle's odometer value over the course of a year. Assuming an average miles per year of 20,000 this would be an error not more than 0.005%.

To select the size of the time interval, several factors were taken into account. The speed requests must be frequent enough to reduce the size of any vehicle speed changes between time intervals, but not so frequent that the CAN bus is overloaded. Also, the data structure used to record the odometer value, and the minimum and maximum distance recorded with each time interval should be considered. A time interval one-half to one second was hypothesized to provide satisfactory accuracy during vehicle acceleration and deceleration and certainly would not tax the CAN bus. This would also record minimum distance intervals of approximately one foot for the slowest readable speed of 1 km/h and 100 foot distance intervals for highway speeds. I decided to record the odometer value in a single 32-bit unsigned integer. To clearly separate miles from fractions of a mile, I selected a minimum recordable distance that is a power of 2. Using 2^{-13} mile (approximately 8 inches), an unsigned integer would have a maximum odometer value of

$2^{32-13} = 2^{19} = 524,288$ miles. The time interval can be calculated by dividing the minimum recordable distance by the minimum readable speed.

$$1 \frac{\text{km}}{\text{h}} \cdot t = 2^{-13} \text{ miles}$$

$$t = 0.70723 \text{ seconds}$$

This reduces the calculation required at each time interval to simply adding the speed value to the previous odometer value to obtain the new odometer value.

Chapter 3: Automobile Information Device (AIDE): Hardware and Firmware

3.1 HARDWARE

For the development board, I needed an inexpensive board that had a variety of communication ports and debug features. The Olimex TMS470-P256 (Figure 2) is a \$90 board that has RS-232 and CAN ports, an SD card slot, and an extension port that accepts the Olimex's MOD-BT module for Bluetooth communication. For debug and user interface, it has an LCD screen, a JTAG port, LED indicators and user input buttons. It features a Texas Instruments TMS470A256 16/32-bit ARM7TDMI microcontroller with 256KB Flash memory and 12KB SRAM [9].

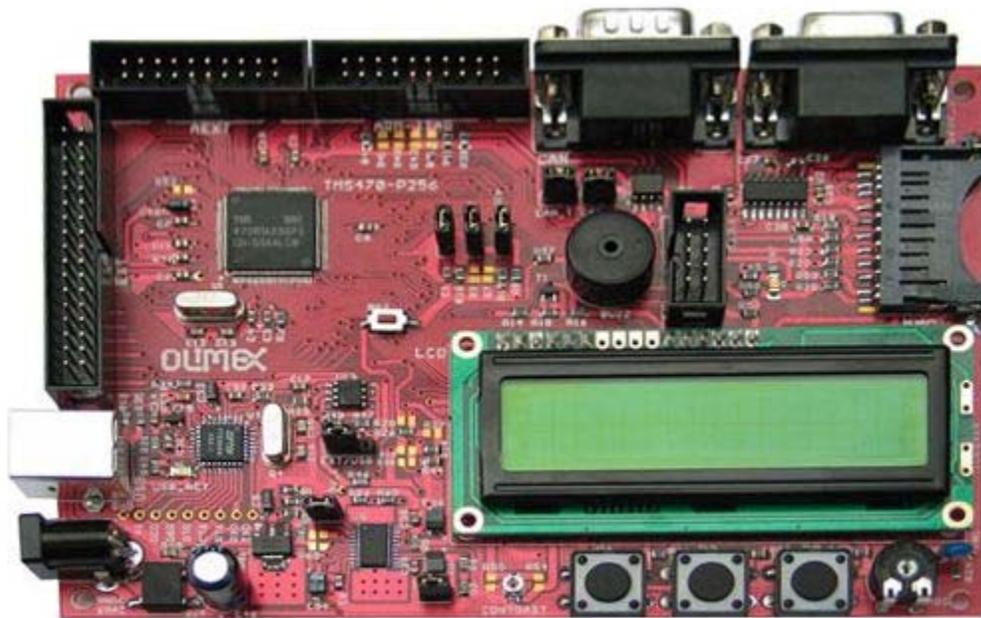


Figure 2: Olimex TMS470-P256 Development Board [9]

To communicate with the Android smart phone, the \$30 MOD-BT extension board from Olimex was used (Figure 3). It connects directly to the main board using a ten pin connector that provides power and a UART connection. It provides an easy to use

Bluetooth Serial Port profile using Philips BGB203 Bluetooth System-in-a-Package radio with baseband controller [10].

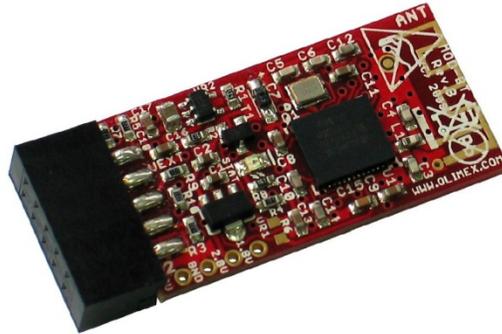


Figure 3: Olimex MOD-BT Bluetooth Extension Board [10]

To ensure the system could be used with older model cars that do not use the CAN protocol, I built an OBD-II to RS-232 interface board (Figure 4) based on ELM Electronics' ELM327 OBD to RS-232 Interpreter [11]. The ELM327 is a microcontroller (Microchip Technology Inc.'s PIC18F2480) and software package. It can communicate with any of the OBD-II standard protocols and automatically handles message headers and check sum bits, allowing the user to focus on the data. The addition of the RS-232 protocol to the system was helpful for debugging and testing as I could simulate the communication to the main board or the ELM board using a computer with a serial port and the terminal emulation program Hyperlink. I created the board based on the circuit design provided in the ELM327 manual. Because the vehicle under test uses the CAN protocol, the circuitry for the other OBD-II supported protocols was left out of the design.

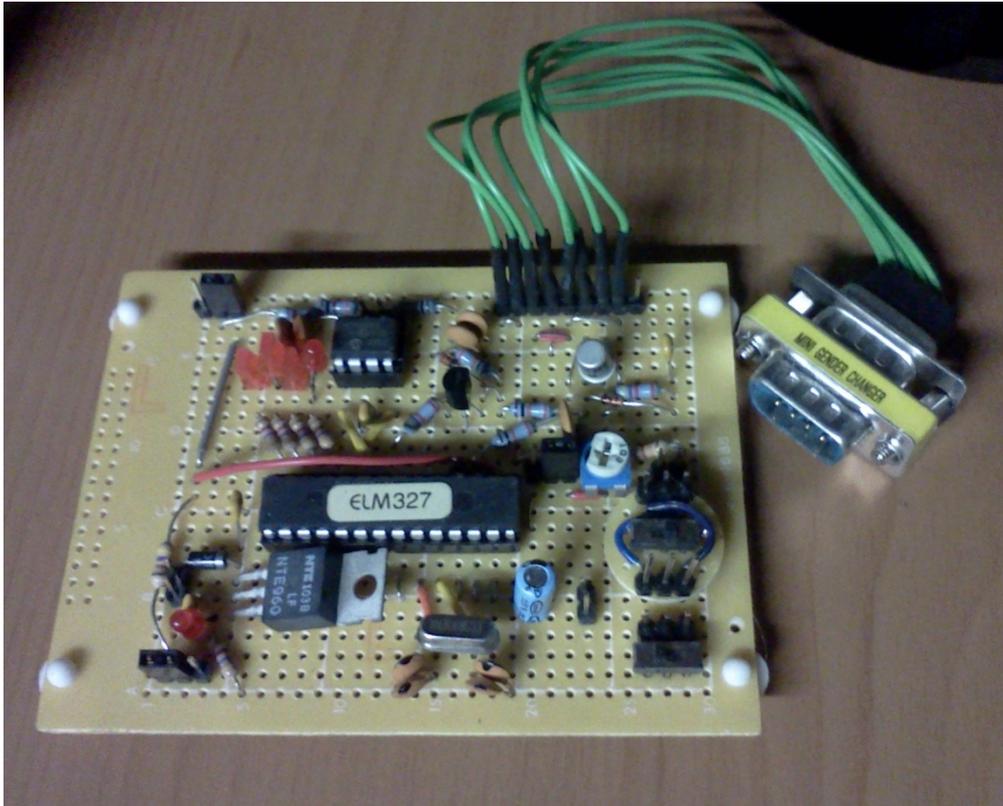


Figure 4: ELM327 RS-232 to CAN Interface Board

3.2 FIRMWARE AND OPERATION

The firmware for the AIDE was written in C using IAR Systems' Embedded Workbench for ARM KickStart edition. The KickStart Embedded Workbench is a free development environment for writing, compiling and linking code designed for embedded devices. It is capable of programming and debugging embedded devices using an interface between the embedded device's JTAG port and the PC's USB or parallel port [12].

3.2.1 Power Up

On initial power up, the firmware sets up the internal clock, interfaces and baud rates. There are two serial communication ports, one set up as a UART that

communicates with the BT-MOD device and the other connected to the ELM327 through an RS-232 port. Serial communication for each port is set up to interrupt and each character is placed in a message buffer until a carriage return character is detected, then the full message is placed in the message FIFO.

A timer interrupt is set to send an identify message to the ELM327 until the ELM responds. At that point the timer interrupt is set to send a request through the ELM to the vehicle for the engine rpm, a request is sent to the BT-MOD to search for the paired mobile device and the TI device enters a continuous loop of checking the FIFOs for new messages.

3.2.2 Bluetooth Communication

The Bluetooth connection procedure (Figure 5) is accomplished through a state-machine setup in the software. The first state, the disconnected startup state, the TI device has requested to search for a serial port profile on the paired device as mentioned earlier. If a serial port profile has been found the port number is recorded and the state is moved to 2, otherwise the state remains 1. State 2 sends a request for a client connection using the recorded port number and moves to State 3. At State 3 if an error message is received the state is returned to 1. If a Bluetooth connection has been established, the state is moved to 4 and the TI device is ready to accept instructions and return information to the Android device.

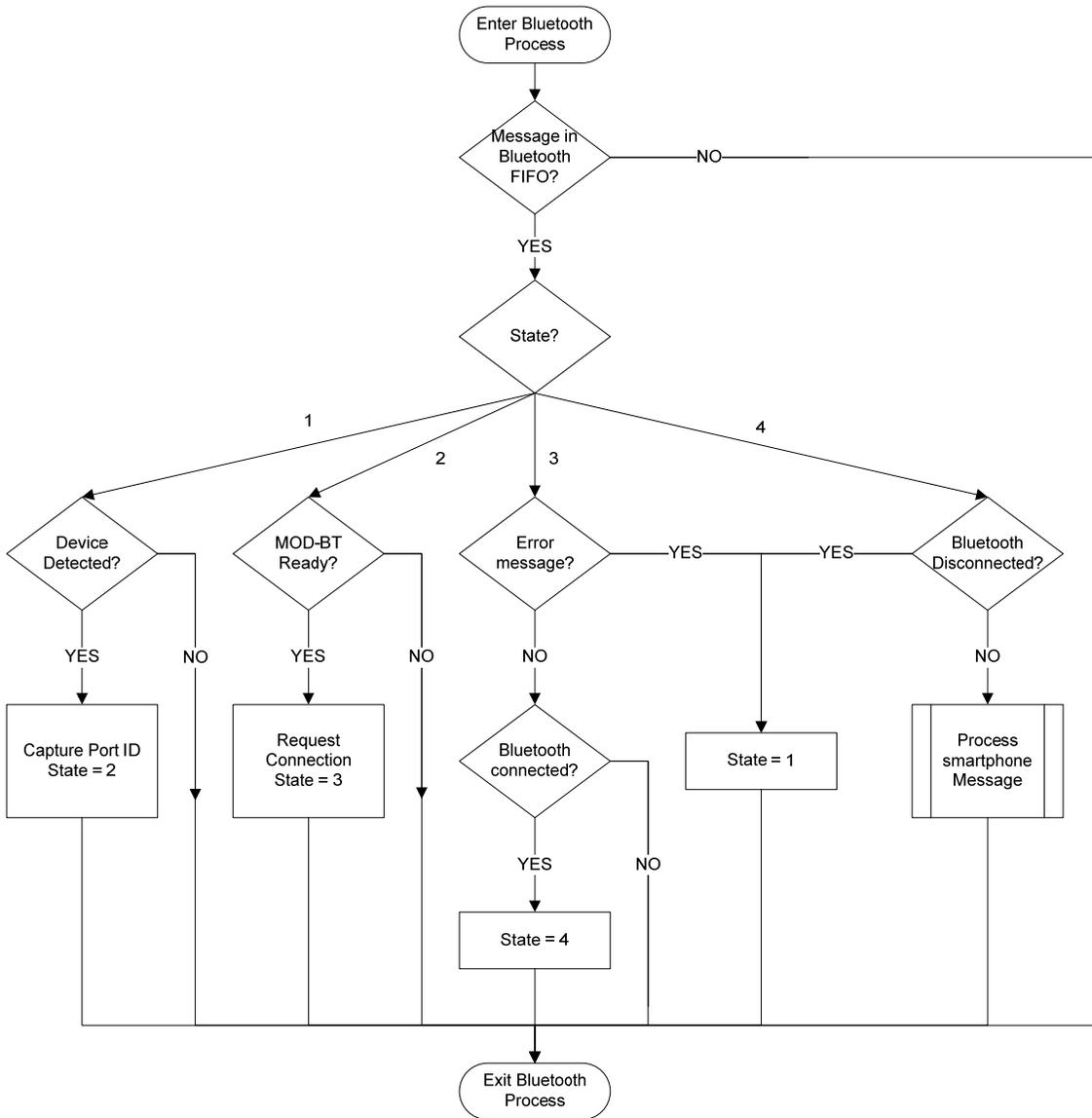


Figure 5: Bluetooth Connection Flow-chart

3.2.3 Communication with Vehicle

The odometer tracking function (Figure 6) operates in basically two states: Engine On and Engine Off. Initially the engine is assumed to be off and the TI device uses the odometer timer to request the engine rpm approximately every half second. When the rpm returned is greater than zero the Engine On variable is set to 1, the device begins to

request the vehicle speed with the passage of each time unit of 0.707 seconds, the number of time units passed is incremented, and an engine start message and starting odometer value are sent to the Android device if connected. If not connected to the Android device, the AIDE requests a connection. The size of the time unit was selected to reduce the number and complexity of calculations needed to convert the vehicle speed in hexadecimal kilometers per hour to a decimal representation of miles. If the vehicle speed is 0, the TI device requests the engine rpm and returns the time unit count to zero. If the engine rpm returns to zero, the Engine On variable is set to 0, an engine stop message and current odometer is sent to the Android device if connected, the current odometer is recorded to the SD card on the TI device, and the timer interrupt returns to checking the engine rpm for an engine start. If the vehicle speed is greater than 0, the odometer calculation function is called.

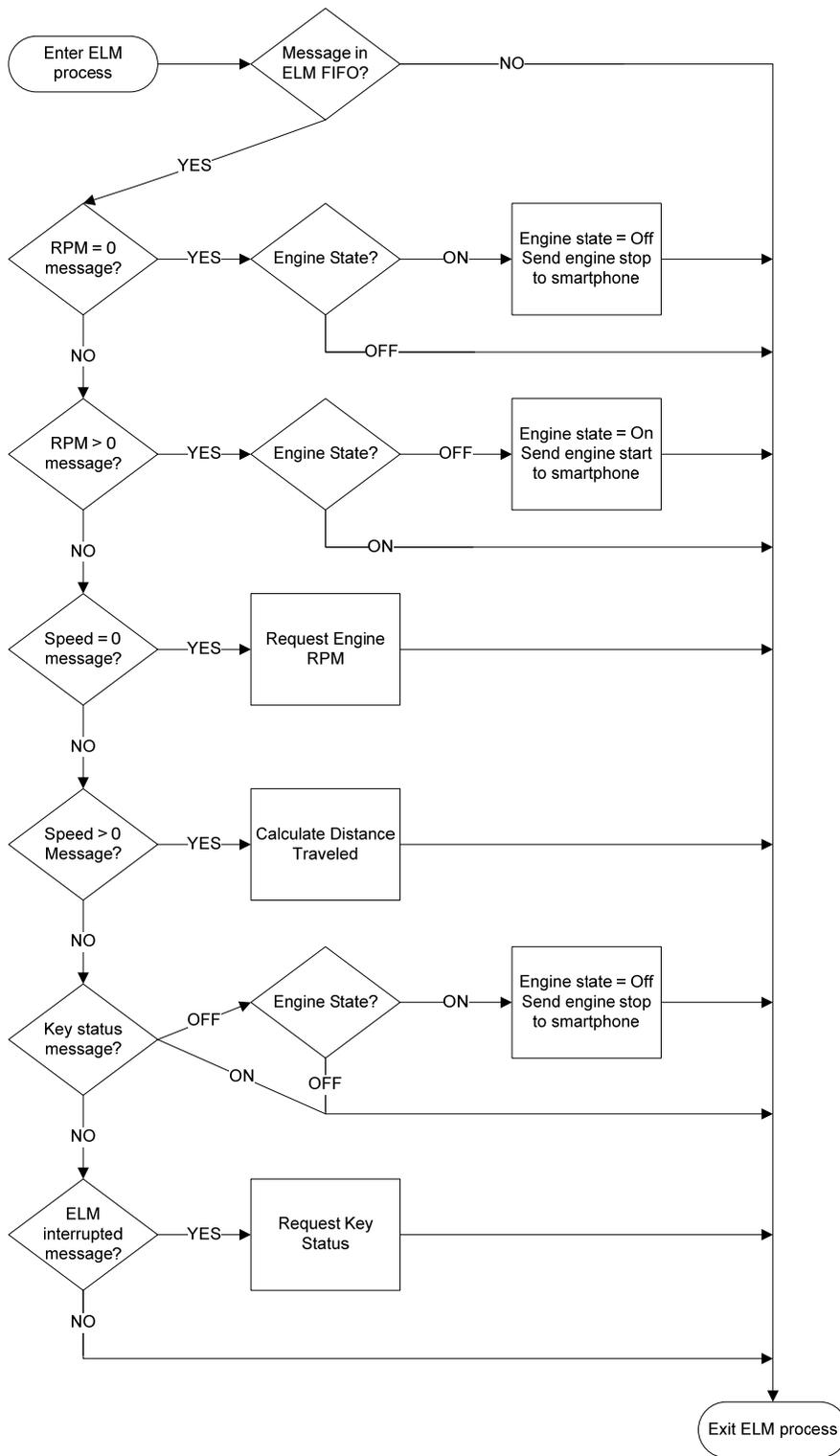


Figure 6: ELM327 Message Processing Flow-chart

3.2.4 Low Power Standby

The test vehicle uses a 45 Amp-hour battery and I measured the current draw while the vehicle was parked and the key removed to be 115mA. The AIDE draws 94mA during normal operation. If left in the vehicle in this mode, it could drain the battery in less than nine days.

$$(115\text{mA} + 94\text{mA}) \cdot t = 45\text{Ah}$$
$$t = 215\text{h}$$

Since the AIDE would remain in the vehicle at all times and the power pin on the OBD-II port is always on, a low power mode was needed to prevent draining the vehicle battery if parked for an extended period of time.

This is accomplished by monitoring the ELM327 for a “STOPPED” message [11]. When received, the device requests the vehicle key status from the ELM. The ELM checks the voltage level of the pin designated as an indicator of the vehicle key position and returns an “ON” or “OFF” message. If an “OFF” message is received, the AIDE prepares to enter low power mode. First, it is ensured that the engine status is set to OFF and the smart phone app, if connected, has completed any communications needed to record the end of the trip. The odometer timer is stopped, the ELM327 is put into low power mode and the Bluetooth connection is terminated. The AIDE now waits for any communication from the ELM327 to initiate the wake-up procedure. When the ELM327 detects the vehicle key turned to the “on” position, it sends a wake-up message to the Olimex board and the wake-up procedure begins. First, the odometer value is read from the SD card and placed in the current and start odometer variables. Then, the AIDE enters the normal operating mode by starting the odometer timer and searching for the user’s smart phone. Figure 7 illustrates the top level flow of the device including the Low Power procedure.

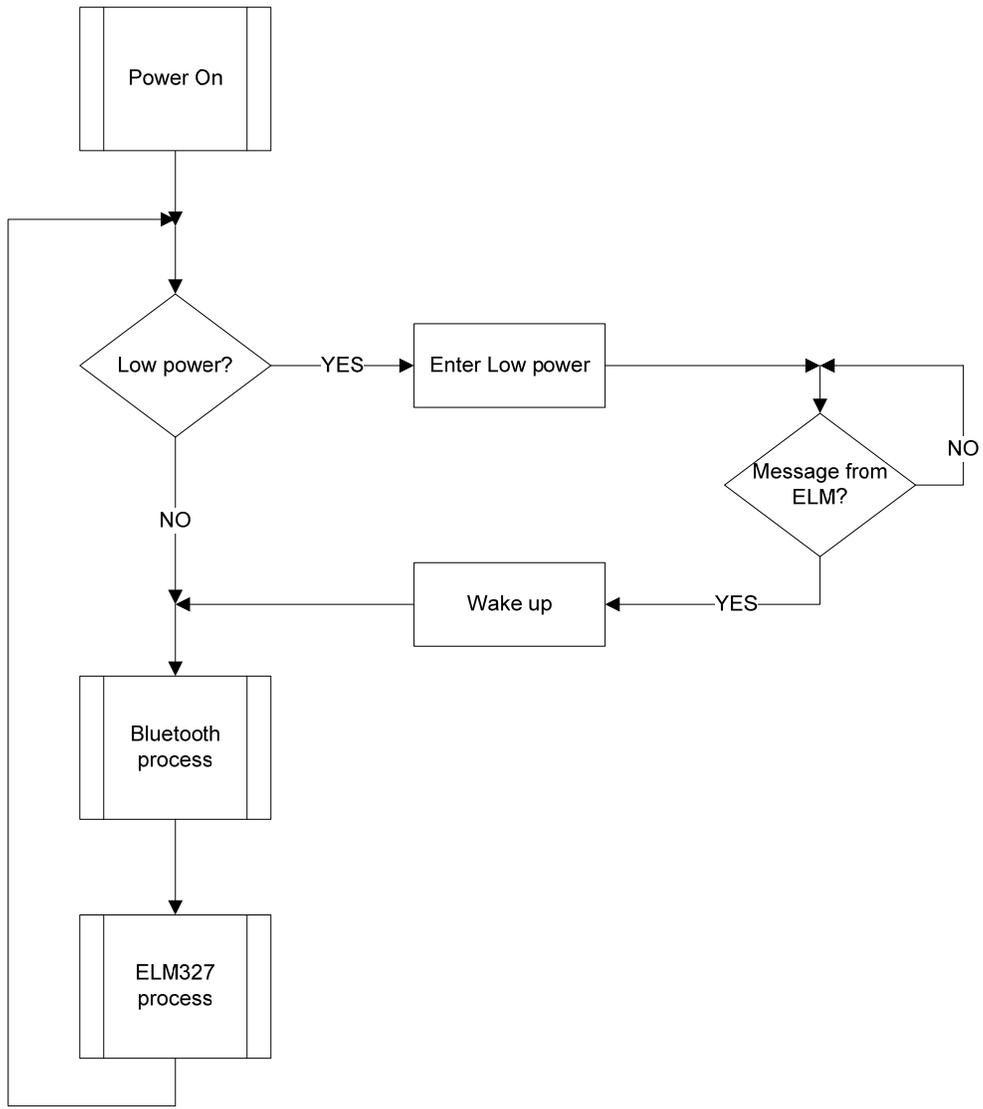


Figure 7: Top Level AIDE Operation.

Chapter 4: BusinessMiles Mobile App

The BusinessMiles App was written in Java using the Eclipse development environment with the Android Software Development Kit. The Android SDK provides an emulator that was used to produce the screen shots featured in this chapter.

I will briefly review the anatomy of an Android application. An Android app consists of a few well defined building blocks: activities, intents, services and views [13].

A view is the user interface. Typically specified in XML files, it handles the content of a particular screen and any user interaction with that content. Views are arranged in a hierarchical structure that consists of a top layout scheme, such as a list or table, composed of other layouts and/or common interface items like buttons and text entry fields.

A service is used to complete background operations such as downloading a file or playing music while the user is using other applications. Services are not associated with a user interface.

An activity performs any function that is associated with one screen of an application. It receives information from the view regarding the user's interaction and performs any intended action, including calling other activities and services. Thus, several activities and possibly services work together to form a cohesive user experience in an application.

Activities and services are called using Intents. Intent is a data structure holding an abstract description of the operation to be performed and can send or request data from the called activity or service.

4.1 BUSINESSMILES ACTIVITY

The main activity of the app is the BusinessMiles activity. It is an extension of the ListActivity Android class [14]. On start up, the activity accomplishes several things.

First, it starts the database adapter and retrieves the title of each trip from the database. It builds the main screen of the app as a list of trip titles (Figure 8). It also starts the Bluetooth service handler.



Figure 8: BusinessMiles Main Screen

At this point, the application waits for inputs from the user or any Bluetooth activity including messages from the AIDE. The user can select the menu button for several setup options. Currently, these setup options are used for debug purposes and include making the phone discoverable to other Bluetooth devices, requesting a Bluetooth connection to another device. If connected to the AIDE the user can set the odometer, retrieve the current odometer or set the AIDE to enter low power. To retrieve the current odometer value, the activity sends a message to the AIDE and it responds with the value. The value is placed in the title bar. In a similar manner, selecting the low power option sends a message to the AIDE and the device enters low power. To set the odometer value on the AIDE a dialog box [15] is opened for the user to enter the odometer value as seen

in Figure 9. Once entered, the value is sent in a message to the AIDE to update the odometer value.

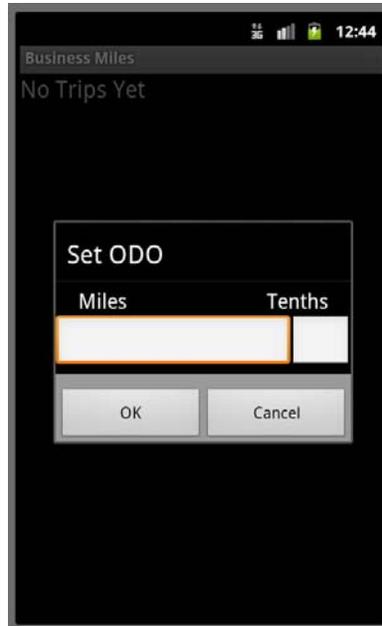


Figure 9: Set Odometer Dialog Box

The main activity interacts with the Bluetooth connection through the BusinessMilesService handler thread. It captures and displays any changes in the Bluetooth connection status as well as sending and receiving a limited number of messages over Bluetooth. The activity is capable of sending messages to the AIDE to notify when it is safe to enter low-power mode and to request the engine status or odometer values. The BusinessMiles activity must maintain its own record of the vehicle engine status. When the application is first opened it is assumed that the engine is off. Once connected to the AIDE, the activity requests the engine status. If the activity receives an engine start message, the starting time is saved, the starting odometer value is retrieved from the AIDE, and a notification is called [16]. The BusinessMiles activity can detect an engine start and send a notification even if the activity has been moved to

the background. The notification is set to alert the user with a sound and an icon in the notification area. It will also bring the BusinessMiles activity to the foreground and open a dialog box to select the trip type (Figure 10). The dialog box retrieves the title of the last trip recorded and gives the user the option of continuing that trip or creating a new trip that is either personal or business.

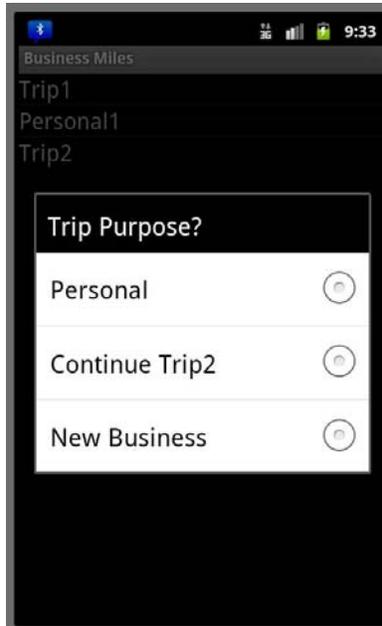


Figure 10: Trip Purpose Alert Dialog

If the user selects “Personal” or “New Business” the BusinessMiles activity sends the database helper a “createTrip” command with the trip type, starting time and starting odometer. The database helper creates the new database entry with the received information and responds with an ID for the new trip. The ID is passed to the continueTrip function and an Intent is sent with the ID to open the TripEdit activity. Selecting “Continue” from the alert dialog simply passes the continuing trip ID to the continueTrip function. The continueTrip function sends the trip ID in an Intent [17] to open the TripEdit activity. There are other events that result in the call of the TripEdit

activity. The user can select a trip from the list on the main screen and the continueTrip function will be called with that trip ID. When an engine stop event is detected, the ending time and odometer are recorded and if the trip type has been selected, the values are sent with the current trip ID to the continueTrip function. If the trip type has not been selected, a new trip is created with an “unknown” type using the createTrip function, which then passes the information to the continueTrip function.

4.2 TRIPEDIT ACTIVITY

The TripEdit activity receives the trip ID and any additional information from the Intent. It opens the trip from the database and populates the screen with the trip information (Figure 11). The user can open a drop down menu to select or change the trip purpose and enter a name for the trip in the Title text entry field. The activity saves any changes whenever the activity is exited by pressing the “confirm, back or home” buttons.

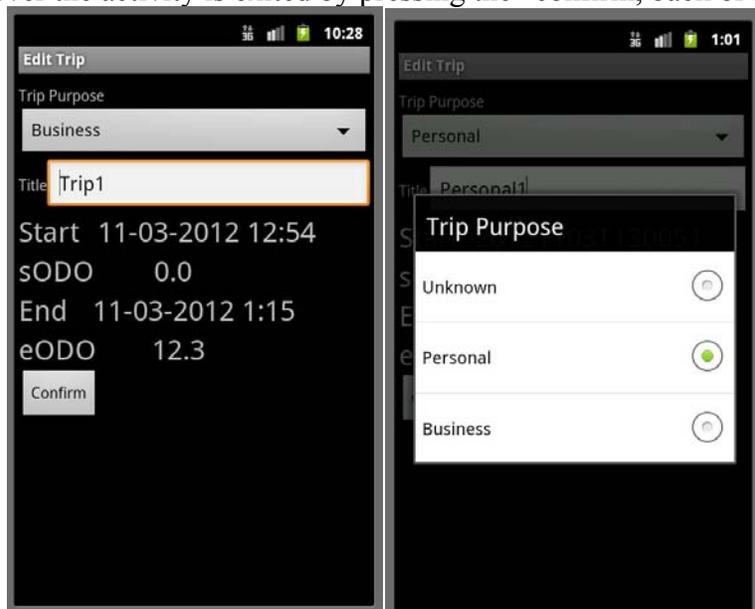


Figure 11: Trip Edit Screen and Trip Purpose Pull-down Menu.

4.3 DATABASE SETUP AND OPERATION

Each trip is stored in a Java-based SQL database which contains the following information for each trip (the data-types for each field are indicated in parenthesis below):

1. Row ID (integer)
2. Trip type: unknown, personal, or business (integer)
3. Title of trip (text)
4. Starting time of trip (text)
5. Starting odometer value (float)
6. Ending time of trip (text)
7. Ending odometer value (float)

The TripsDbAdapter driver class implements the SQL statements for performing the following database operations.

1. Creating the database if none exists already.
2. Creating new trips in the database.
3. Updating trips already in the database.
4. Fetching all trips in the database to populate the trip list in the main activity.
5. Fetching a single trip to populate the TripEdit activity screen.
6. Selecting the most recent trip entered to provide the user the ability to easily continue updating a trip with multiple starts and stops.

Chapter 5: Testing and Evaluation

5.1 DEVELOPMENT

The first version of the firmware used a polling method for reading from the UART ports connected to the Bluetooth and ELM327 hardware modules. This would stop all other operations to attempt to read a full message. If the timer interrupted this message reading operation or a character of the message was dropped, it would result in the message not being acted upon or the system waiting indefinitely to complete the message. I switched to an interrupt method that placed each character into a buffer as it arrived. If a return character was received or the buffer became full, the message in the buffer would be moved into a FIFO array, allowing the embedded device to continue operating at all times even if an incomplete or erroneous message was received as these would just be ignored.

Originally both the ELM327 board and the entire Olimex board were placed into a low power state when the vehicle was shut down. With this set up, the entire AIDE system ran at 94mA when operating and 21mA when in low power mode. However, the device would not consistently come out of low power mode and therefore lose odometer accuracy and fail to connect to the user's smart phone. I could not replicate the failure when debugging so in order to proceed with testing of the odometer accuracy I altered the low power profile. I changed the firmware to allow the TMS470 microcontroller to stay on at all times while still reducing power on the rest of the Olimex board and setting the ELM327 board to low power. The AIDE then operated consistently and would draw 63mA in low power mode.

5.2 ACCURACY

Once working consistently, I began testing the accuracy of the odometer calculation. For long distance highway travel (490.1 miles), the AIDE maintained an odometer value equal to the vehicle's odometer. For in-town driving the AIDE odometer value (573.8) was 3% less than the vehicle odometer value (590.8). To eliminate the possibility that the problem results from more frequent acceleration and braking, the frequency of speed requests was doubled. Instead of reducing the error by half, the error was doubled. The AIDE's odometer value was approximately 6% less than the vehicle's odometer value. To debug further, I set the device to connect and send information to my laptop over Bluetooth. I gathered every speed message received from the vehicle and the number of time units that have passed when the speed message was received. From this information I was able to calculate the mileage and it was accurate when compared against the vehicle's odometer. The recorded information also provided evidence that a reset of the system had occurred. Some messages unique to the power-on sequence would appear in the information and the time since startup would reset. These resets would lose any previously calculated mileage since engine start. I discovered that the reset could be caused by excessive vibration or acceleration. A more robust physical design would be needed in the future.

5.3 TIMELINE AND CODE SIZE

The code size was measured in lines of code using the SLOCCount tool [18]. The firmware for the AIDE totaled 1325 lines of code. The software for the BusinessMiles Android app totaled 1289 lines of code. This is a total of the Java source code only, and not the XML view layout or resource files which are used for building the user interface of an Android Mobile app. The number of lines of code for each activity and service in the BusinessMiles app is detailed in Table 1.

File	Lines of Code
BusinessMilesService	316
BusinessMiles	426
DeviceListActivity	116
TripEdit	129
TripList	18
TripListProvider	189
TripsDbAdapter	95
Total	1289

Table 1: Lines of Code in BusinessMiles App

The code was developed over the course of about two months with approximately 70% of the time spent on the AIDE firmware. An additional month was required to change the AIDE firmware from a polling method to an interrupt method. I was able to spend approximately ten to fifteen hours per week writing and debugging the code.

During the process of developing the system, I found the ELM327 to be simple, well documented and provides an excellent way to translate CAN messages. The Philips BGB203 was also well documented and an easy to use Bluetooth Serial Port Profile.

I would change to using the ELM as a debugger to help develop CAN on the system instead of including it in the system. I would use an IAR development board in conjunction with the software. The Olimex boards require cobbling together various development environments which took too much time.

The following photographs show the working project. Figure 12 shows the AIDE prototype connected to the OBD-II port of the test vehicle. Figure 13 shows the main screen of the BusinessMiles app running on my LG Optimus S smart phone.



Figure 12: AIDE Prototype Connected to Test Vehicle.

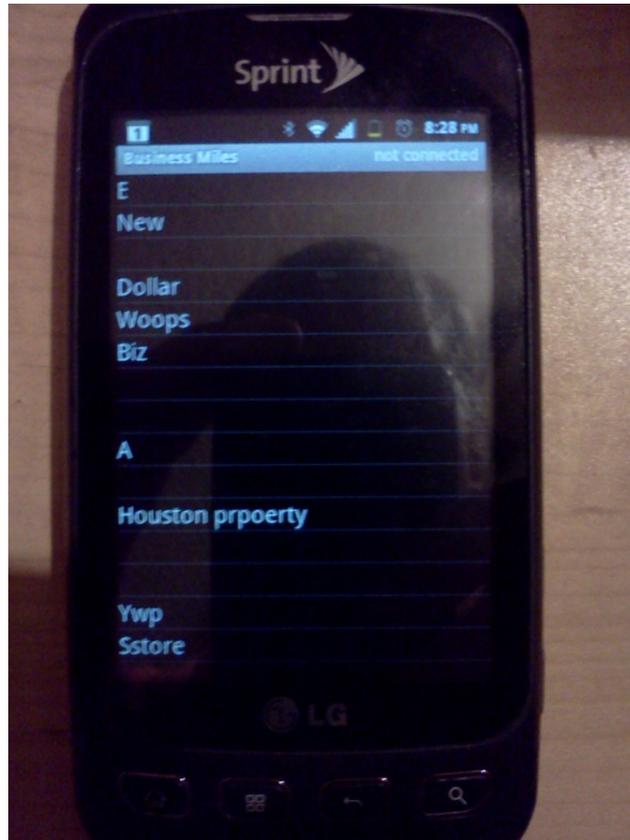


Figure 13: BusinessMiles Main Screen on LG Optimus S Smart Phone.

Chapter 6: Conclusion

The MileageAIDE system is designed for the business professional that uses a vehicle for business and personal use. It simplifies tracking and recording business mileage for tax records and employee reimbursement by automatically recording trip details and reminding the user to classify the trip as business or personal.

6.1 FUTURE WORK

The Automobile Information Device can be improved by debugging the low power exit problem to minimize battery drain while the vehicle is parked. Removing the ELM327 board and implementing a direct communication between the microcontroller and CAN network will reduce the cost of the full device and still be able to communicate with most vehicles. A Bluetooth pairing procedure will need to be added to the AIDE. Currently, the phone identifier is hardcoded into the firmware. A custom board layout will make the device less susceptible to vibrations and reduce the device size to approximately the size of the OBD-II connector, which will allow the device to be easily and discreetly placed in the vehicle.

The BusinessMiles app provides a helpful method for tracking business miles, but will need to be integrated with other features to be truly useful to the business professional. The ability to easily download the mileage data to an expense tracking software or automatically sync to cloud will be needed. Adding a map overlay of the beginning and ending GPS coordinates will be helpful for remembering the trip purpose if the user needs to add it at a later time. Integrating the Contacts and Calendar in the user's phone will give the user the option to tie a trip to a specific contact or calendar event. The phone could also select the trip purpose automatically based on information from the user's Contacts, Calendar or the GPS coordinates.

Bibliography

- [1] Internal Revenue Service Publication 463 *Travel, Entertainment, Gift, and Car Expenses*, 2011
- [2] TripLog – GPS Mileage Tracker. Available online at <http://www.bizlogapp.com>
- [3] Mileage Logger. Available online at <http://www.mileagelogger.com/Mileage-Logger-OBD-P27.aspx>
- [4] Jorge Zaldivar, Carlos T. Calafate, Juan Carlos Cano, Pietro Manzoni. Providing Accident Detection in Vehicular Networks Through OBD-II Devices and Android-based Smartphones, *IEEE 36th Conference on Local Computer Networks (LCN)*, 2011
- [5] Heng-Fa Teng, Meng-Jil Wang, Cheng-Min Lin. An Implementation of Android-Based Mobile Virtual Instrument for Telematics Applications. *Second International Conference on Innovations in Bio-inspired Computing and Applications (IBICA)*, 2011
- [6] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental Security Analysis of a Modern Automobile. *IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2010.
- [7] PinoutsGuide.com, OBD II (VAG) vehicle diagnostic interface pin out. Available online at http://pinoutsguide.com/CarElectronics/car_obd2_pinout.shtml
- [8] Robert Bosch, *CAN Specification* Version 2.0, 1991
- [9] Olimex TMS470-P256 product information. Available online at <https://www.olimex.com/Products/ARM/TI/TMS470-P256/>
- [10] Olimex MOD-BT User manual. Available online at <https://www.olimex.com/Products/Modules/RF/MOD-BT/resources/MOD-BT.pdf>
- [11] ELM Electronics' ELM327 OBD to RS-232 Interpreter User Manual (v1.4b). Available online at <http://www.elmelectronics.com/DSheets/ELM327DS.pdf>
- [12] IAR Embedded Workbench. Information and downloads available at <http://www.iar.com/Products/IAR-Embedded-Workbench/>
- [13] Google Inc. (2012, Nov.) Android Developers. Available online at <http://developer.android.com/guide/components/fundamentals.html>
- [14] Google Inc. (2012, Nov.) Android Developers Reference. Available online at <http://developer.android.com/reference/android/app/ListActivity.html>

- [15] Google Inc. (2012, Nov.) Android Developers API Guides. Dialogs. Available online at <http://developer.android.com/guide/topics/ui/dialogs.html>
- [16] Google Inc. (2012, Nov.) Android Developers API Guides. Notifications. Available online at <http://developer.android.com/guide/topics/ui/notifiers/notifications.html>
- [17] Google Inc. (2012, Nov.) Android Developers API Guides. Intents and Intent Filters Available online at <http://developer.android.com/guide/components/intents-filters.html>
- [18] D. Wheeler. SLOCCount. Available online at <http://www.dwheeler.com/sloccount/>