Application Notes

ECE 492 Capstone Project

Launch Control

2019

Contents

[Overview of Launch Control 2](#_Toc3306566)

[Overview of Hardware 2](#_Toc3306567)

[PCB 2](#_Toc3306568)

[Teensy 3.6 Microcontroller 3](#_Toc3306570)

[Freematics OBD-II UART Adapter 4](#_Toc3306571)

[HC-05 Bluetooth module 4](#_Toc3306572)

[Getting started with the Teensy 3.6 4](#_Toc3306573)

[Relevant links 4](#_Toc3306574)

[Getting started with the Android application 4](#_Toc3306575)

[Relevant links 5](#_Toc3306576)

[Getting started with the Web application 5](#_Toc3306577)

[Relevant links 5](#_Toc3306578)

[Python Scripts 5](#_Toc3306579)

[Android Application Python Script 5](#_Toc3306580)

[Relevant links 5](#_Toc3306581)

[Testing Web Server 6](#_Toc3306582)

[Relevant links 6](#_Toc3306583)

[Appendix A 6](#_Toc3306584)

# Overview of Launch Control

Launch control is an automobile data logging system that consists of the following three major subsystems:

1. Custom made PCB data collector
2. Android app that displays the collected data and uploads it to the web
3. Cloud hosted dashboard for viewing the data

The PCB is responsible for collecting values such as speed, RPM, fuel level and other performance metrics through various parts of a vehicle. This information is collected through an OBD-II interface. The collected data is then transmitted to an Android app (via Bluetooth). The app then forwards this data (through the internet) to the cloud hosted dashboard for logging and viewing.

All of our original source code is located on a GitHub organization here: <https://github.com/launch-control-app/>

Individual repositories within the organization (linked above) are linked in the sections below (as needed).

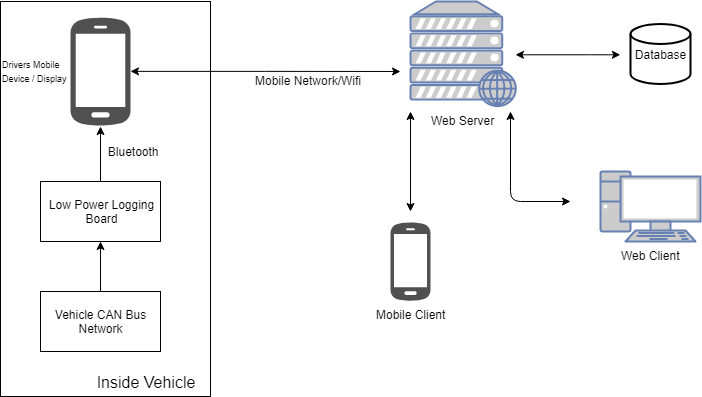


Figure 1: High level overview of the system

# Overview of Hardware

## PCB

The PCB is designed to replace the breadboard, and connects a Teensy 3.6 Microcontroller with the Freematics OBD-II UART Adapter and HC-05 Bluetooth module. More on each of the hardware is listed below.

## https://lh5.googleusercontent.com/FFw5qQ8-5HC8zGvKdLpaK2vRRvwZjkGTgXCVQPWynhwKQUKpB06wSPn9JCUyA_hgNzuzvYrviS5cvjvu8vvFsVCEsy_kbC1QJSlmvT4ZgO1f5sGog4D4-RTHghMr0yI51Lu_yhIR

Figure 2: Hardware layout of the OBD-II data collector

## Teensy 3.6 Microcontroller

The Teensy 3.6 microcontroller is the central hardware piece that polls and collects data from the OBD-II adapter and transmits it over Bluetooth. This is essentially an Arduino clone from a software point of view. It comes with a total of 5 UARTS, of which 2 are used (for the OBD-II adapter and HC-05 Bluetooth module). While the Teensy operates at 3.3V, it is still 5V compatible (Vin range of 3.6 to 6.0 V), and hence works seamlessly (without the need for a voltage step down) with the adapter’s power supply. The Teensy plugs into female headers on the PCB, however only 6 pins are actually in use - two for power supply, two for transmit/receive to the OBD-II adapter and two for transmit/receive to the Bluetooth module.

The Teensy uses a 180 MHz 32-bit ARM Cortex M4 Processor, which is found on the MK66FX1M0VMD18 chip. In addition to the UART interfaces being used, the chip also provides CAN, Ethernet, I2C, I2S, SDHC and SPI interfaces. The Teensy board also includes a SD card slot. These interfaces are as of yet unused by Launch Control, however they do allow expandability for future projects to work from.

Link to product: <https://www.digikey.com/product-detail/en/sparkfun-electronics/DEV-14058/1568-1465-ND/6569369>

## Freematics OBD-II UART Adapter

This adapter acts as a data source from the car as well as a Power supply, supplying 5V at up to 2.1A. The adapter connects to four male pin headers on the PCB. With two being power and ground, and the other two being transmit and receive. We are using a baud rate of 9600 bps to communicate with the adapter.

Link to product: <https://freematics.com/products/freematics-obd-ii-uart-adapter-mk2/>

## HC-05 Bluetooth module

This Bluetooth module is used for transmitting collected OBD-II data to a paired Android smart device. Totally, it plugs into 6 female headers on the PCB, which hold it in place. However, only 4 pins are actually used here - two for power and ground, and the other two for transmit and receive.

Link to product: <https://www.amazon.ca/J-DEAL%C2%AE-Wireless-Bluetooth-Transceiver-Arduino/dp/B01M248TJU/ref=sr_1_3?keywords=bluetooth+module+hc-05&qid=1552423528&s=gateway&sr=8-3>

# Getting started with the Teensy 3.6

For developing on the Teensy 3.6 (C++), we used Visual Studio Code as well as the Platform I/O extension for VS Code. The extension allows you to build and upload to the Teensy. It also provides serial monitor capabilities.

For interfacing with the OBD-II UART, we used an Arduino library provided by the manufacturer.

For interfacing with the Bluetooth HC-05, we interfaced with it like any other serial device plugged into a microcontroller.

## Relevant links

* Code for the Teensy is located in this repository: <https://github.com/launch-control-app/LaunchControlHardware>
* Visual Studio Code: <https://code.visualstudio.com/>
* Platform I/O extension: <https://docs.platformio.org/en/latest/ide/vscode.html>
* OBD-II UART Library: <https://github.com/stanleyhuangyc/ArduinoOBD>
* Tutorial for Interfacing with Bluetooth module: <https://causeyourestuck.io/2015/12/14/communication-between-Android-and-hc-06-module/>

# Getting started with the Android application

For developing the android application (Java), we used Android Studio. Due to the limited capabilities of Bluetooth, we were not able to run the app on the emulator, and hence tested and ran builds only using an actual Android Device.

For interfacing with Bluetooth, the web as well as displaying graphs, we used open source libraries.

For providing a map, for which we used Google’s Mapping library. It is important that you register for an API key with Google’s developer console to use the map.

## Relevant links

* Code for the Android application is located in this repository: <https://github.com/launch-control-app/LaunchControlApp>
* Bluetooth Library: <https://github.com/OmarAflak/Bluetooth-Library>
* Web Socket Library: <https://socket.io/blog/native-socket-io-and-android/>
* Graphing Library: <https://github.com/PhilJay/MPAndroidChart>
* Google’s developer console: <https://console.developers.google.com>

# Getting started with the Web application

The web application was written using NodeJS for the back end and ReactJS for the front end. The getting started guide for running these two components is located on the main page of the web app’s repository. Before running them, installing Yarn and NodeJS is required.

Libraries used for mapping, graphing and authenticating users are linked below.

## Relevant links

* Code for the Web application is located in this repository: <https://github.com/launch-control-app/LaunchControlWebsite>
* Yarn homepage: <https://yarnpkg.com/en/>
* NodeJS homepage: <https://nodejs.org/en/>
* Getting started with ReactJS: <https://reactjs.org/docs/getting-started.html>
* Maps Library: <https://github.com/uber/react-map-gl>
* Graphing Library: <https://uber.github.io/react-vis/>
* Authentication Library: <http://www.passportjs.org/>

# Python Scripts

In addition to the microcontroller, android and web code, we wrote two scripts in python to mock the microcontroller as well as the web. This was used heavily while developing the app and web server to test out the UI and enable data flow without having the need for the hardware component.

## Android Application Python Script

This script simply sends mock values over bluetooth to the android app. To use it, you must have python installed, as well as the associated Bluetooth library. There is also a second python script that plays back values from a file, if a realistic data set is needed to be used.

You must also change the *“deviceName”* variable in BluetoothManager.java in the android application to your host device (running the python script).

## Relevant links

* Code for python script: <https://github.com/launch-control-app/Testing/blob/master/testAndroidApp.py>
* Code for another python script, that plays back values from a file: <https://github.com/launch-control-app/Testing/blob/master/test_data_playback.py>
* Bluetooth Library: <https://github.com/pybluez/pybluez>
* Installing Python: <https://www.python.org/downloads/>

## Testing Web Server

The reason the web server python script can’t be used directly is because it has no authentication (the script was created before authentication was added to the webserver). However, you can disable authentication in the webserver code to use this script if you’d like.

## Relevant links

* Code for python script: <https://github.com/launch-control-app/Testing/blob/master/testWebServer.py>
* WebSocket Library: <https://pypi.org/project/socketIO-client-nexus/>
* Installing Python: <https://www.python.org/downloads/>

# Appendix A

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Units | Minimum Value | Maximum Value |
| Engine RPM | RPM | 0 | 16383.75 |
| Calculated engine load | % | 0 | 100 |
| Engine coolant temperature | °C | -40 | 215 |
| Absolute engine load | % | 0 | 100 |
| Engine oil temperature | °C | -40 | 210 |
| Engine torque percentage | % | -125 | 130 |
| Engine reference torque | Nm | 0 | 65535 |
| Intake temperature | °C | -40 | 215 |
| Intake manifold absolute pressure | kPa | 0 | 255 |
| MAF flow rate | grams/s | 0 | 655.35 |
| Barometric pressure | kPa | 0 | 255 |
| Vehicle speed | km/h | 0 | 255 |
| Engine running time | s | 0 | 65535 |
| Vehicle running distance | km | 0 | 65535 |
| Throttle position | % | 0 | 100 |
| Ambient temperature | °C | -40 | 215 |
| Vehicle control module voltage | V | 0 | 65.535 |
| Fuel tank level input | % | 0 | 100 |

Table A: Logged OBD-II data points