Reading a car's OBDII port with a Raspberry Pi (/blog/reading-a-cars-obdii-port-with-a-raspberry-pi)

05/04/2014 2 Comments (/blog/reading-a-cars-obdii-port-with-a-raspberry-pi#comments)

I have been planning for a while to try to get information from my car and to associate it to other sensors (e.g. a GPS and an accelerometer). The most natural thing would be to connect an OBD port scanner and a phone. There are several commercial tools of this type and you can find them for a few pounds online. However you will then be constrained to use only the phone's sensors while with the Pi you have free reins to choose the sensors.

What is the OBD port and how to find it
The OBD (On Board Diagnostics (http://en.wikipedia.org/wiki/On-board_diagnostics)) port is used by garages to test the electronics of a car. It is mandatory on every car since the 90s. It is easily accessible and generally located under the steering wheel. In order to find the port in my car (a Mercedes Benz B Class), I just googled "OBD port Mercedes B class" and found several pages providing instructions including pictures and even a couple of videos.

Finding the OBD scanner
There are several cheap commercial readers available, I decided to go for a bluetooth enable (http://www.amazon.co.uk/Diagnostic-ELM327-Interface-Bluetooth-Version/dp/B00BMHAILG/)d (http://www.amazon.co.uk/Diagnostic-ELM327-Interface-Bluetooth-Version/dp/B00BMHAILG/)ELM327 (http://www.amazon.co.uk/Diagnostic-ELM327-Interface-Bluetooth-Version/dp/B00BMHAILG/)

Reading from the OBD scanner in Python
As I am working on a Raspberry PI I needed to develop in Python. I have found the code developed by (https://github.com/martinohanlon/pyobd) martinohanlon (https://github.com/martinohanlon/pyobd) which is available on github and that is able to read from the OBD scanner. The code is created for a motorbike but it works for cars as wells. Also it was originally created for a wired scanner. However if you pair the scanner with the Pi as explained below, it will also work with bluetooth.

Preparing the Pi
The Pi will need a bluetooth scanner. There are several commercial ones. I went for a USB one. Installing it is very easy. Just plug, update the libraries and play. ModPy provides an excellent tutorial (https://www.modmypi.com/blog/installing-the-raspberry-pi-nano-bluetooth-dongle). I then connected a GPS to the pi. A brilliant tutorial is provided by @dmandle (http://www.danmandle.com/blog/getting-gpsd-to-work-with-python/).

Pairing the PI and the OBD scanner
As mentioned in the ModPy's tutorial, the bluetooth OBD adaptor needs pairing with the Pi. To get the address of the adaptor run in the Pi's shell:

```
#     hcitool scan
```

it will return:

```
#     Scanning ...
#           <bluetooth address> OBDII
#     ...
```

The scanner will be called OBDII (or similar name). Its bluetooth address will be something like 00:0D:18:3A:67:89.
To pair it with the Pi, I used the following command:

```
#        echo 1234|bluez-simple-agent hci0 <bluetooth address>
```

Note: this supposes that the PIN code of the scanner is 1234, which is generally the case. If different, change

Author

I am professor of Knowledge and Language Technologies at The University of Sheffield. These pages here are dedicated to my experiments with electronics (e.g. Raspberry Pi and Arduino). Some of these experiments I do for work, others just for fun.

Archives

December 2013 (/1/archives/12-2013/1.html)

Categories

All (/1/category/all/1.html)

RSS Feed (/1/feed)

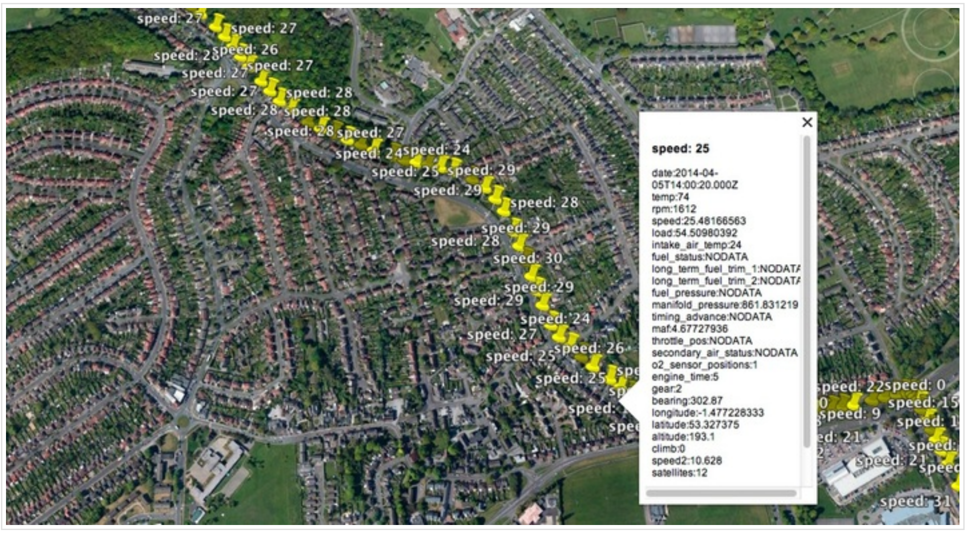
appropriately.
If the Pi complains that bluez-simple-agent is not found, then go back to the ModPy's tutorial and install bluez (with sudo apt-get install bluetooth bluez-utils blueman).

In some adaptors the bluetooth connection drops after few seconds after pairing. if so, in a separate shell run

```
#          sudo rfcomm connect 0 <bluetoothaddress> 1
e.g.  sudo rfcomm connect 0 00:0D:18:3A:67:89 1
```

The command will regularly poll the scanner so to avoid dropping the connection. Please note that the command will hang the shell, so add an '&' if you do not want that.

Putting everything together
At this point I just had to make sure that every time the OBD port was polled, the GPS was polled as well. The results were saved as a csv log file. I then created a python script taking the csv file and turning it into a kml file as required by Google Earth. Here is an example of the result (speed is in miles per hour).



```
speed: 25

date:2014-04-
05T14:00:20.000Z
temp:74
rpm:1612
speed:25.48166563
load:54.50980392
intake_air_temp:24
fuel_status:NODATA
long_term_fuel_trim_1:NODATA
long_term_fuel_trim_2:NODATA
fuel_pressure:NODATA
manifold_pressure:861.831219
timing_advance:NODATA
maf:4.67727936
throttle_pos:NODATA
secondary_air_status:NODATA
o2_sensor_positions:1
engine_time:5
gear:2
bearing:302.87
longitude:-1.477228333
latitude:53.327375
altitude:193.1
climb:0
speed2:10.628
satellites:12
```

Tweet 2   Like 0

Comments

Dominic Threlfall (http://www.home-technology.com.au)   23/06/2014 11:26am

Content of this blog is too helpful for mass people. Thanks for submitting this kind of Blog.

Reply

domantylar (http://www.autophix.com/product_list.html)   04/07/2014 8:10am

i have an obdmate om580 scanner ,always using

Reply

Leave a Reply

Name (required)

Email (not published)

Website

Comments

Notify me of new comments to this post by email

Submit