CSC 591, CSC 791, ECE 592, ECE 792
Internet Of Things –
Architectures, Applications, and Implementation
SPRING 2018

Final Project Report

DriverMonitor: Young Driver Behavior Monitoring System Using Smartwatch and OBD II Sensor

Team 6
Jason Nance
Kriti Sharma
Shaohu Zhang

Instructor: Muhammad Shahzad

04/26/2018
North Carolina State University

TEAM CONTRIBUTIONS:

Component	Weight	Jason	Kriti	Shaohu
Develop smartwatch /smartphone App	0.2	25%	25%	50%
Cloud connection	0.2	50%	25%	25%
Data visualization	0.1	50%	25%	25%
Algorithm development	0.2	25%	50%	25%
System integration	0.1	25%	25%	50%
Field test	0.1	25%	50%	25%
Report writing	0.1	33.3%	33.3%	33.3%
Per student aggregate contribution		0.2*25+0.2*50+0.1*50+0.2*25+0.1*25+0.1*25+0.1*33.3=33.33	0.2*25+0.2*50+0.1*50+0.2*25+0.1*25+0.1*25+0.1*33.3=33.33	0.2*25+0.2*50+0.1*50+0.2*25+0.1*25+0.1*25+0.1*33.3=33.33

DriverMonitor: Young Driver Behavior Monitoring System Using SmartWatch and OBD II Sensor

Shaohu Zhang, Kriti Sharma and Jason Nance
North Carolina State University

1. Introduction

Young drivers tend to be at high crash risk relative to experienced drivers [1]. According to Annual United States Road Crashes Statistics, nearly 8,000 people are killed in crashes involving drivers ages 16-20 every year in the United States [2]. Researchers [3,4] reported that novice drivers have particularly high crash rates in the first 6 months just after licensure. Those crashes are more likely to involve speeding and hitting stationary objects, which could reflect both risk-taking and poor judgment.

Getting knowledge of your driver's performance is important to the researchers and parents. Pressing the need of development of an application that addresses the need of monitoring the young driver. The logic being, that a young driver is expected to be more careful while driving when he/she knows that his/her activities are being monitored by a guardian. The driver is expected to behave more responsibly when he/she can be held accountable for the actions. Therefore, we proposed the Driver Monitor system to keep track of young driver's performance through his/her physical activity and vehicle's sensors.

The objectives of this project were to examine over time novice teenage driving performance and risk, including kinematic risky driving and speeding. Driver Monitor enables to perform two tasks: 1) track and monitor driving behavior in real-time manner; this includes the acceleration patterns, speed of the vehicle, speed limit adherence, and driver's heart rate; 2) send parents alert for abnormal activities, e.g., speeding and hard brake.

2. System Design

Figure 1 illustrates the system design. Driver Monitor has three main components including vehicle sensors end, mobile application and web server application.

Vehicle sensors end

This component comprises an On-board diagnostic (OBD-II) sensor and Raspberry Pi. Since 1996, all cars manufactured in the United States/ to be sold in the United States must be installed with an OBD port, which can be accessed to capture the status of the various vehicle subsystems. The vehicle's sensor readings (e.g., vehicle speed, fuel remaining, RPMs) from OBD-II port can be requested and processed by a Raspberry Pi through Bluetooth. Then the Pi sends data to the cloud database.

Mobile application

A smartwatch will measure the driver 's heart rate if the server requests it, and smartphone reports GPS location and heart rate to the cloud server.

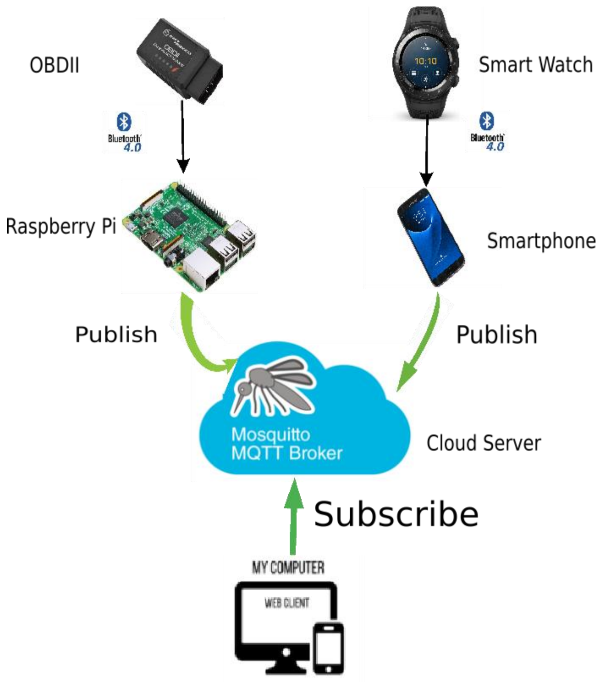


Figure 1 System Design

Web application

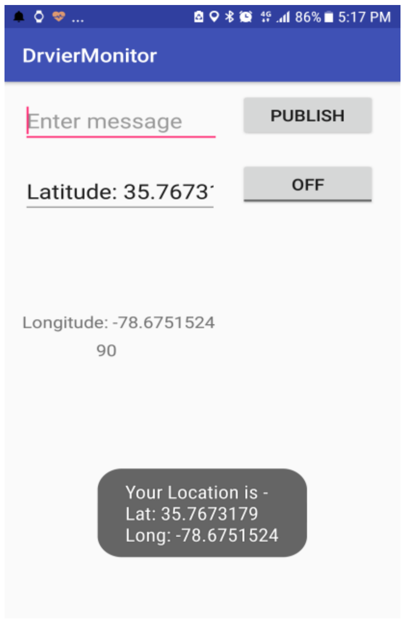
Web server accesses the database stored in the Cloud and displays a dashboard showing parents how the child is driving (e.g., heart rate and speed)

3. Implementation

As shown in Figure 2, the BAFX Bluetooth Diagnostic OBDII Reader ($22) [5] is used to access vehicle's sensors reading. For the convenience of operation, we use the Raspberry Pi B3 to read OBD-II. The open source Python-OBD library [6] was used to read data from a car's OBD-II port. This library is designed to work with standard ELM327 OBD-II adapters, which can stream real time sensor data, perform diagnostics (such as reading check-engine codes), and is fit for the Raspberry Pi. A python program named as obdReader.py reads data and sends data to the MQTT broker in the Google Cloud. The vehicle sensors topic is topic/obd2.

Figure 2 BAFX Bluetooth Diagnostic OBDII

A mobile application with Android Wear 2 was developed using the Android Studio 3.0 Software Development Kit (SDK) [7]. Figure 2 shows the interface design of the mobile App. The smartphone App shows the GPS location and heart rate. For the purpose of demo, the Publish button is designed for sending message manually. For example, if the GPS location is (33.557423, -74.897743), and the heart rate is 89, the message will be '33.557423, -74.897743,89'. The on/off button is designed to publish to the topic automatically if the user request message in each time interval (i.e. 5 seconds). The mobile application publishes to the "topic/watch" topic. Figure 4 demos the test scenario in a car. The mobile App was running on a Samsung Galaxy S7 Edge. The watch Wear system was running on a Huawei Smartwatch 2.


DrvierMonitor

Enter message

PUBLISH

Latitude: 35.7673'

OFF

Longitude: -78.6751524

90

Your Location is -
Lat: 35.7673179
Long: -78.6751524

Figure 3. Mobile App interface

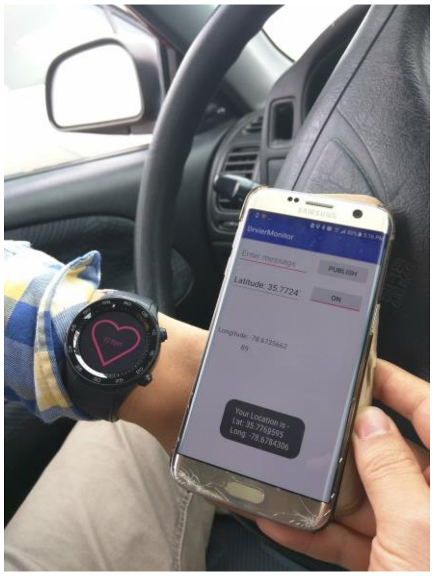
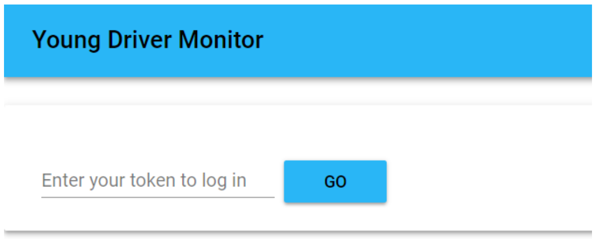


Figure 4 Smartwatch and smartphone app interface

On the Google Cloud, several application components run in Docker containers and work together to display the dashboard for users. The first container contains a Mosquitto MQTT broker protected by a username and password, which is available on the open Internet for devices to publish data to. In another container, a python script subscribes to all the topics that were being published by the application to the MQTT broker. As it receives the data from published topics, it stores them in a time-series database (InfluxDB, in another Docker container) tagged with the token of the device that sent it. The sensor readings are then stored in the database indefinitely.

A third Docker container houses a Go API server, which can then query the database by device token at some time interval. The front-end dashboard is a single-page app written in React (A JavaScript Library for building User Interfaces). The application first asks the user to log in with their device token as shown in Figure 5. Once logged in, the user can query the data for the given device token across a selectable time interval as shown in Figure 7. The website in turn gets this data from the Go API server as shown in Figure 6.


Young Driver Monitor

Enter your token to log in

GO

Figure 5 Web dashboard

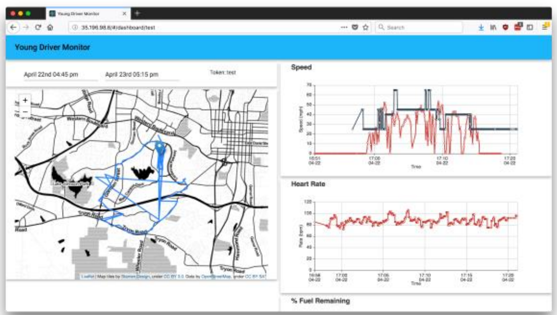


Figure 6 Web dashboard

Graphs on the dashboard show the location (Figure 8), speed (Figure 9), driver's heart rate (Figure 10), fuel level (Figure 11), and engine RPM (Figure 12).

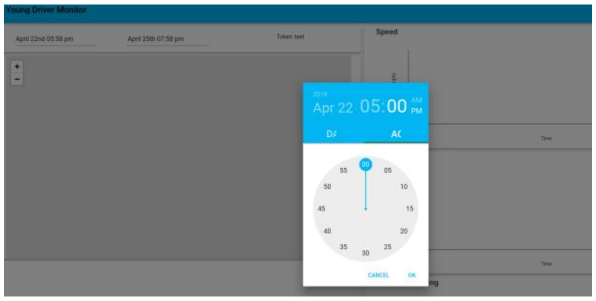


Figure 7 Time period selection interface

4. Result and Discussion

Figure 8 shows a 5-mile test route around the centennial campus at North Carolina State University. The Android-based application measures the driver's heart rate every 5 seconds. The application samples the geolocation every second and transmits the acquired data after every 5 seconds. The OBDII sensor monitors engine's characteristics, like fuel remaining, acceleration, and braking. The data is sampled every second by the sensor.

The map shown in Figure 8 attempts to track the whole path that was followed by the vehicle. The location uploaded to the web server is stored as a data point; the application joins all the data points plotted on the map to form a path. The path is curved to the nearest known road on the map.

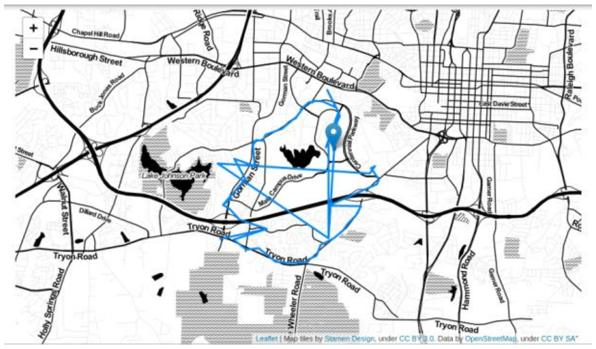

Hillsborough Street
Western Boulevard
Tryon Road
Lake Johnson Park
Gorman Street
Wheeler Road
Leaflet | Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under CC BY SA"

Figure 8 Driver's locations mapping

The speed graph (Figure 9) with the reference from geolocation data, shows the speed limit for the road that vehicle is travelling on. It accesses the speed limit of the location by downloading all the roads for the covered area from the Overpass API (Database over the web), which serves OpenStreetMap data. The website then links each sensor reading to the closest road from the network and records the speed limit of that road, later serving the purpose of determining our alert criterions.

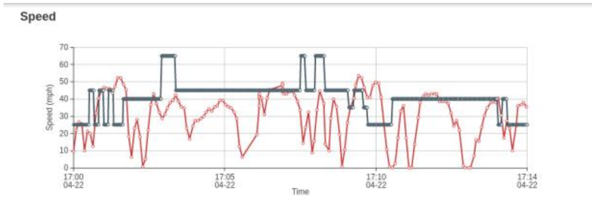


Figure 9 Vehicle speed(red) vs speed limit(black)

As shown in Figure 10, the smartwatch application is programmed to sense the heart rate every 5 seconds. The interval could have been larger, but since the application is targeting real time scenarios, we do not know when an accident-like situation might come up. Hence, we kept the interval of five seconds, which doesn't create a huge overhead even for locations with bad or slow internet connectivity. However, it is not necessary to sense heart rate every 5 seconds. It can be requested when an accident-like situation occurs in the future design, e.g., the hard-braking event.

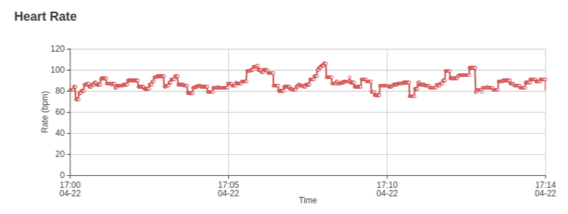


Figure 10 Driver's heart rate

Figure 11 shows the fuel level. Figure 12 shows the Revolution Per Minute (RPM) of the vehicle engine. RPM is a measure of the frequency of rotation, specifically the number of rotations around a fixed axis in one minute. It is used as a measure of rotational speed of a mechanical component.

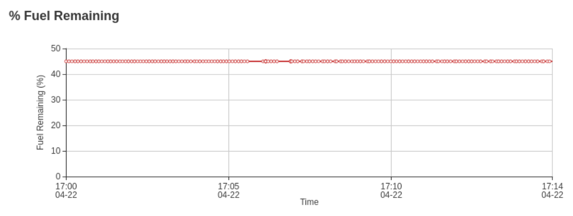


Figure 11 Fuel level

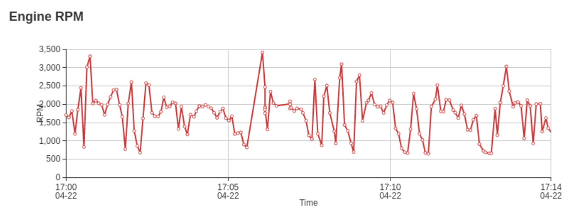


Figure 12 Engine revolution per minute

This study includes the design, implementation and evaluation of the DriverMonitor system. The system uses the Raspberry Pi to read OBDII in the current design. However, this can be simply integrated using a single smartphone to link smartwatch and OBDII. We haven't considered the acceleration rate due to the incompatible issue in the current test car. Recent research [8] indicated that drivers involved in crash events took 0.487 seconds longer to react and decelerated at 0.018 g's slower than drivers in equivalent near-crashes.

The proposed system provided in this report has the potential to apply DriverMonitor system for monitoring young driver performance. However, as with any system, there are limitations. Some limitations in this study include the following:
1) We found the OBDII sensor we use is not compatible with all vehicle models due to the various OBDII protocols. However, this can be solved by developing compatible software.
2) There was no enough filed experiment to examine the system integration, for example, how to handle with the Bluetooth connectionless case.
3) Only one driver was evaluated in the field test. The experiment participant was not a novice driver.

The angle of the steering wheel can be measured to monitor the young drive's maneuver behavior. Future work should include the acceleration rate and the angle of the steering wheel. While DriverMonitor is designed for young drivers, it could also be applied to other ages such as elder people.

Reference

[1] Simons-Morton, Bruce, Johnathon Ehsani, Pnina Gershon, Sheila Klauer, and Thomas Dingus. "Teen Driving Risk and Prevention: Naturalistic Driving Research Contributions and Challenges." Safety 3, no. 4 (12, 2017): 29. doi:10.3390/safety3040029.
[2] Annual United States Road Crashes Statistics http://asirt.org/initiatives/informing-road-users/road-

safety-facts/road-crash-statistics Accessed by 4/24/2018.

[3] McCartt, A.T.; Shabanova, V.I.; Leaf, W.A. Driving experience, crashes, and traffic citations of teenage
beginning drivers. Accid. Anal. Prev. 2003, 35, 311–320

[4] Simons-Morton, B.G.; Ouimet, M.C.; Zhang, Z.; Lee, S.L.; Klauer, S.E.; Wang, J.; Chen, R.; Albert, P.E.; Dingus, T.E. Crash and risky driving involvement among novice adolescent drivers and their parents. Am. J.Public Health 2011, 101, 2362–2367.

[5] BAFX Bluetooth Diagnostic OBDII https://www.amazon.com/gp/product/B005NLQAHS/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

[6] Python-obd library http://python-obd.readthedocs.io/en/latest/

[7] Android Studio https://developer.android.com/studio/index.html

[8] Wood, Jonathan, and Shaohu Zhang. Evaluating Relationships Between Perception-Reaction Times, Emergency Deceleration Rates, and Crash Outcomes Using Naturalistic Driving Data, MPC-17-338. North Dakota State University - Upper Great Plains Transportation Institute, Fargo: Mountain-Plains Consortium, 2017.