CS5041 P4: Real-time automobile data visualisation in Processing

120008200
University of St Andrews

INTRODUCTION

For years, I have been curious about in-vehicle computer systems. How sophisticated are they? What kind of data can they provide? Is is possible to interact with them without manufacturer-specific hardware? Practical 4 provided me with the opportunity to explore these systems.

On-board diagnostics (**OBD**) computers became widely standardized in the 1990s, primarily for the purposes of emissions testing and fault diagnosis [1]. Most automobiles manufactured since the 2000s follow the ODB-II standard, which is the interface I have used in my implementation.

The solution presented in this report demonstrates how to interact with a vehicle over ODB-II using the Processing programming language.

GETTING CONNECTED

The ODB-II data link connecter (DLC) is the endpoint at the vehicle. The DLC is a universal 6-pin connecter, usually located below the driver's steering wheel (Figure 1).

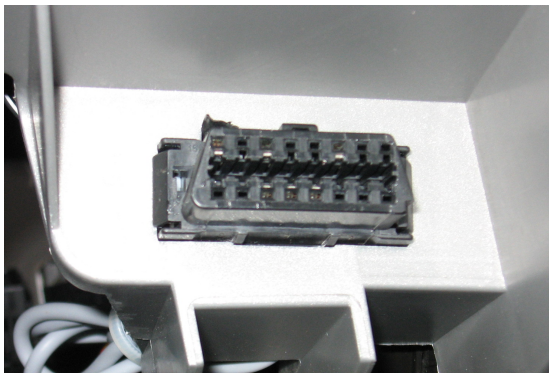

Figure 1: Female ODB-II DLC [5]

An ODB-II to DB9 cable (left in Figure 2) is connected to the vehicle. The DB9 end of the cable is connected to a SparkFun ODB-II UART [3] which interfaces the vehicle computer. Finally, I have connected the SparkFun FTDI Basic Breakout [4] to the ODB-II UART by soldering male headers onto the board. Now we have a micro-USB (type B) connecter to the vehicle, allowing us to connect to a personal computing device (e.g. MacBook Pro).

The ODB-II UART is an interpreter used to convert messages between the various regional flavours of the ODB-II protocol, and UART. This allows us to interact with vehicles using an industry standard (ELM327) command set.

Figure 2 shows all of the hardware requirements for the project. In this diagram, the leftmost connecter is plugged into the vehicle, the rightmost to my MacBook. These components (or variants of them) can be purchased from various electrical component stores at small cost.

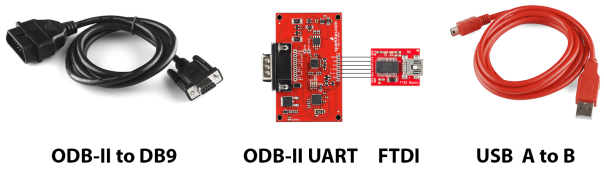

Figure 2: Hardware requirements

Figure 3 shows all of the components connected and ready for communication.

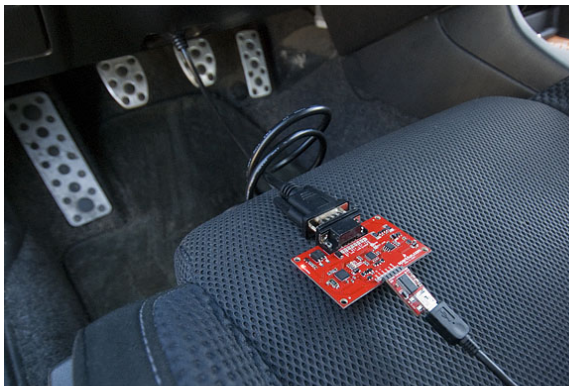

Figure 3: Connected to the vehicle [6]

PROCESSING

Once the USB connection with the vehicle was established, I tested it from a serial terminal by typing in commands manually. The serial connection with the UART uses 9600 bps, 8 data bits, 1 stop bit and no parity.

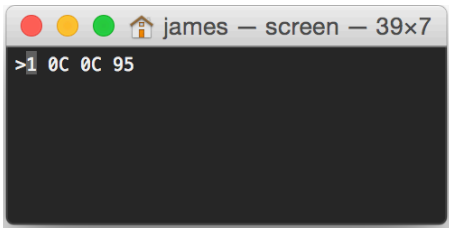

Figure 4: Testing the "010C" command (read engine RPM) in a serial terminal.

Figure 4 shows the response given for the 010C (read engine RPM). The first part of the response (1 0C) is simply

reflecting the command. The second part (0C 95) is the value (in quarters of RPM). 0C95 is 3321 in decimal, dividing by 4 (to account for the response being in quarters of RPM) gives an RPM value of 830.

The goal of this project was to visualise live vehicle data in Processing. The official 'Serial' library was perfect for communication with the UART, allowing me to send and receive data directly from Processing using *serial.write* and *serial.read* commands. The *Vehicle* class in my Processing sketch creates a level of abstraction over vehicle communication, allowing the rest of the program to call functions like *vehicle.getRPM()* to request data.

Setting up reliable communication between Processing and the vehicle was quite fiddly at first, and required me to gain a thorough understanding of the ODB-II UART. One of my early issues was caused by trying to read data too soon after sending requests. The UART has a max response time of 250 milliseconds. I had initially set this time to 200 milliseconds which led to some spurious responses.

Once I had established a reliable connection, and a suite of functions exposing the vehicle data, I was able to move on to data visualisation.

There are many different types of data available from the vehicle computer, the full list of which can be found online [7]. I picked a few of the most familiar data types to visualise for demonstration purposes:

- Engine RPM
- Vehicle speed
- Engine coolant temperature
- Intake air temperature
- Throttle position
- Engine load

Although these were the only data types used in this demonstration, there are many other metrics available which might be of interest. I have taken a modular approach in the design of this solution, making it easy to add or remove different metrics with just a few modifications to source code.

The program starts up in a main menu screen (figure 5) with buttons to connect to the vehicle or quit the program.

Figure 5: The main menu screen allows the user to connect to the vehicle.

Clicking the connect button will attempt to send a reset command to the UART board ("ATZ" command). If the connection is successful, the user is taken to the live graph screen (figure 6). This screen has a top bar containing buttons which allow the user to switch between the available graph metrics. The rest of the screen space is dedicated to the information which is most important: the real-time graph.

Care was taken in designing the live graph screen. I didn't want to lose valuable graphing space for the title, but it needed to be clear what the current graph metric was. I decided to put the graph title (graph metric) in the background of the graph. The colour of this text is a slightly lighter shade of the background colour, preventing it from being too distracting, but also making it clear to the user what the current graph metric is.

Figure 6: Real-time data graph for engine RPM

CONCLUSION

I have created a working demonstration of a connection between automobile diagnostics computers and Processing, using just a few, widely available, low-cost electronic components. This connection has allowed me to create real-time visualisations of engine metrics by leveraging the power of the Processing environment.

REFERENCES

1. On-Board Diagnostics (OBD), United States Environmental Protection Agency http://www3.epa.gov/obd/
2. http://www.onboarddiagnostics.com/page03.htm
3. SparkFun OBD-II UART https://www.sparkfun.com/products/9555
4. SparkFun FTDI Basic Breakout - 5V https://www.sparkfun.com/products/9716
5. Female OBD-II connector - M. Minderhoud
6. Connected to the vehicle https://cdn.sparkfun.com//assets/parts/3/3/1/4/09555-05.jpg
7. ODB-II PID's https://en.wikipedia.org/wiki/OBD-II_PIDs