Driving and Fuel Consumption Monitoring with Internet of Things

https://doi.org/10.3991/ijim.v11i3.6473

Emir Husni
Institut Teknologi Bandung, Bandung, Indonesia
ehusni@lskk.ee.itb.ac.id

Abstract—Problems related to fuel consumption efficiency are likely to occur in a car rental company. A system for the Internet of Things – VISCar to monitor fuel consumption on Android mobile application is proposed to solve the problem. The VISCar has four features, including driving behavior analysis, monitoring, notification, and location and route to the gas station. This system consists of three subsystems: connector, server, and user interface. Car's engine data is scanned by OBDII and sent via Bluetooth to Raspberry Pi then passed using 3G connection to the server and being stored. The data can be accessed by users through a mobile application via MQTT and processed on server for analysis. The test results show VISCar has been successfully built and implemented to monitor fuel consumption efficiency for the car rental company.

Keywords—Bluemix, Fuel Efficiency, Gas Station Location, IoT, Monitoring.

1 Introduction

There is fierce competition among car rental companies to gain profit. To increase the benefit, they need to manage their finances well, either income or outcome. It is such a waste if the outcome increases due to the inefficient fuel consumption.

One of the factors that affect fuel consumption efficiency is driving behavior. Therefore, the monitoring of driving behavior will be needed. Besides, fuel efficiency degenerates as the fuel tank is in low level. This condition is often neglected by the drivers. Insufficient information about the nearest gas station worsens it.

Those problems are likely to occur in a car rental company. They can also disturb the passengers' comfort during the trip and inflict a financial loss to the company. Hence, the car rental company needs something to solve the problem

The System for Internet of Things – VISCar to monitor fuel consumption on Android mobile application is proposed to solve the problem. The features are divided into two: features for owner and features for the driver. The features for the owner consist of analysis report and monitoring while the features of the driver consist of notification also location and route to the nearest gas station. This paper will discuss the literature review, design, implementation, and testing of the system, also the conclusion

2 Literature Review

2.1 On Board Diagnostic (OBD)

OBD is an automotive term referring to a vehicle's self-diagnostic and reporting capability. OBD systems give the vehicle owner or technician access to the status of the various vehicle subsystems. OBDII is an improvement over OBDI in both capability and standardization. The OBDII standard specifies the type of diagnostic connector and its pin out, the electrical signaling protocols available, and the messaging format. It also provides a candidate list of vehicle parameters to monitor along with how to encode the data for each. There is a pin in the connector that provides power for the scan tool from the vehicle battery [1-2].

In general, OBDII socket is located under the steering wheel. On the latest OBDII standard, SAE J1962, the socket of OBDII consists of sixteen pins. The SAE J1979 standard defines a method for requesting various diagnosis data and a list of standard parameters that might be available from the ECU. The various parameters that are available are addressed by Parameter Identification Numbers (PID) [3].

Table 1. Parameter Used Based on SAE J1979

PID (hex)	Data bytes returned	Description	Min value	Max value	Units
0D	1	Vehicle speed	0	255	km/h
10	2	MAF air flow rate	0	655.35	g/s
11	1	Throttle position	0	100	%
2F	1	Fuel Tank Level Input	0	100	%
31	2	Distance traveled since codes cleared	0	65,535	km

There are ten modes of operation described on SAE J1979, but the VISCar uses only mode 01 that shows current data. This mode provides more than 100 cars' engine data that can be read. Vehicle manufacturers are not required to support all PIDs and there can be manufacturer defined custom PIDs that are not defined in the OBDII standard. The VISCar focuses only on five cars' engine data, including speed, throttle position, fuel level, distance, and mass air flow. The PID description used in here is shown in Table I [3].

2.2 Internet of Things (IoT)

IoT is a network which connects different kinds of devices. According to the definition of International Telecommunication Union, the Internet of Things is a network which can connect the message instruction and sensor equipment including RFID, IR sensor, GPS, and so on to implement the exchange and sharing of information and eventually we can get a simultaneous and intelligent management network [4-7].

The structure of IoT comprises three layers: information acquisition layer, network transmission layer, and application processing layer. The information acquisition layer to achieve the object of perception and information collection. It uses sensor technologies to monitor the real time changes of the surrounding environment. Network transmission layer mainly realizes the transmission of the acquired information and the interconnection among equipment. The application processing layer does various computing and processing of the information, finally provides intelligent application services according to the different needs [8].

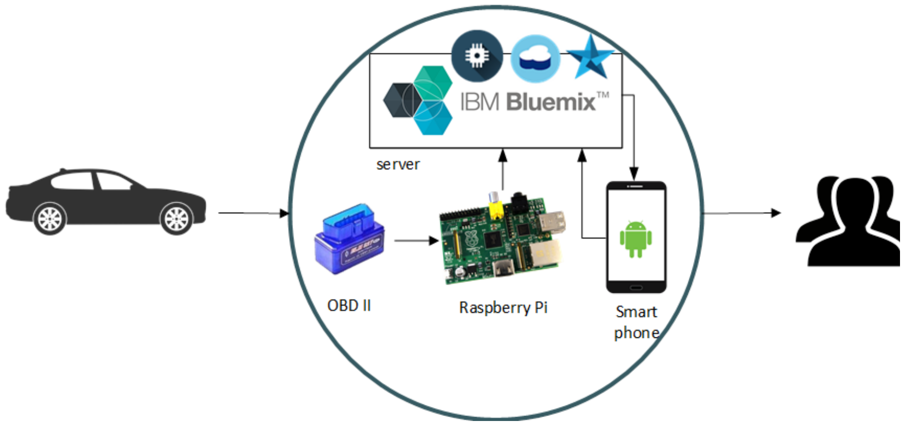


Fig. 1. VISCar system architecture

2.3 IBM Bluemix

Bluemix is an implementation of IBM's Open Cloud Architecture, on Cloud Foundry that enables you to rapidly create, deploy, and manage your cloud applications. Because Bluemix is for Cloud Foundry, you can tap into a growing ecosystem of runtime frameworks and services. In addition to providing additional frameworks and services, the Bluemix provides a dashboard for you to create, view, and manage your applications and services as well as monitor your application's resource usage. The Bluemix dashboard also provides the ability to manage organizations, spaces, and user access [9].

Bluemix provides access to various services that can be incorporated into an application. Some of these services are delivered through Cloud Foundry. Others are delivered from IBM and third party vendors. New and enhanced services are added to the catalog often. Fig. 1 illustrates the architecture of the VISCar system using IBM Bluemix.

2.4 Fuel Consumption

Fuel consumption, f, is calculated using two variables, vehicle speed and mass air flow, based on (1) [10].

$$f = \frac{\mathrm{VSS}\times\lambda}{\mathrm{MAF}}\times\alpha \qquad (1)$$

where VSS is the vehicle speed each second in km/hour, MAF is the mass air flow in g/s, λ is a constant to yield f in MPG (miles per gallons), and α is a constant to convert MPG to km/liter [10].

Besides vehicle speed, fuel consumption is also affected by the throttle position. Pushing the throttle will make the fuel flow to the combustion chamber. The maximum value of throttle position that gives the optimum fuel consumption is 50% [10-13].

2.5 Message Queuing Telemetry Transport (MQTT)

MQTT was invented by Dr. Andy Stanford-Clark of IBM, and Arlen Nipper of Arcom (now Eurotech), in 1999. MQTT stands for MQ Telemetry Transport. It is a lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks [14, 15]. The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

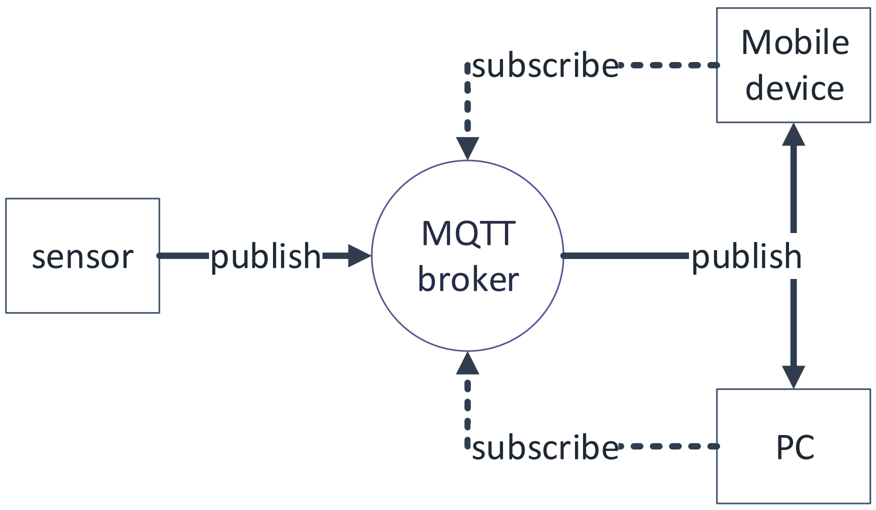


Fig. 2. Publishing and subscribing mechanism

The MQTT protocol is on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. The broker and MQTT act as a simple, common interface for everything to connect to. Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a file system. The advantage of the system publish/subscribe is space and time decoupling, where the sender and recipient do not know each other because there are brokers between them and do not need to be connected simultaneously. Fig. 2 illustrate the principle of publishing and subscribing mechanism.

2.6 Node-RED

Node-RED is a tool for wiring together hardware devices, APIs and online services in new and interesting ways. Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range nodes in the palette. Flows can be then deployed to the runtime in a single-click. JavaScript functions can be created within the editor using a rich text editor. A built-in library allows you to save useful functions, templates or flows for re-use. The lightweight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking mode. This makes it ideal to run at the edge of the network on low-cost hardware, including the Raspberry Pi as well as in the cloud. Node-RED is one of the services that available in the IBM Bluemix [9].

3 Design and Implementation

To realize the VISCar system, three subsystems are needed: connector, server, and user interface.

3.1 Connector Subsystem

The specifications of connector subsystem are as follows.

1. Be able to communicate with OBDII to read the car's engine current data [1, 2].
2. Be able to translate the car's engine raw data to car's engine value that can be understood by human using SAE J1979 standard [3].
3. Be able to send the data to server every second.
4. Be able to store the data in database server.
5. Be able to function properly in car.
6. Have a reliable connection between devices to ensure data flow in real time (except for provider and OBDII connectivity).

The Connector subsystem built to read and translate car engine data, send it to the server and store it in the database. Within this subsystem, car engine data is scanned by OBDII and sent via Bluetooth to Raspberry Pi then passed using 3G connection to the server and being stored. Raspberry Pi takes car engine data by sending the proper Parameter Identification Number (PID) of each sensor. The data are then sent to the server via MQTT and Node-RED. The data flow of the connector subsystem is illustrated in Fig. 3.

OBDII is used because it is a standard for reading car's engine data since 2006. OBDII send the data to Raspberry Pi via Bluetooth and Raspberry Pi sends the data to the server via 3G connection. Bluetooth is used because it is wireless and has the optimum data rate. Modem USB 3G is used because it is portable, small, and using Raspberry Pi as a supply.

The VISCar needs five car's engine data, including speed, throttle position, fuel level, distance, and mass air flow.

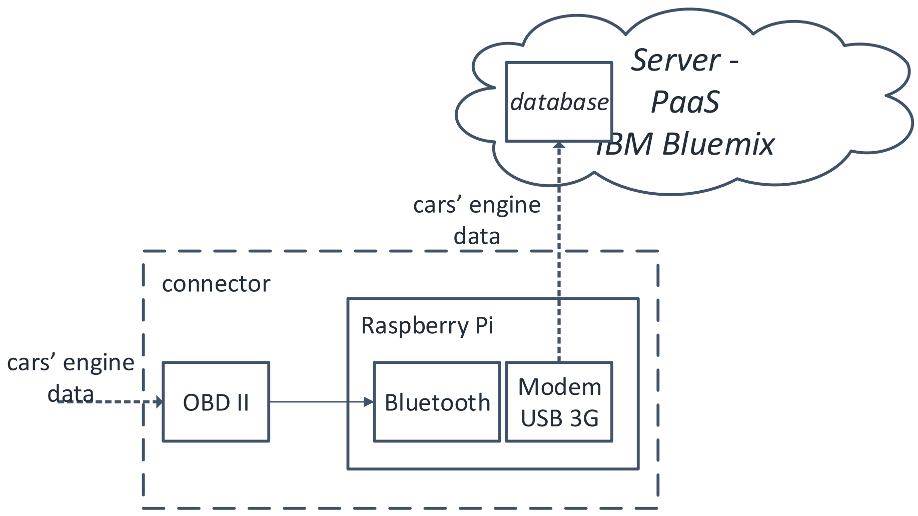


Fig. 3. Dataflow of connector subsystem

3.2 Read and Translate Car Engine Data

Raspberry Pi is connected to OBDII using Bluetooth. When the connection is established, Raspberry Pi will read the car's engine data using open source program in Python called pyOBD. pyOBD is a GUI based application as an interface between ELM32x OBDII and user [9-13].

The data stored in a log file called 'datainternalmobil' in /home/pi/pyobd-pi/log/ folder. Data is stored in JSON format for ease of storage in a database. The file is always being overwritten to minimize the size of file memory. The Flowchart of pyobd function can be seen in Fig. 4.

Send the Data to Server: Here, Node-RED is used to facilitate the data transfer from Raspberry Pi to the server. The data in datainternalmobil.log must be sent to Node-RED and 3G connection must be established. Therefore, Raspberry Pi must have Node-RED, MQTT broker, and 3G package. Furthermore, Raspberry Pi must have usb_modeswitch package, so modem USB 3G acts as a modem, not storage and Mosquitto package, as an open source MQTT broker. The Flowchart of Node-RED's flow is shown in Fig. 5.

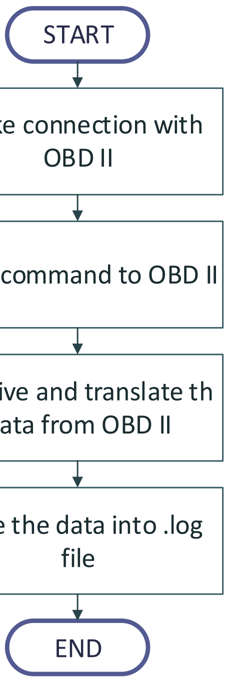

START
Make connection with OBD II
Send command to OBD II
Receive and translate th data from OBD II
Save the data into .log file
END

Fig. 4. Flowchart of pyobd function

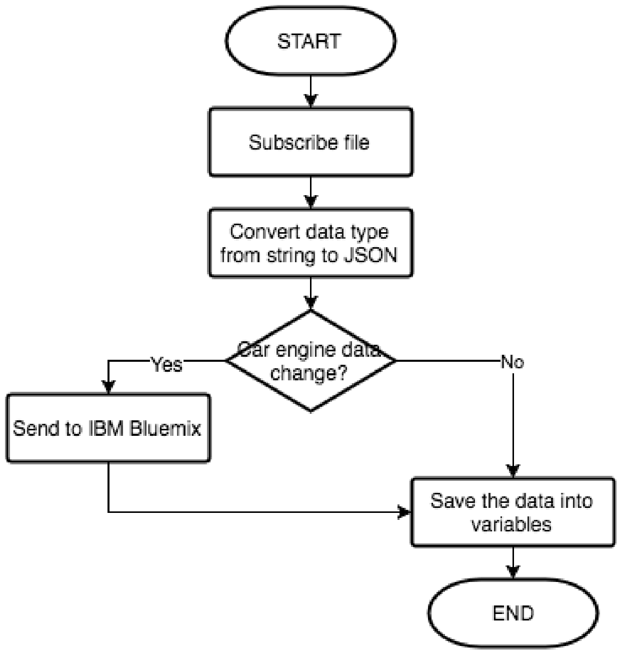


Fig. 5. Flowchart of Node-RED's flow

Store the Data in Database: Node-RED is used to facilitate the data transfer between Bluemix services. After the data being received in server through an IBM IoT node, the data will be passed and stored in Cloudant database but before that, the data will be given an identification, like "datadariobd dd-mm-yyyy hh:mm:ss" to be stored sequentially. The flowchart of Node-RED's flow in Bluemix is shown in Fig. 6.

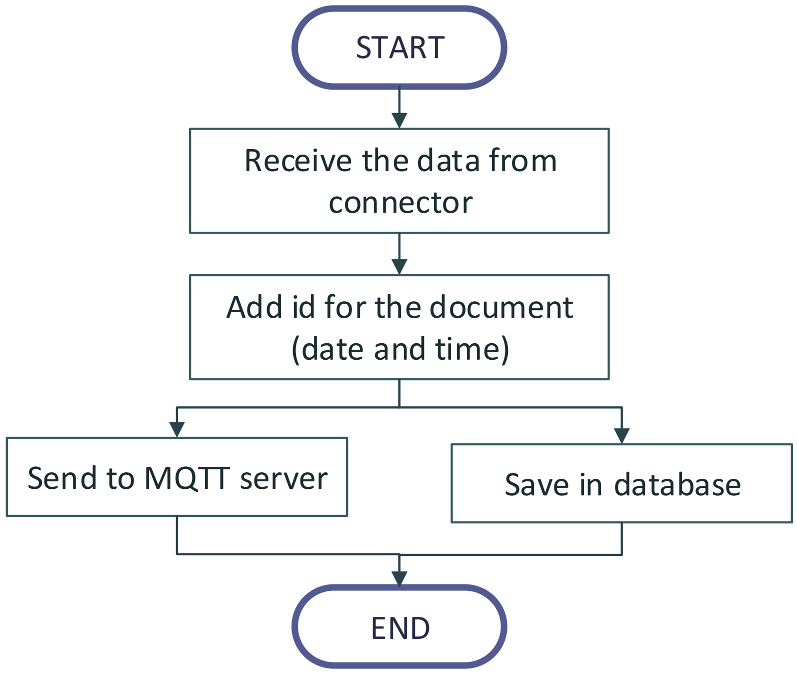

START
Receive the data from connector
Add id for the document (date and time)
Send to MQTT server
Save in database
END

Fig. 6. Flowchart of Node-RED's flow in Bluemix

3.3 Server Subsystem

The function of server subsystem is to manage the transmission, processing, and storing data to/from other subsystems, connector and user interface. In designing this subsystem, MQTT Broker is used as the server in real-time data transmission via MQTT protocol, while HTTP protocol is used to send non real-time data, and IBM Bluemix is the PaaS (Platform as a Service) to store and process data, with IoT Foundation, Apache Spark, Cloudant NoSQL. The data flow of the server subsystem is shown in Fig. 7.

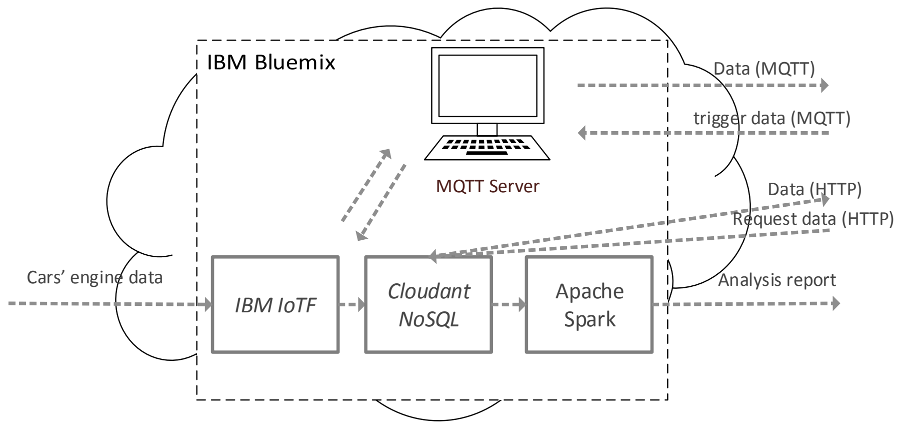


Fig. 7. Dataflow of server subsystem

Analysis Design: One of the features that VISCar has been an analysis feature that intended for the owner of the car rental company. This feature's function is so that the owner could know the driving behavior of his/her drivers that related to fuel consumption efficiency. For this feature to function, the internal data of car collected will be processed. Like I have explained before, there are four data that will be used for this feature to work, including car's speed, throttle position, distance, and mass air flow.

With four kinds of data collected, data processing will be done to see how is the driver's driving behavior, such:

- Calculation of fuel consumption for each second with the formula explained in the literature review section. That fuel consumption will be divided into several categories on the car's speed. Aside from that, the average of the fuel consumption along the trip will also be calculated.
- Calculation how long the driver exceeds the ideal speed. Every car has its own span of speed that produce the optimum fuel consumption. In this analysis, will be calculated how long the driver exceeds that ideal speed, because when that happened the fuel consumption will not be efficient.
- Calculation how long the driver exceeds the ideal throttle position, here exceeds the 50% limit. Like has been explained in the literature review, exceeding the 50% throttle position will make the fuel consumption inefficient.
- Calculation of total distance, traveling time, and the total volume of fuel used along the trip. At the end of the analysis report, that three variable will be displayed as the closing of the report that will be sent to the owner via email.

Programming Algorithm: The data processing will be done by making a program with programming language Python in the Apache Spark service in IBM Bluemix. The programming algorithm used is shown in Fig. 8.

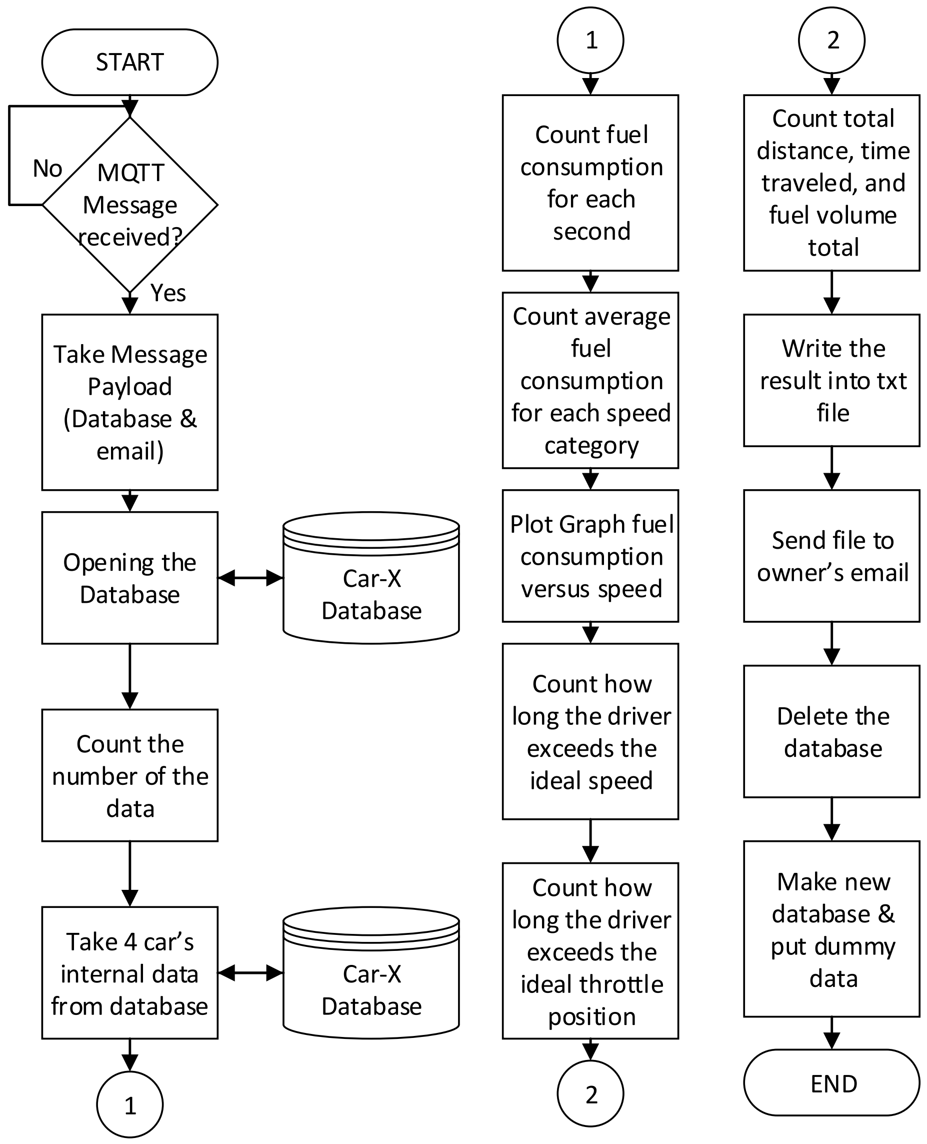

START
MQTT Message received?
No
Yes
Take Message Payload (Database & email)
Opening the Database
Car-X Database
Count the number of the data
Take 4 car's internal data from database
Car-X Database
1
1
Count fuel consumption for each second
Count average fuel consumption for each speed category
Plot Graph fuel consumption versus speed
Count how long the driver exceeds the ideal speed
Count how long the driver exceeds the ideal throttle position
2
2
Count total distance, time traveled, and fuel volume total
Write the result into txt file
Send file to owner's email
Delete the database
Make new database & put dummy data
END

Fig. 8. Programming Algorithm

The program will start when the MQTT server send a message with a topic "log-out", that message will be sent after the owner do the logout process on the driver's smart phone when the driver is back from the trip. In the block "Count fuel consumption for each second", the fuel consumption each second in km/liter will be calculated with the formula from the literature review. Next in block "Count the average fuel consumption for each category", the average fuel consumption in each speed category will be calculated, the speed category will be divided each 10 km/hour.

After the email containing the analysis report has been successfully sent to the owner, to reduce the server capacity, the database containing the car's internal data will be deleted. The deleting process will be done using HTTP request delete, where the link of Cloudant with the database must be included in the program. After the deletion process is complete, a new database with the same name will be made again using the HTTP request put. This process is done, so when the car will be used, the database is ready. Aside from that, a dummy data also will be put in the new database using the HTTP request post, this will prevent a miss reading in the type of the variable that will affect the program.

3.4 User Interface Subsystem

Android application is chosen as the user interface of the system, considering its great privilege like cross compatibility, mashup capability, and Android market share that reached 74.2% in Indonesia in December 2015. In developing the application, Android Studio is used as the IDE with Java and XML as the programming language. Application, based on Android IceCreamSandwich (level API 14) operating system as minimum, as the user interface of VISCar system has the following functions :

- Display car engine real-time data, including vehicle speed, throttle position, fuel level, mass air flow (MAF), and distance for the owner.
- Calculate and display the volume of used fuel during a trip, both for the owner and the driver.
- Generate the notification for the driver in sound and icon display on smart phone whenever the fuel is in critical condition, below 5%.
- Generate a route to the nearest gas station (radius < 30km) from the driver whenever the fuel notification is selected, using GoogleMaps application.
- Generate the notification that can be heard whenever the throttle position exceeds the tolerance (50%) to prevent the inefficient fuel consumption.
- Send a signal to the server to indicate the termination of the trip and to trigger the processing of analysis report email.

Furthermore, the user interface is responsible in user authentication through log in, sign up, and log out process integrated with database on server. This application also has a feature to help in detecting error in data transmission through rest mode interaction by the driver.

Activity of Application: In developing the VISCar application, there are eight main activities to realize the functions of the system. Those activities are arranged in a stack (back stack) that represents the execution order of each activity, as illustrated in Fig. 9.

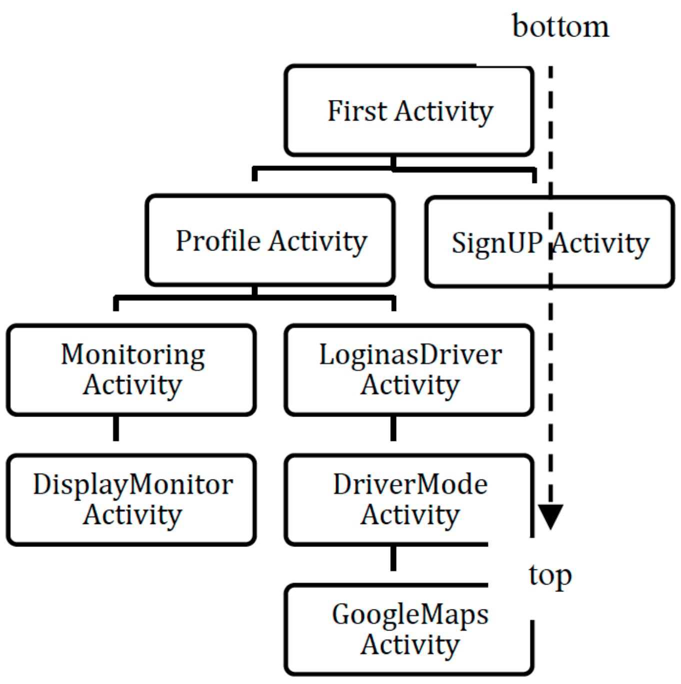


Fig. 9. Hierarchy of VISCar activites

The whole activities are linked to each other in such a way to realize the main functions that can be categorized as follows :

Monitoring: Monitoring functions consist of displaying car engine real-time data, including vehicle speed, throttle position, fuel level, mass air flow (MAF), and distance for the owner also calculating and displaying the volume of used fuel during a trip, both for the owner and the driver. The objective of this function is the owner of car rental company can monitor his/her drivers' driving behavior through car engine data in real-time and remotely. Besides, detecting error in data transmission through rest mode interaction with the driver is also a part of this function. Flowcharts in Fig. 10 and 11 describe the implementation of monitoring function for a driver and an owner respectively.

Notification also Route and Location of Gas Station: The notification also route and location of gas stations consist of issuing notification for the driver in sound and icon display on smart phone whenever the fuel is in critical condition, below 5%, also generating a route to the nearest gas station from the driver using the GoogleMaps application. This function is needed to prevent the neglected critical condition of the driver. Furthermore, issuing the notification that can be heard whenever the throttle position exceeds the tolerance (50%) to prevent the inefficient fuel consumption is also done. The flowchart in Fig. 12 describes the implementation of this function.

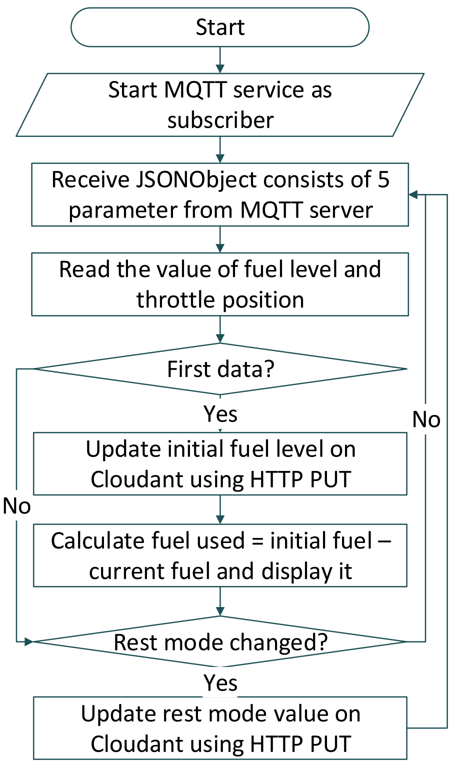

Start
Start MQTT service as subscriber
Receive JSONObject consists of 5 parameter from MQTT server
Read the value of fuel level and throttle position
First data?
Yes
No
Update initial fuel level on Cloudant using HTTP PUT
Calculate fuel used = initial fuel – current fuel and display it
Rest mode changed?
No
Yes
Update rest mode value on Cloudant using HTTP PUT

Fig. 10. Flowchart of monitoring function implementation for a driver

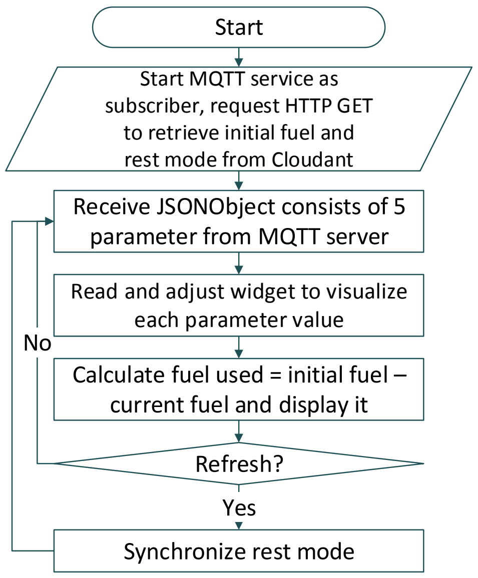

Start
Start MQTT service as subscriber, request HTTP GET to retrieve initial fuel and rest mode from Cloudant
Receive JSONObject consists of 5 parameter from MQTT server
Read and adjust widget to visualize each parameter value
No
Calculate fuel used = initial fuel – current fuel and display it
Refresh?
Yes
Synchronize rest mode

Fig. 11. Flowchart of monitoring function implementation for an owner

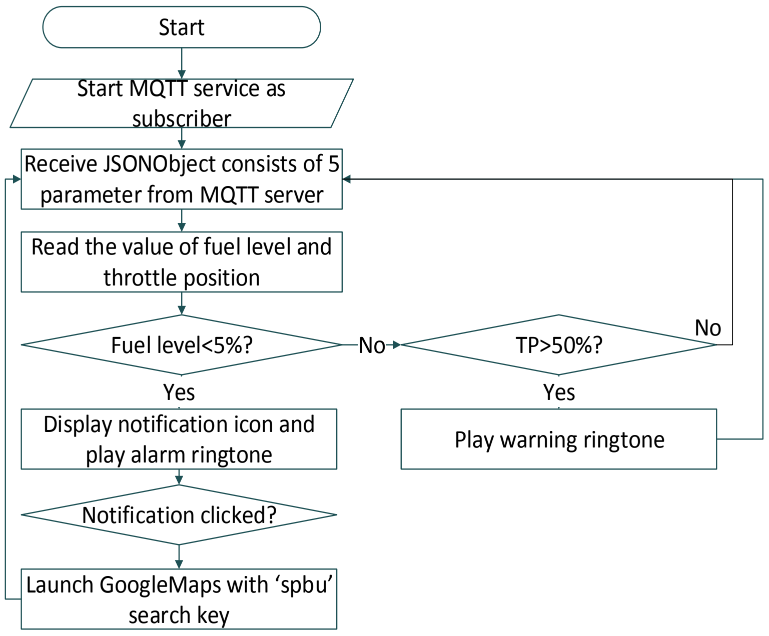

Start
Start MQTT service as subscriber
Receive JSONObject consists of 5 parameter from MQTT server
Read the value of fuel level and throttle position
Fuel level<5%?
No
TP>50%?
No
Yes
Yes
Display notification icon and play alarm ringtone
Play warning ringtone
Notification clicked?
Launch GoogleMaps with 'spbu' search key

Fig. 12. Flowchart of notification and gas station location&route function implementation for the driver

Trigger of analysis report: In this function, the VISCar application takes part in the transmission of a signal to the server that indicates the termination of the trip during log out process to trigger the processing and delivery of analysis reports in email. The flowchart in Fig. 13 describes the implementation of this function.

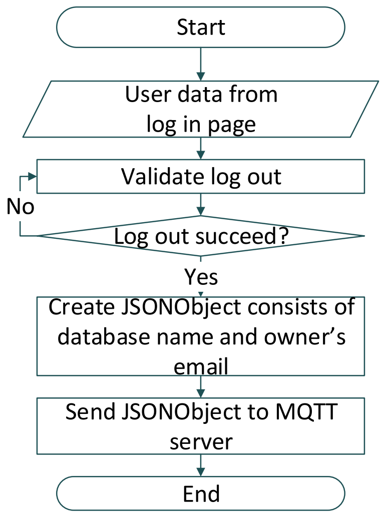

Start
User data from log in page
Validate log out
No
Log out succeed?
Yes
Create JSONObject consists of database name and owner's email
Send JSONObject to MQTT server
End

Fig. 13. Flowchart of analysis report trigger function implementation

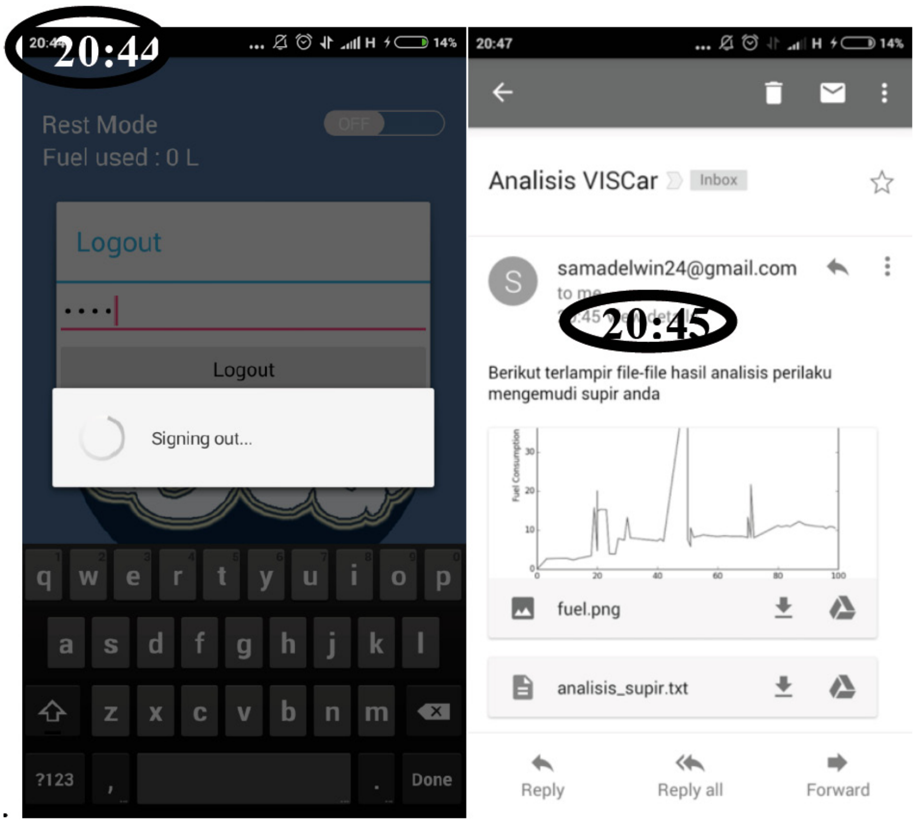
4 Testing

In this testing, The ELM327 is used. The ELM327 microcontroller functions as a bridge between the OBDII in the car and the Raspberry Pi on the VISCar system.

4.1 The Analysis of Fuel Consumption Efficiency

The testing is done by logging out from VISCar application on the smartphone and verifying the occurrence of analysis report on email that is registered by the owner, as displayed in Fig. 14.

As shown in Table II and Fig. 15, the results show that analysis feature has been successfully implemented. The analysis report email related to driving behavior is sent and received by the owner successfully. The computation, done by the program, gives the same result as the manual computation using spread sheet. Furthermore, the report represents driving behavior during the trip well, like driving with high speed too often. From Fig. 15, we can set the car speed for saving the fuel consumption.



Berikut terlampir file-file hasil analisis perilaku mengemudi supir anda

Fig. 14. The display of VISCar application (left) dan incoming analysis report email (right)

Table 2. Result of Analysis Calculation

Speed > 60	39	second
TP > 50	56	second
Speed range (km/jam)		
0-10	1.820807887	km/L
10-20	5.492061895	km/L
20-30	10.08191562	km/L
30-40	12.44308499	km/L
40-50	16.28583668	km/L
50-60	11.99969487	km/L
60-70	8.266608188	km/L
70-80	10.02761992	km/L
80-90	10.80854445	km/L
90-100	10.97734755	km/L
>= 100	0	km/L
Mileage	23	Km
Time travel	103	second
Average of fuel consumption	9.733762401	km/L
Fuel consumption	2.362909536	Liter

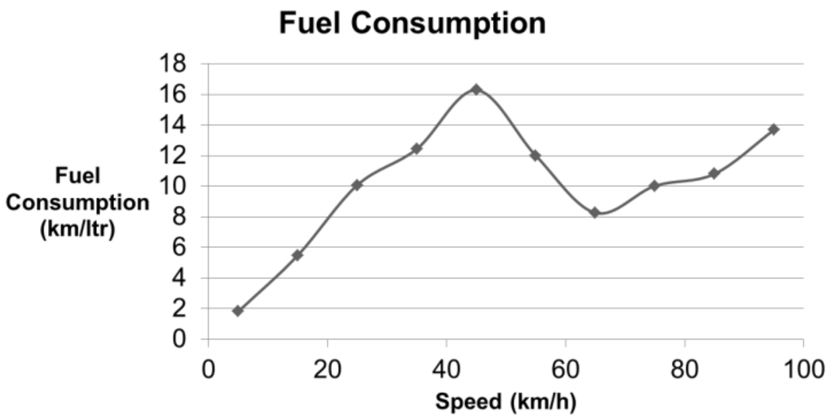


Fig. 15. Fuel consumption efficiency analysis report

4.2 Car Engine Data Monitoring

Car engine data represents the driving behavior of the driver. The testing is to verify the dataflow from car to mobile application on smart phone. It is done by observing the user interface during a trip.

The result shows that monitoring feature has been successfully implemented. Therefore, the connection among the subsystems that build the VISCar works well, start from the OBDII, Raspberry Pi, server, and mobile application.

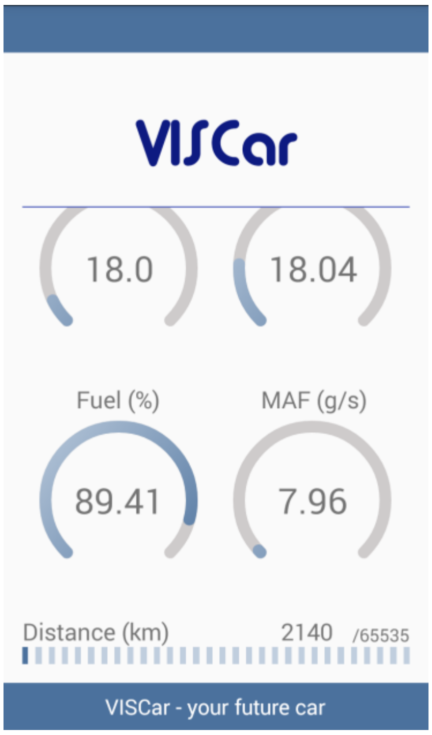
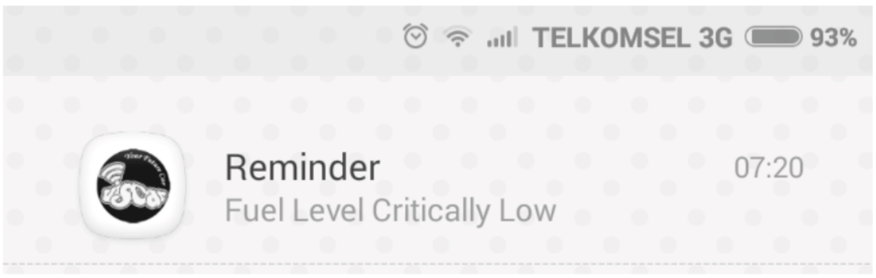


Fig. 16. The display of VISCar application while monitoring

4.3 Notification

The notification is used to warn the driver whenever he exceeds the optimum tolerance that can be seen through the throttle position and vehicle speed, also whenever the fuel level is below 5%. The notification is generated as sound and icon display. The tolerance notification testing is done by driving badly in purpose. The critical fuel level notification testing is done since the fuel level is almost critical until it reaches the critical condition, as shown in Fig. 17. In addition to the display, the sound notification can be turned on.


TELKOMSEL 3G 93%

Reminder
Fuel Level Critically Low
07:20

Fig. 17. The display of critical fuel level notification

4.4 Location and Route to Gas Station

The testing is done as the continuation of critical fuel level notification testing. The result in Fig. 18 shows that location and route to gas station feature has been successfully implemented.

5 Conclusion

The VISCar system has been successfully built and implemented to monitor fuel consumption efficiency for car rental company with its four supporting features: driving behavior analysis, monitoring, notification, also location and route to the gas station. The VISCar system is developed using OBDII standard. The fuel consumption efficiency of the car can be analyzed using four kinds of data, including car's speed, throttle position, distance, and mass air flow. From this we can set the car speed for saving the fuel consumption.

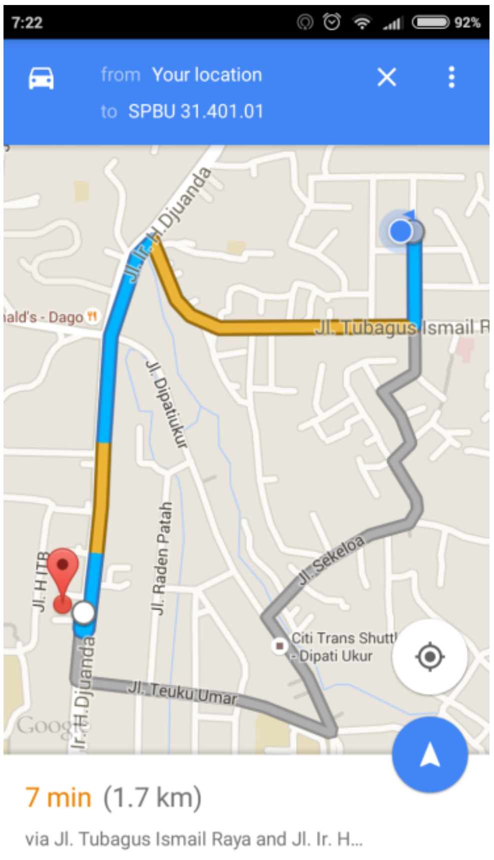


Fig. 18. The display of location and route to the nearest gas station

6 References

[1] M. Carignani, S. Ferrini, M. Petracca, M. Falcitelli, and P. Pagano, “A Prototype Bridge Between Automotive and the IoT,” *Proceedings of IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, Milan, pp. 12-17. https://doi.org/10.1109/wf-iot.2015.7389019

[2] G. Richards, “Dangerous driving? [Control Software],” *Engineering & Technology*, vol.5, pp. 42-43, 2010.

[3] Society of Automotive Engineers (SAE), *SAE J1979 Standard*, SAE Standard, 2012.

[4] W. Baoyun, “Review on internet of things,” *Journal of electronic measurement and instrument*, vol.2009, pp. 1-7, 2009.

[5] Q. Zhang, M. Almulla, and A. Boukerche, “An Improved Scheme for Key Management of RFID in Vehicular Adhoc Networks,” *IEEE Latin America Transactions*, vol.11, pp. 1286-1294, 2013. https://doi.org/10.1109/TLA.2013.6710374

[6] W. He, G. Yan, and L.D. Xu, “Developing Vehicular Data Cloud Services in the IoT Environment,” *IEEE Transactions on Industrial Informatics*, vol.10, pp. 1587-1595, 2014. https://doi.org/10.1109/TII.2014.2299233

[7] K.M. Alam, M. Saini, and A.E. Saddik, “Toward Social Internet of Vehicles: Concept, Architecture, and Applications,” *IEEE Access*, vol.3, pp. 343-357, 2015. https://doi.org/10.1109/ACCESS.2015.2416657

[8] L. Chunli, “Intelligent Transportation Based on the Internet of Things,” *Proceedings of IEEE International Conference on Consumer Electronics, Communications and Networks (CECNet)*, 2012, Yichang, pp. 360-362. https://doi.org/10.1109/cecnet.2012.6201865

[9] Gheith, R. Rajamony, P. Bohrer, K. Agarwal, M. Kistler, B.L.W. Eagle, C.A. Hambridge, J. B. Carter, and T. Kaplinger, “IBM Bluemix Mobile Cloud Services,” *IBM Journal of Research and Development*, vol.60, pp. 7:1-7:12, 2016.

[10] R. Kwiatkowski, *Monitoring Fuel Consumption on Your Vehicle in Real-Time*, Windmill Software Ltd., 2015.

[11] I. Skog and P. Händel, “Indirect Instantaneous Car-Fuel Consumption Measurements,” *IEEE Transactions on Instrumentation and Measurement*, vol.63, pp. 3190-3198, 2014. https://doi.org/10.1109/TIM.2014.2315739

[12] S.E. Li, S. Xu, G. Li, and B. Cheng, “Periodicity based cruising control of passenger cars for optimized fuel consumption,” *Proceedings of IEEE Intelligent Vehicles Symposium*, 2014, Dearborn, pp. 1097- 1102. https://doi.org/10.1109/ivs.2014.6856424

[13] S.A. Birrell, M. Fowkes, and P.A. Jennings, “Effect of Using an In-Vehicle Smart Driving Aid on Real-World Driver Performance,” *IEEE Transactions o Intelligent Transportation Systems*, vol.15, pp. 1801–1810, 2014.

[14] Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Communications Surveys & Tutorials*, vol.17 , pp. 2347-2376, 2015. https://doi.org/10.1109/COMST.2015.2444095

[15] A.A. Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, “Toward better horizontal integration among IoT services,” *IEEE Communications Magazine*, vol.53, pp. 72-79, 2015. https://doi.org/10.1109/MCOM.2015.7263375

7 Author

Emir Husni is Associate Professor at School of Electrical Engineering & Informatics at Institut Teknologi Bandung. Husni received the Ir. degree in Electrical Engineering from Institut Teknologi Bandung, the M.Sc. and Ph.D. degrees in Satellite Communication and Satellite Technology from University of Surrey, UK. Husni's research interests are in satellite technology, internet of things, and pervasive computing. (e-mail: ehusni@lskk.ee.itb.ac.id).

Article submitted 30 November 2016. Published as resubmitted by the author on 09 April 2017.