Software Documentation

For

Access to Vehicle CAN Bus

Project Members: Kyle Booker, Braeden Burrows

Project Supervisor: Kevin O'Neil

Project Client: Dr. Mahnhoon Lee

Last Modified: April 15th, 2018

Table of Contents

1 Introduction

1.1 Overview

The objective of this project is to create a system that retrieves both real-time and diagnostic information from a modern vehicle and display it on an Android device. This process will require knowledge of vehicle CAN bus protocols, as well as Arduino, Android and Bluetooth development. The final product will be an Android application and Arduino program that work simultaneously to retrieve vehicle data for the client.

1.2 Scope

Most modern vehicles use an OBD-II protocol. In particular, there are five common implementations with the most common being ISO 15765-4 CAN. This implementation allows for more functionality and monitoring of real-time vehicle data. It has also become standard in all vehicles after the year 2008. For this reason, the scope of our application will be restricted to vehicles after this date.

1.3 Goals and Objectives

- Read vehicle data in real-time using the CAN bus and Arduino hardware.
- Develop an Android application to display vehicle data.
- Present a visualization of vehicle data.
- Allow for diagnostic vehicle errors to be displayed on the Android device.
- Provide a complete project that allows for future evolution and modification.

2 Requirements

2.1 Purpose

This project utilizes the CAN bus protocol in recent vehicles with OBD-II. The goals of the project are to develop an Arduino program that reads and writes to the CAN bus as well as an Android application that can send commands to the Arduino board and present incoming vehicle data via Bluetooth.

2.2 General System Requirements

Functional Requirements

- System should display engine RPM.
- System should display throttle valve position.

- System should display vehicle speed.
- System should display battery voltage.
- System should display the malfunction indicator light (MIL).
- Display of other relevant vehicle diagnostics.
- Ability to alter the refresh rate of vehicle diagnostics on Android device.
- Include error messages on Android device for connection failure, and lost connection.

Nonfunctional Requirements

- System must be accessible by Android devices.
- Display all information on Android device clearly and logically.
- Cost of any libraries must be free.
- Intuitive to navigate between pages.
- Fast response times in fractions of a second to display real-time vehicle information.

Modularity

- Functions should be separated and as modular as possible.
- Ability to remove functions or add functionality without drastic changes to the application.

Readability

- Functions should be separated.
- Concise and clear code.
- Well documented methods.

Performance Requirements

- Refresh rate of vehicle information between 0.1 seconds and 1 second.
- Navigation buttons are responsive at all times.

Design Requirements

- Home screen with navigation to Dashboard and Diagnostic page, displays connection (optional additional settings).
- Dashboard page displaying vehicle speed, RPM, throttle valve position, fuel percentage, and battery voltage.
- Diagnostic page displaying malfunction indicator light and error codes stored on vehicle's internal CPU (if applicable).

System Interfaces

- Android program must interface with the Arduino board via Bluetooth module.
- Arduino program will interface with the vehicle via the CAN bus shield and OBD-II cable.

2.3 Initial System Architecture

Hardware Requirements

- An Arduino board equipped with a Bluetooth module, the CAN bus shield, as well as an OBD-II converter cable and an Android device capable of Bluetooth communication.

Software Requirements

- The library used for Arduino will be the Mechanic library from the google code archive (See reference for link). Includes functions for vehicle speed, engine RPM, throttle, battery voltage and all other features of the CAN bus Shield.
- Refresh rate option on Android device for Arduino data request from vehicle.
- Display after option on Android device to display vehicle information after receiving 2-20 values.

Programming Languages

- The Android application will be written in Java.
- The Arduino application will be written in C with possibly libraries in C++.

3 Analysis

3.1 Proposed System

Access to vehicle CAN bus is a real-time system that has a connection with the vehicle via a physical OBD-II cable. That data is then transmitted by Bluetooth to an Android device so it can display dashboard and diagnostic information on the vehicle. The structure of development will follow an agile iterative approach, with continuous refinement of existing tasks as well as introduction of new requirements.

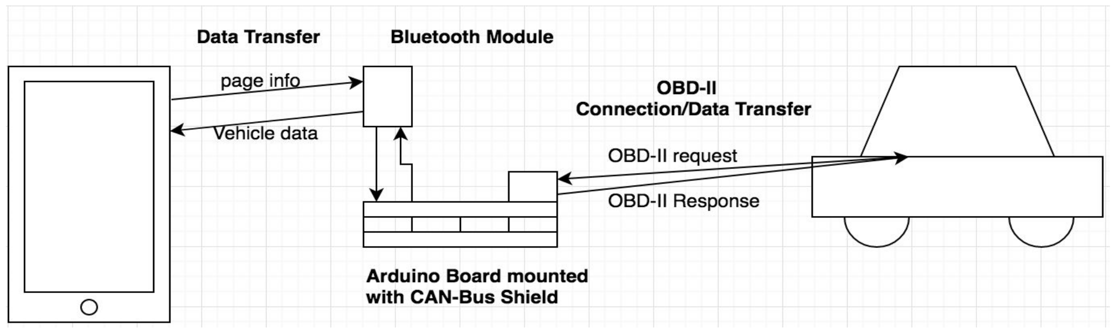
System Hardware

- Arduino.
- Android Device.
- HC-06 Bluetooth module.
- CAN bus shield.
- OBD-II converter cable.

3.2 System Architecture

Conceptual Model of the System

The vehicle will be connected to the Arduino board via OBD-II converter cable and a CAN bus shield. This will allow data to be transferred from the vehicle to the Arduino device. This transfer will be accomplished using a CAN bus library to create a CAN bus message to request specific data from the vehicle. This library will also be used to translate the response of this request into human readable form. To determine what data is to be requested, the Arduino will recieve page information via Bluetooth from an Android application. The page information will allow the Arduino to have access to which page of the application the Android device is currently on and request information from the vehicle accordingly. When the Android device receives the data from the Arduino it will display it to the user. The user will use the menu options in the application to navigate the different activities of the application.



3.3 Implementation Plan

The implementation plan, in order, will be:

- Test Bluetooth connection with Android device from Arduino.
- Test Bluetooth connection with Arduino from Android.
- Test CAN bus connection from vehicle to Arduino with required libraries.
- Implement Android vehicle application design.
- Format and parse data from vehicle to Arduino.
- Format and parse data from Arduino to Android.
- Displaying of vehicle information on Android device.

As this project is following an iterative design, initiation or completion of these tasks may overlap one another.

CAN Bus Standard

- Allows microcontrollers to communicate with each other.
- Uses a broadcast message-based protocol for data consistency across the network.

We use this message-based protocol when we wish to query the engine speed, RPM, etc. by the specific parameter identification (PID) codes.

3.4 Libraries

Arduino: https://code.google.com/archive/p/mechanic/
Library for functionality to communicating with the vehicle CAN bus. Using the MCP2515 protocol, the user is able to request vehicle information such as vehicle speed. We chose this library because it provided methods that allowed us to obtain all the information that our client was looking for from the vehicle. The code in this library is readable and easier to understand than most of the other libraries that we came across. This was also the most complete library that we came across.

Arduino: https://www.arduino.cc/en/Reference/SoftwareSerial
Library to create a software serial port on the Arduino. Needed for use of the Bluetooth module for data communication between Android and Arduino due to CAN bus shield also requiring serial communication.

Android: https://github.com/anastr/SpeedView
Library for Android speedometer. Used to display user speed on the dashboard page of the application. We chose this library because it had gauges that were both customizable and simplistic. The gauge that we chose matched our design style and easy to manipulate for our application.

3.5 Sending Data

Arduino to Android
The data is sent from the Arduino to the Android device using Bluetooth protocol via HC-06 Bluetooth module.

- Data is sent as a string of values separated by commas.
- The last characters of all strings sent are the termination sequence ‘\r\n.’
- The termination sequence identifies that no more characters are in transmission.
- The string “ready,” is sent to the Android to let it know that a connection between the Arduino and car has be found.
- The string “nocom,” is sent to the Android to let it know that a connection cannot be found between the car and the Arduino.

Android to Arduino
The data is sent from the Android device to the Arduino using Bluetooth protocol via the Android devices built in Bluetooth module. Data is sent as a string of two characters:

- The first character represents the current page of the application.
    - m — is the main page of the application.
    - c — is the DTC (Diagnostic Trouble Codes) page of the application.
    - d — is the dashboard page of the application.

- The second character represents the rate at which data is to be requested from the vehicle (only applicable for the dashboard page).

Arduino to Vehicle

The data is sent to the vehicle from the Arduino via CAN bus shield and OBD-II cable. Three functions from the mechanic.h library are used to send data to the vehicle.

- **getMultiframePid** function is used to get the multiple values from a current mode and process ID. This function takes the following arguments:
    - The mode of operation of the onboard diagnostics (for DTC this value is set to 0x03).
    - A process ID (for DTC this value is set to -1).
    - A character array to store the returned values.
    - A integer variable to store the number of bytes returned.
- **isPidSupported** function is used to check whether a process ID is supported by the vehicle. This function takes the following arguments:
    - The process ID to check.
    - A boolean variable to hold the the result.
- **getPidAsFloat** function is used to get a floating point value for the requested process ID. This function takes the following arguments:
    - The processes ID.
    - The minimum value of the process ID.
    - The maximum value of the process ID.
    - A float variable to hold the result.

Vehicle to Arduino

The data is sent to the Arduino from the vehicle via OBD-II cable and CAN bus shield. The data that is sent to the Arduino is a response to a request for a specific process ID and is sent using the CAN bus protocol.

Error Checking

- All messages that are received by the Android device on the dashboard page are tested for length to ensure that the correct amount of data has been received from the Arduino. The values are also then converted to integer values if possible and if this is not possible a value of 0 is returned.
- All messages received on by the Android device on the diagnostics page are tested to ensure that the first value received is an integer value determining the number of DTC’s.
- All messages that are received by the Android device on the main menu page are tested for specific values (“read” and “nocom”).

3.5 Functional Requirements

Display
The main page of our application is a menu that is made up of image buttons with text descriptions underneath. We chose images that represent the pages they linked to, this allows the user to quickly and easily identify what each button is for. The refresh rate is a drop down menu to conserve on screen space for the application. We partitioned the data received from the vehicle into the diagnostic page and the dashboard page. Because vehicle errors are infrequent, so should polling the vehicle for trouble codes.

- Dashboard Page: Continuously updates the user on vehicle information in real-time.
- Diagnostics Page: Static page that polls the vehicle once for any vehicle errors to display to the user.

User Input
The user will navigate the application by pressing buttons from the main menu. The buttons will provide feedback to the user that indicates what they have selected. On the dashboard page, the user will be able to choose from a set array of integer values to control both the rate at which data is read from the vehicle and the rate this data is displayed on the Android device. This input will be done by selecting the value from a spinner object. We chose values from 100 to 1000 milliseconds as the possible rates to read data from the vehicle and 2 to 20 as the values for the number of values collected before they are displayed to the Android device. We chose these values to allow the application to have a real-time feel while still allowing enough time for the data to be collected correctly, accurately and displayed to the user with enough time to be read before another value is displayed.

3.6 Nonfunctional Requirements

Reliability
To ensure reliability of the system, a comprehensive testing phase will be required.
- Each individual component will be tested for accuracy.
- System as a whole will be tested on multiple vehicles under the same scenarios.
- System may be tested under moderate stress (i.e. large amount of data transfer, long run time).

Readability
The client requires to be able to modify and adapt the system in the future so accommodations must be made, such as:
- Functions are clearly defined with input/output.
- Code should be uniform in style.
- Good programming style standards will be adhered to.
- Excessive functionality will be avoided or refactored.

4 Design

4.1 Android Application

Home Page
The home page is general purpose, allowing the user to easily connect with the Arduino via Bluetooth and navigate towards the possible options of the application. Each option will be clearly displayed with a title as well as an appropriate picture to convey its message.

- Title at the top of the application, titled: “Vehicle Information”.
- 4 icons, in a 2 x 2 grid layout.
- Connect icon will display connection to Arduino, and light up when connection has been established.
- Diagnostic button will bring user to the diagnostic page.
- Dashboard button will bring user to the dashboard page.
- Exit button will exit the application.

Diagnostic (Car Error) Page
The purpose of the diagnostic page is to display any vehicle errors. The indicator light will indicate if the car has an error, the number of errors will be displayed and any 5 character vehicle error code will be displayed beneath.

- Title at the top of the application, “Diagnostics”.
- Indication light with name will be displayed in a list view.
- Indication of car error will result in illuminated icons.
- Car errors will be displayed in list format below indication light.
- Will update only once when page is opened..
- Button to return to the main menu.
- Possible lights: Check engine light, battery light, temperature warning light, brake light, oil pressure light, etc.

Dashboard (Real-time) Page
The purpose of the Dashboard page is to provide real-time feedback on the vehicle’s dashboard information. Examples include speed, RPM, and throttle position. The speed will come equipped with an animated speedometer to provide both a numerical and visual representation of the speed. Other gauges may not be animated, but will come equipped with a changing numerical value and label for easy identification. The only user input will be in the form of spinners, which will allow for changing of the refresh rate of the data sent to the Android application and the number of values to be collected before they are displayed on the screen. The refresh rate will be between 0.1 seconds and 1 second with 5 possible increments. The number of values until

display will be between 2 and 20 with 5 possible increments. The spinners will appear at the top of the page.

- Title at the top of the application, “Dashboard”.
- Spinner input to adjust refresh rate of vehicle information.
- Spinner input to adjust the number of values until one is displayed.
- Speedometer with animated gauge and digital display.
- Fuel Gauge, displaying percentage.
- Engine throttle position.
- Tachometer, displaying RPM.

4.2 Arduino Application

The Arduino program will not be visible to the user in terms of operation, however memory and efficiency must be considered for optimal running of the application. A continuous loop will monitor for Bluetooth commands from the application, in the form of two variables:

- Page variable, indicating what page the device is on.
- Refresh rate variable, indicating the refresh time on the vehicle information.

The Arduino will then receive information and OBD-II diagnostics and send them to the Android device via Bluetooth to be parsed and displayed. The Arduino program should be short and concise for space and efficiency.

4.3 Images and Screenshots

- Buttons should be able to light up upon press or activation.

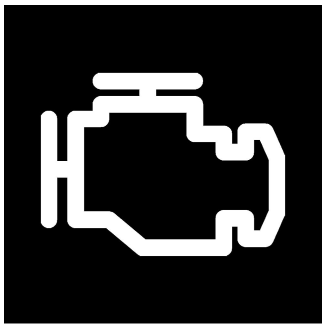


Figure 1. Unlit engine button example.

:

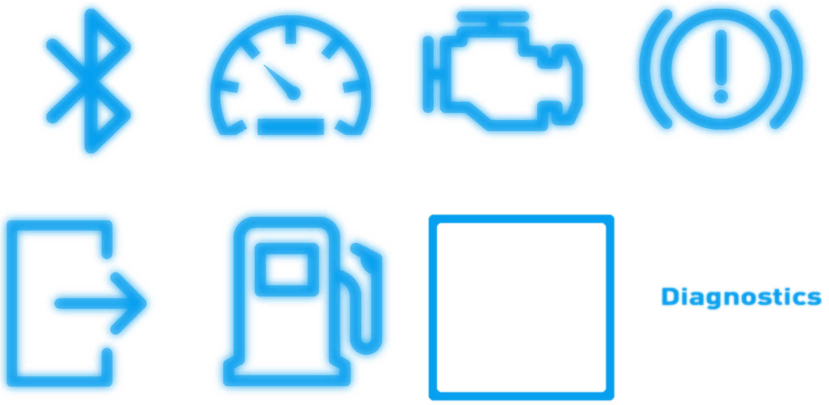
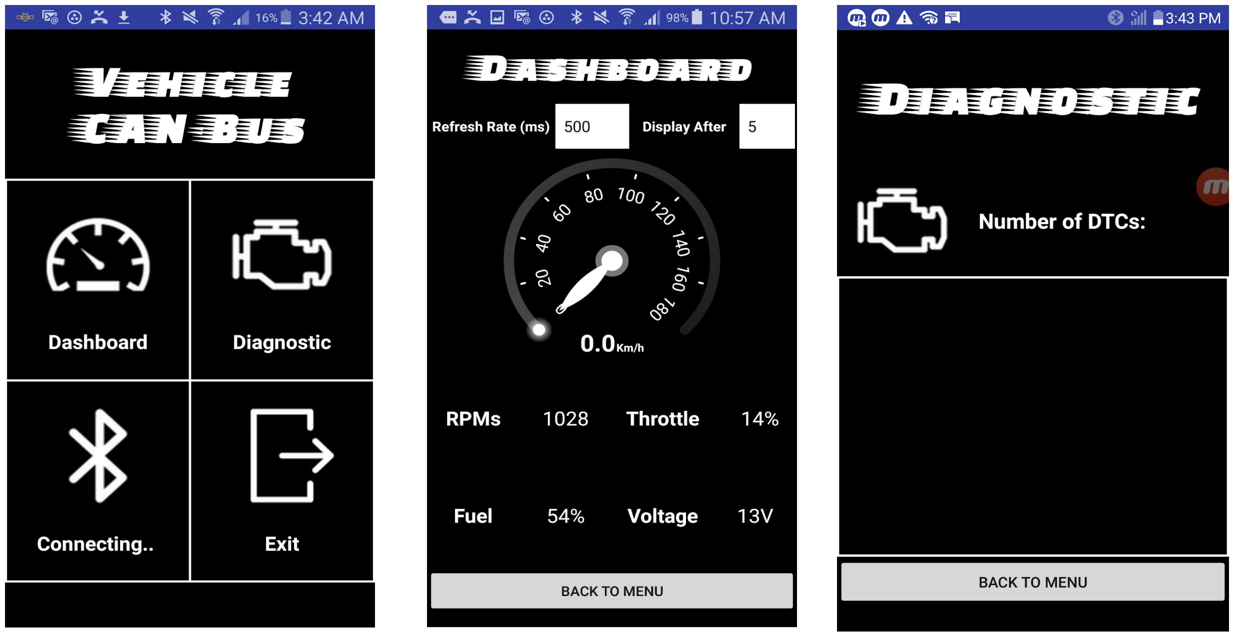


Figure 2. Blue lit button examples.



Refresh Rate (ms) 500 Display After 5

RPMs	1028	Throttle	14%
Fuel	54%	Voltage	13V

Number of DTCs:

Figure 3. Screenshot of the various application pages.

5 Testing

5.1 Testing Cases

Test Cases: Home Page
All Test Cases will be documented as pass/fail.

- Application opens on icon press and brings user to the Home page.
- Bluetooth automatically connects to Arduino on launch of application.
- Home page is properly formatted.
- Forbids user from accessing Diagnostic or Dashboard Page until Bluetooth connection is made.
- All icons on home page light up when depressed or when Bluetooth connection is established.
- Exit button closes the application.
- Application is responsive under button presses.
- If Bluetooth fails to connect then an error message pops up informing the user.
- If Bluetooth loses connection then an error message pops up informing the user and brings them to the Home page (from any other page).

Test Cases: Diagnostic Page

- Diagnostic Button opens the Diagnostic page.
- Page is formatted correctly on all Android device screens.
- Displays the correct error count of vehicle errors.
- Engine light on application illuminates if the number of vehicle errors is greater than zero.
- Properly formats and displays any vehicle error codes on phone application with their unique identification.
- List of vehicle error codes becomes scrollable if large amount of data is sent from the vehicle.
- Phone back button returns user to the Home page.

Test Cases: Dashboard Page

- Dashboard Button opens the Dashboard page.
- Dashboard page is displayed properly on all Android devices.
- Incoming data is accurate and consistent.
- Speedometer displays vehicle speed as well as a working visual speedometer.
- RPM’s is displayed.
- Fuel Percentage is displayed.
- Throttle Position is displayed.

- User can select data rate speed from a dropdown menu (0.100 - 1 second).
- Data rate speed is accurate +/- 0.150 seconds.
- User can select amount of data smoothing by changing the number of requests required before averaging (higher values will delay results).

Test Cases: Stress Testing

- Application can run successfully for extended length of time (30 minutes).
- Application runs successfully under high baud rate and data transfer rate.

5.2 Vehicles

An important aspect of our testing is factoring in different vehicle makes and models, which can be highly variable in their protocols. Listed below is the year, make, model and any other relevant details of the vehicle. All of the above test cases will be tested on each individual car to make sure accuracy, efficiency, and consistency of the application is maintained.

Table 1

Vehicle Testing

MAKE	MODEL	YEAR	PASS/FAIL
Hyundai	Sante Fe	2016	PASS
Mazda	3	2014	PASS
Toyota	Corolla	2014	PASS
Hyundai	Sante Fe	2014	PASS
Kia	Optima	2013	PASS
Ford	Focus	2010	PASS
Toyota	Tacoma	2009	PASS
Mazda	RX-8	2008	PASS
Mazda	3	2007	FAIL
Honda	Pilot	2006	FAIL
Nissan	Maxima	1995	FAIL

5.3 Android Devices

This application was developed with a phone in mind and testing will be done on several Android devices. This is to ensure formatting and design is consistent across different devices, as well as how the application handles scaling for screen size.

Phone Brand/Model/OS

Brand/Model	Operating System	Pass/Fail
LG G3	Marshmallow (6.0)	Pass
Samsung S4	Lollipop (5.1)	Pass
Samsung S3	Jelly Bean (4.4.2)	Pass
Samsung A5	Nougat (7.0)	Pass

6 References

Potential CAN Bus Libraries

Arduino CAN Bus ECU Reader. (n.d.). In *Github.* Retrieved from https://github.com/skpang/Arduino_CAN-Bus_ECU_reader/blob/master

Arduino OBD-II Adapter Library and Sketches. (n.d.). In *Github.* Retrieved from https://github.com/stanleyhuangyc/ArduinoOBD

Mechanic. (n.d.). In *Google Code Archive.* Retrieved from https://code.google.com/archive/p/mechanic/

OBD-II PIDs. (n.d.). In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/OBD-II_PIDs

Seeed-Studio CAN Bus Library. (n.d.). In *Github.* Retrieved from https://github.com/Seeed-Studio/CAN_BUS_Shield

SparkFun CAN Bus Arduino Library. (n.d.). In *Github.* Retrieved from https://github.com/sparkfun/SparkFun_CAN-Bus_Arduino_Library

Android Bluetooth Information

Android Bluetooth Connection Example. (2016, August 4). In *Java Code Geeks.* Retrieved from https://examples.javacodegeeks.com/android/android-bluetooth-connection-example/

Bluetooth. (n.d.). In *Android Develop.* Retrieved from https://developer.android.com/guide/topics/connectivity/bluetooth.html

Understanding the Android Application Class. (2017, November 12). In *Github*. Retrieved from https://github.com/codepath/android_guides/wiki/Understanding-the-Android-Application-Class

Android and Arduino Communication

Arduino Bluetooth Basic Tutorial. (2016, May 23). In *Arduino*. Retrieved from https://create.arduino.cc/projecthub/user206876468/arduino-bluetooth-basic-tutorial-d8b737

How to: Create an Android App with Android Studio to Control LED. (2015, February 21). In *Instructables.* Retrieved from http://www.instructables.com/id/Android-Bluetooth-Control-LED-Part-2/

How to Receive Arduino Sensor-Data on your Android Smartphone. (2016, April 13). In *Instructables.* Retrieved from http://www.instructables.com/id/How-to-Receive-Arduino-Sensor-Data-on-Your-Android/

Modify the HC-05 Bluetooth Module Defaults Using AT Commands. (2013, August 31). In *Instructables.* Retrieved from http://www.instructables.com/id/Modify-The-HC-05-Bluetooth-Module-Defaults-Using-A/

Android Gauge Libraries

AndroidWidgets. (n.d.). In *Github.* Retrieved from https://github.com/ntoskrnl/AndroidWidgets

SpeedView. (n.d.). In *Github.* Retrieved from https://github.com/anastr/SpeedView

Android Fonts

Digital Fonts. (n.d.). In *1001 Fonts.* Retrieved from http://www.1001fonts.com/digital-fonts.html

Other CAN Bus Information

CAN Bus Explained - A Simple Intro (2018). In *CSS Electronics.* Retrieved from https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en

CAN-BUS Shield V1.2. (n.d.). In *Seeed Studio.* Retrieved from http://wiki.seeed.cc/CAN-BUS_Shield_V1.2/

Controller Area Network (CAN) Overview. (2014, August 1). In *National Instruments.* Retrieved from http://www.ni.com/white-paper/2732/en/

Introduction to the Controller Area Network. (2016, May). In *Texas Instruments*. Retrieved from http://www.ti.com/lit/an/sloa101b/sloa101b.pdf

7 Appendix

7.1 Progress Reports

Week 1

Summary

Met with Mahnhoon Lee and discussed the overview of the project. Mahnhoon Lee advised us to do some basic research of Arduino libraries involving CAN bus and Android libraries for Bluetooth. Informed us that all hardware for the project would be provided by him.

Task Completion from Last Week

- Formed project group.
- Met with client Mahnhoon Lee.
- Preliminary research with Arduino and CAN bus Libraries.

Task for This Week

- Begin Requirements Document.
- Schedule second meeting with Mahnhoon Lee.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- The hardware for the project is not yet available.

Week 2

Summary

Met with Mahnhoon Lee and went over our first draft of requirements. Mahnhoon provided more insight into what he is looking for in terms of user interfaces. Informed us that all hardware for the project should arrive early next week and that he would contact us when it has arrived. We chose the CAN bus library (Sparkfun CAN bus Library for Arduino) that we will be using to access data from vehicles.

The project requires reading diagnostic information about the vehicle using an Arduino board attached to the vehicles OBD-II connector. That information will then be displayed on the Android device using Bluetooth. The application requires little user input, but proper management and presentation of data will be required.

Task Completion from Last Week

- Chose a CAN bus library.

- Met with client Mahnhoon to confirm and establish requirements.
- Preliminary research on Android Bluetooth communication.

Task for This Week

- Add to Requirements Document.
- Schedule third meeting with Mahnhoon Lee.
- Continue Android Bluetooth research specifically transmitting data between devices.
- Research Arduino CAN bus library and Bluetooth communication.

Additional Information

- None this week.

Known Issues/Things blocking progress

- The hardware for the project is not yet available (Should be arriving this week).

Week 3

Summary

This week we received the Arduino, CAN bus, and other peripherals. Braeden soldered the CAN bus shield pins and we now are able to send Bluetooth commands from the Arduino to the Android application. Last week we worked on experimenting and researching the Arduino to Android connection as well as creating the design documentation of our application to be implemented.

Task Completion from Last Week

- Soldered CAN bus shield.
- Researched Bluetooth and CAN bus libraries.
- Started Design document.
- Met with client.

Task for This Week

- Implementation of UI for the Android application.
- Finding or Designing graphical images for application use.
- Begin Analysis documentation.
- Meet with client.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- None this week.

Week 4

Summary

Several Documents were added to and completed. Met with Mahnhoon who added the requirement of wanting the code to be modular and readable, so changes could be made in the future. The CAN bus library and Arduino were tested to confirm connection and data transmission with the car is possible. We are currently looking into libraries that can retrieve car error codes as well as testing scenarios. A design was approved by the client.

Task Completion from Last Week

- Tested CAN bus library from Arduino to vehicle.
- Tested Bluetooth functionality.
- Finished Requirements document.
- Started Analysis document.
- Confirmed design with client.

Task for This Week

- Implementation of UI for the Android application.
- Finish Analysis document.
- Meet with client.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- Finding an Arduino library that accesses and retrieves vehicle error codes.

Week 5

Summary

Connection from Arduino to Android has been established. Data can been requested from using the Arduino and transmitted to the Car via Bluetooth. Vehicle Speed and RPM can be displayed, but more work is required to create a function to display the car error codes on the Android device. Speedometer library is functional on the Android device and can display an animation of current speed as long as data is being sent to it.

Task Completion from Last Week

- Data sent from Vehicle to Arduino to Android.
- Tested Speedometer library.
- Home Page UI.
- Researched car error code libraries.

Task for This Week

- Test vehicle error codes request on vehicle.

- UI of diagnostic and dashboard page.
- Work on Analysis Document.
- Meet with client.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- Trouble retrieving car error codes from vehicle.

Week 7

Summary

We completed our midterm review. Client was happy with progress and added minor requirements such as being able to determine if vehicle is going in reverse, which we are investigating. All libraries for the project have been selected and implemented. Error checking and transmission of data via Bluetooth is in development, as sending the data is not as concrete as we'd like. Main home page has been completed, with progress on the Diagnostics and Dashboard page.

Task Completion from Last Week

- Midterm Review.
- Implemented libraries and basic functionality.
- Compiled References documentation.
- Small changes to design of application.
- Tested program on other vehicles.
- Implemented preliminary scheme for sending data over Bluetooth.

Task for This Week

- Work on Analysis document.
- Iterate on Android application design.
- Add additional items to Dashboard page.
- Test and improve error checking of and transmission of Bluetooth data.

Additional Information

- One member will be away for 3 days next week.

Known Issues/Things Blocking Progress

- None this week.

Week 8

Summary

Met with the client and discussed the requirements. Client added a new requirement to see battery percentage of vehicle due to the fact that a previous request for current gear was not possible. Error checking and transmission of data via Bluetooth is still in development and testing. We have fixed an error encountered when receiving Bluetooth data on the Android device. Diagnostic page has been completed, with progress on the Dashboard page.

Task Completion from Last Week

- Worked on Analysis document.
- Iterate on Android application design.
- Added additional items to Dashboard page.
- Test and improve error checking of and transmission of Bluetooth data.
- Fixed Bluetooth data transmission error.
- Updated requirements based on meeting with the client.

Task for This Week

- Test and improve error checking of and transmission of Bluetooth data.
- Implement updated requirements.
- Test on different vehicle makes and models.
- Continue work on Analysis document.

Additional Information

- One member will be away for 2 days.

Known Issues/Things Blocking Progress

- None this week.

Week 9

Summary

Work was done on the Analysis document and the Testing document has been started. Error handling has been implemented to deal with issues where the user loses Bluetooth connection to the Arduino (By going out of range or Arduino failure) or no Bluetooth connection is found. Application has been tested on several vehicles with good results, but unfortunately battery life has not been possible to retrieve from the vehicle (It appears as this only applies to hybrid cars). There a few things to be implemented and worked on such as how to handle data smoothing. Future tasks involve polishing all other documents before compilation, as well as planning and discussing project presentation options.

Task Completion from Last Week

- Started Testing Document.
- Iterate on Android application design.

- Created error handling for losing/no Bluetooth connection.
- Tested on several vehicles.
- Found issues with checking battery life on vehicles.
- Added averaging functionality.

Task for This Week

- Begin formal testing of application.
- Create Application Icon Design.
- Retrieve Engine Coolant or Oil Vehicle temperature.
- Continue work on Testing document.
- Improve accuracy on data transfer rate.
- Meet with Client.

Additional Information

- None this week.

Known issues / things blocking progress

- None this week.

Week 10

Summary

Testing document is currently in process. Work needs to be done compiling data to be included for final document, such as response time of CAN bus, efficiency, reliability message requests, and speed of application overall. Compilation of final documentation will begin, appending and editing past documents. User functionality of displaying data was clarified. The computer science presentation fair was also discussed.

Task Completion from Last Week

- Started Testing of Application.
- Included Vehicle Voltage.
- Added features to test the response time of the CAN bus.
- Began icon design creation.
- Met with Client.

Task for This Week

- Generate data (such as response time, messages dropped, accuracy, etc.) for report.
- Compile and begin creation of final project document.
- Meet with Client.
- Finish and implement application icon.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- Team member will be away for 5 days.

Week 11

Summary

Because of how many components in this system, a large portion of time will be devoted towards testing of the application. All features requested by the Client have been implemented. Testing on multiple vehicles, as well as data collection is of interest for presentation to the Client. Final documentation is underway, as well as searching for possible errors/issues in code and code documentation.

Task Completion from Last Week

- Generate data (such as response time, messages dropped, accuracy, etc.) for report.
- Creation of Final Document.
- Created application Icon.
- Tested application on two additional vehicles.

Task for This Week

- Search for bugs/continue testing.
- Comment and refactor code.
- Polish final documentation.
- Meet with Client.

Additional Information

- None this week.

Known Issues/Things Blocking Progress

- None this week.

Week 12

Summary

Application is the final stages of testing, as well as a final document being created and ready to submit. We have emailed Mahnhoon Lee on enrollment in the CS Showcase. A decision on when to have the final project presentation will take place this week. Application was tested on more vehicles and changes were updated to the final document.

Task Completion from Last Week

- Presented project to class.
- Implemented application icon.
- Tested application.
- Made changes to final document.

- Emailed for enrollment in CS Showcase.

Task for This Week
- Decide and confirm final project presentation date.
- Create CS Showcase presentation.
- Meet with client.

Additional Information
- None this week.

Known Issues/Things Blocking Progress
- None this week.

Week 12
Summary
We will be presenting in the CS showcase on Thursday April 12th. This week we prepared slides and notes for our talk, as well as filmed a demonstration video for the presentation. We are awaiting confirmation our final project presentation date while we make final changes to our document and application. The application was tested successfully on two additional vehicles this week.

Task Completion from Last Week
- Prepared presentation for CS showcase. ;
- Tested application on additional vehicles.
- Prepared video demonstration for CS showcase.

Task for This Week
- Decide and confirm final project presentation date.
- Present in CS Showcase.
- Edit final document.
- Meet with client.

Additional Information
- None this week.

Known issues / things blocking progress
- None this week.

7.2 Mid-Review (February 16th, 2018)

Introduction
Access to vehicle CAN bus is an application that provides a communication from a vehicle to an Android device. This project requires research the CAN bus protocol, onboard diagnostics, serial Bluetooth from the Arduino to Android, as well as Android and Arduino application programming. Documentation of the Android and Arduino application is also required.

Main Requirements
This list outlines the core functional and nonfunctional requirements of the system. They are not exhaustive, but is rather a high-level outline of how the application should operate.

- Main home page should allow access to both a dashboard and a diagnostic page.
- Connection to Bluetooth should be intuitive and simple.
- Diagnostics Page: Application should be able to request error trouble codes and display them on Android device.
- Dashboard Page: Application should be able to request and display SPEED, RPM, and THROTTLE POSITION of the vehicle on the Android device in real-time.
- Each item should be clearly labelled and formatted.
- Application should contain an animated working speedometer for vehicle speed.
- Delay for updating vehicle information should be a variable user input.
- Code should be Readable and Modular.

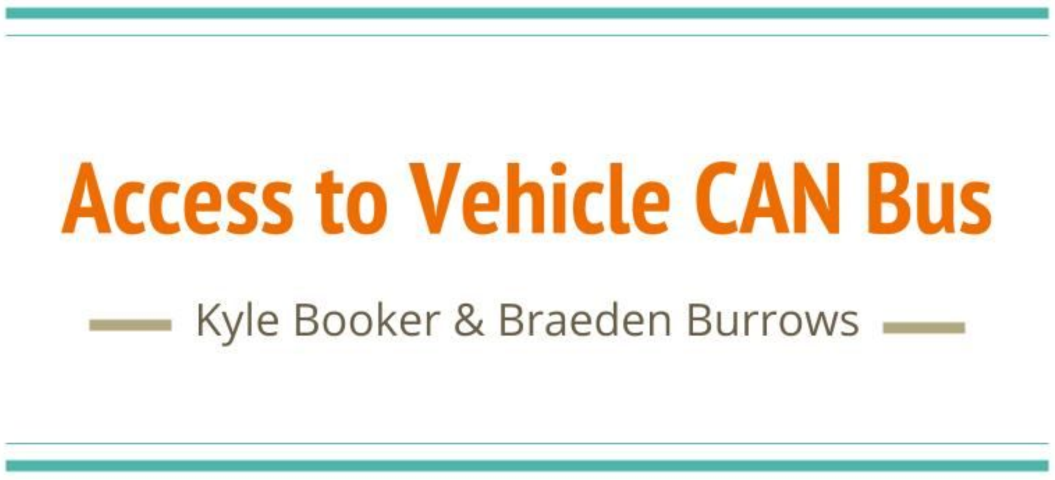
Progress (Completed Tasks)
Majority of the documentation and research has been completed. All libraries have been selected for implementation. Individual components and interfaces have been confirmed functional, as well as a very general connection between all interfaces.

- OBDII library has been selected and implemented. We chose this library due to its functionality for requesting any PID, as well as retrieving the vehicle diagnostic error codes.
  https://code.google.com/archive/p/mechanic/
- Speedometer library has been chosen for a dynamic view of the vehicle speed:
  https://github.com/anastr/SpeedView
- Bluetooth connection from Arduino to Android has been established, and data transmission is possible, as well as vice versa.
- Arduino can request error codes or onboard diagnostics from the vehicle.
- Android application can display vehicle SPEED, RPM, FUEL PERCENTAGE and ERROR CODES from vehicle.
- Design and images have been created.
- Home page of Android application has been implemented.

Upcoming Tasks
As most of the individual pieces and interfaces have been established, what remains is an iterative process of refinement and connecting components. Analysis, Testing and Code documentation must also be completed.

- Create Dashboard and Diagnostic Page for Android application.
- Parse and format data from vehicle onto correct Android pages (separate vehicle error codes from onboard diagnostics).
- Implement speedometer library to function with vehicle speed data.
- Refactor code to become more modular/readable.
- Design icon for Android application.
- Implement onboard diagnostic update delay as variable user input for Android device.
- Test application for functionality/errors.

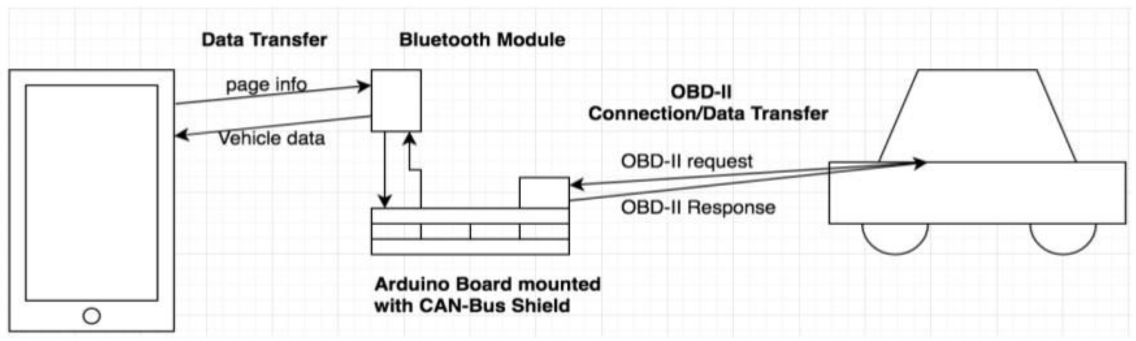
7.3 Computer Science Showcase



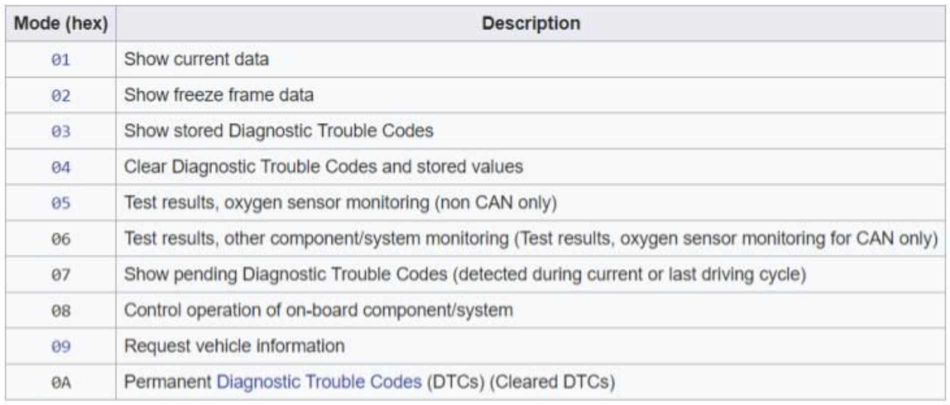
Access to Vehicle CAN Bus

Kyle Booker & Braeden Burrows

System Architecture



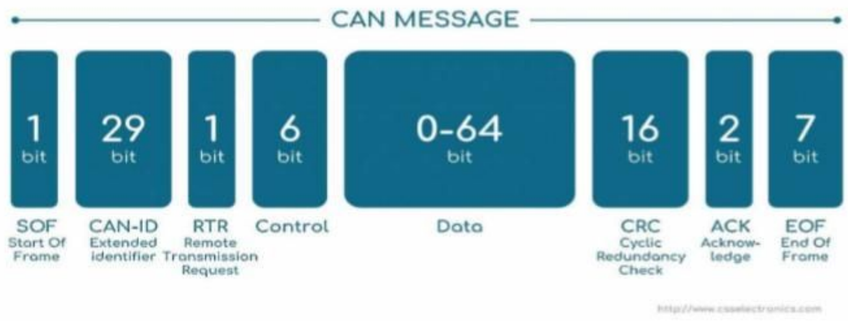
OBD-II Modes

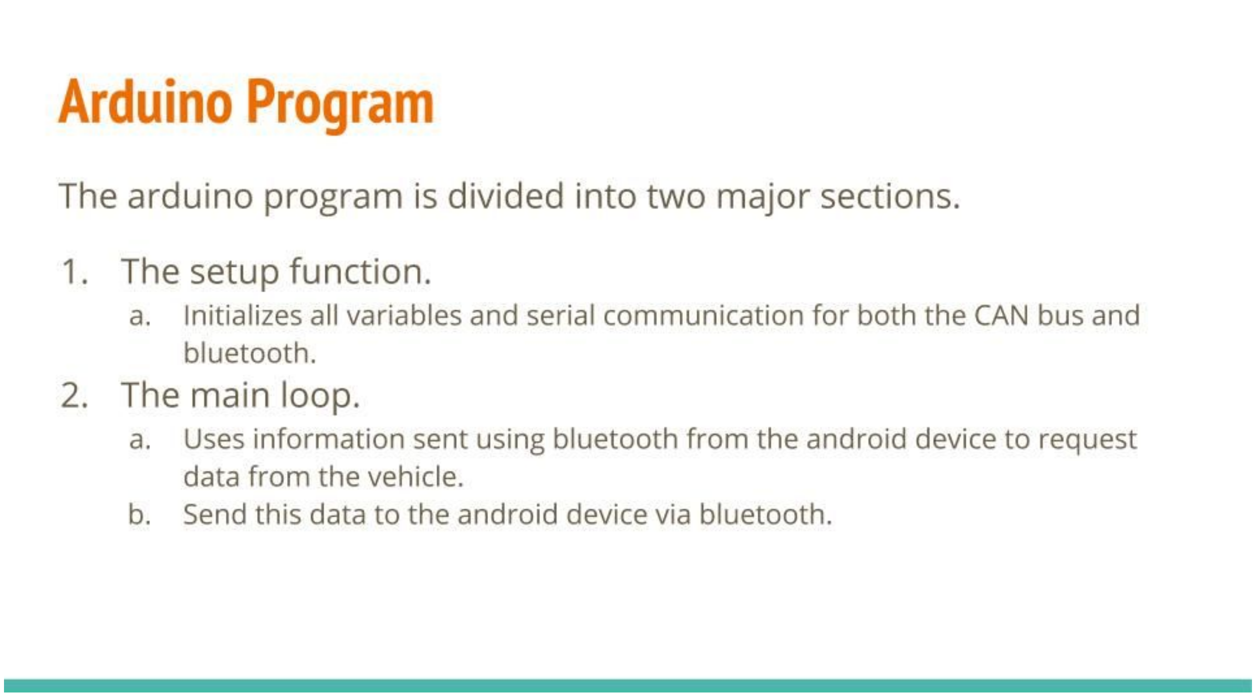


Mode (hex)	Description
01	Show current data
02	Show freeze frame data
03	Show stored Diagnostic Trouble Codes
04	Clear Diagnostic Trouble Codes and stored values
05	Test results, oxygen sensor monitoring (non CAN only)
06	Test results, other component/system monitoring (Test results, oxygen sensor monitoring for CAN only)
07	Show pending Diagnostic Trouble Codes (detected during current or last driving cycle)
08	Control operation of on-board component/system
09	Request vehicle information
0A	Permanent Diagnostic Trouble Codes (DTCs) (Cleared DTCs)

CAN Bus

- CAN is a serial bus protocol that allows microcontrollers to communicate with each other without complex and dedicated wiring.
- The data portion of the CAN message is made up of the message length, the OBD-II mode, the process ID and the data.

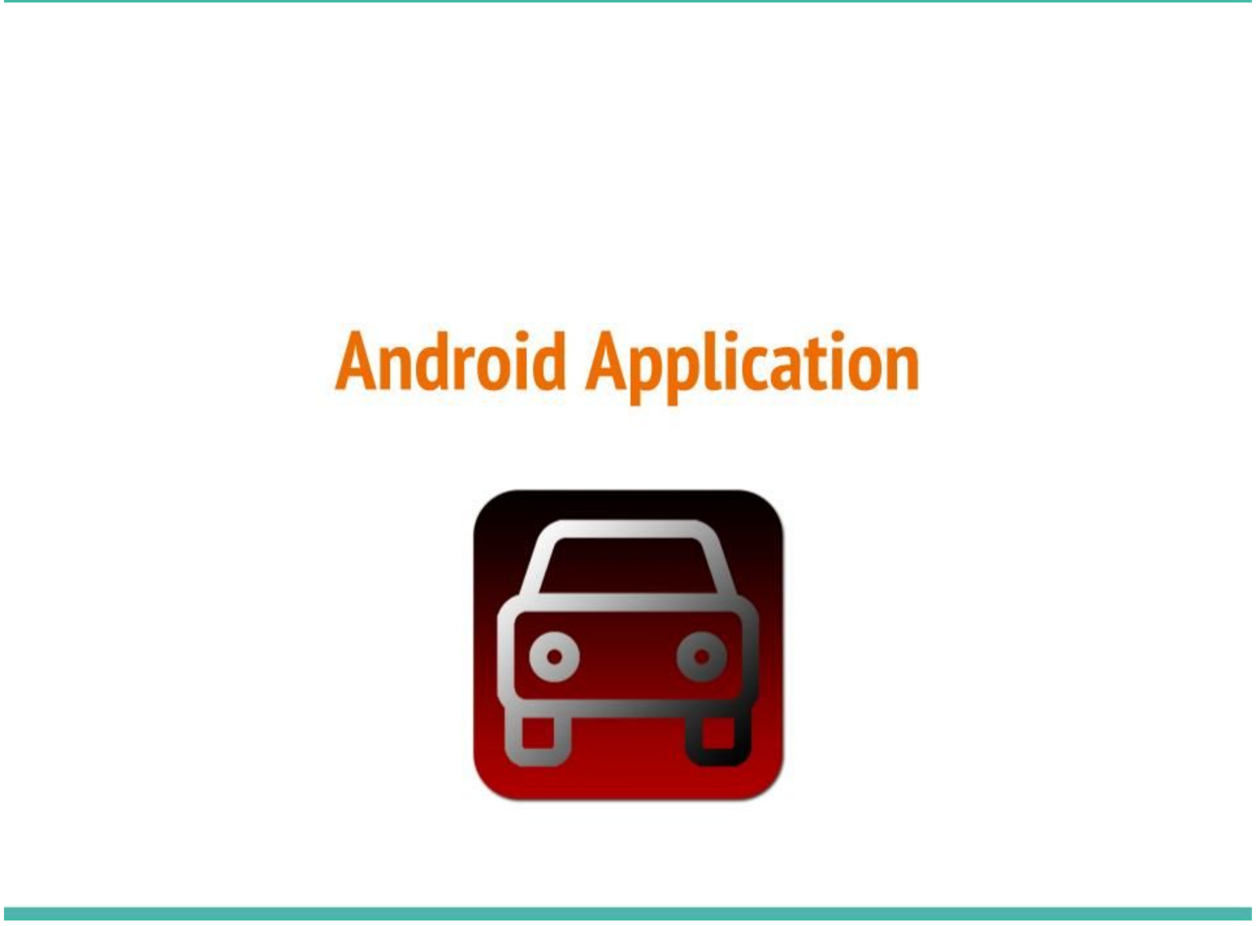




Arduino Program

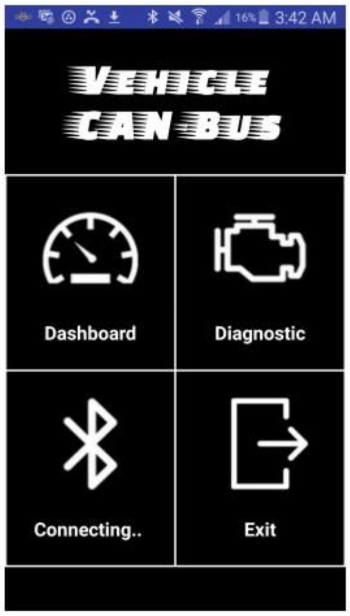
The arduino program is divided into two major sections.

1. The setup function.
   a. Initializes all variables and serial communication for both the CAN bus and bluetooth.
2. The main loop.
   a. Uses information sent using bluetooth from the android device to request data from the vehicle.
   b. Send this data to the android device via bluetooth.

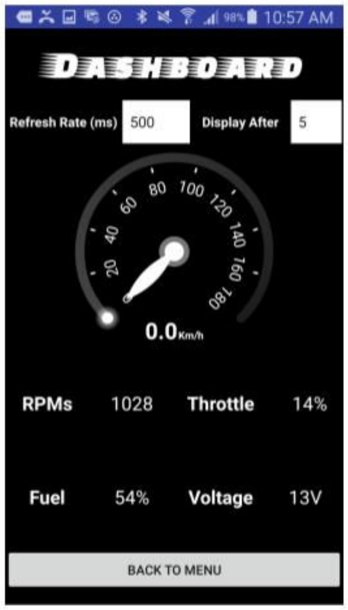


Android Application

Home Page



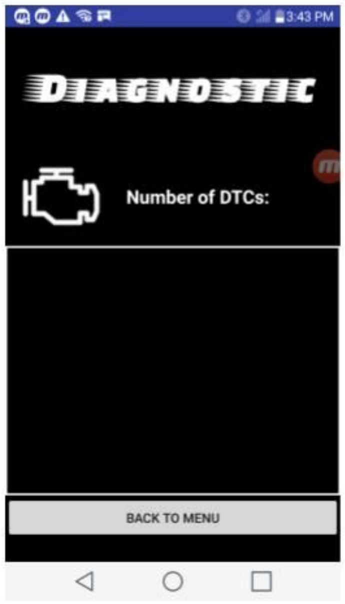
Dashboard Page



Refresh Rate (ms) 500 Display After 5

RPMs	1028	Throttle	14%
Fuel	54%	Voltage	13V

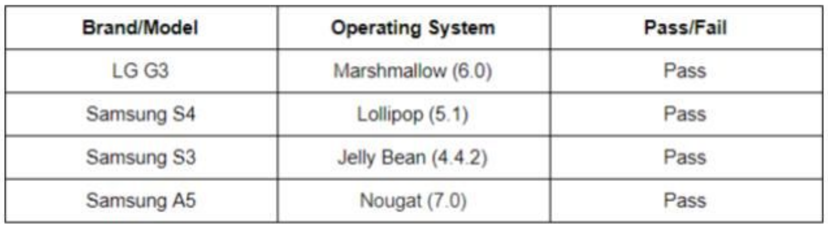
Diagnostic Page



Vehicle Testing

MAKE	MODEL	YEAR	PASS/FAIL
Hyundai	Sante Fe	2016	PASS
Mazda	3	2014	PASS
Toyota	Corolla	2014	PASS
Hyundai	Sante Fe	2014	PASS
Kia	Optima	2013	PASS
Ford	Focus	2010	PASS
Toyota	Tacoma	2009	PASS
Mazda	RX-8	2008	PASS
Mazda	3	2007	FAIL
Honda	Pilot	2006	FAIL
Nissan	Maxima	1995	FAIL

OS/Phone Brand Testing



Brand/Model	Operating System	Pass/Fail
LG G3	Marshmallow (6.0)	Pass
Samsung S4	Lollipop (5.1)	Pass
Samsung S3	Jelly Bean (4.4.2)	Pass
Samsung A5	Nougat (7.0)	Pass