Monitoring Electric Vehicles on The Go

Monitoring Electric Vehicles on The Go

Davide Aguiari*, Ka Seng Chou†, Rita Tse†, and Giovanni Pau*‡
* Department of Computer Science and Engineering, Alma Mater Studiorum - University of Bologna, Bologna, IT
† School of Applied Sciences - Macao Polytechnic Institute, Macao
‡ UCLA Samueli Computer Science, University of California, Los Angeles, USA
Email: {p1204600, ritatse}@ipm.edu.mo; {davide.aguiari2, giovanni.pau}@unibo.it

***Abstract*—Electric vehicles (EV) feature detailed monitoring and control over the CAN bus. Some of this data is made available to users on the On-Board Diagnostic version II (OBDII) bus thus providing an opportunity for large scale high-frequency data collection. This paper introduces a connected monitoring system for OBDII equipped vehicles. The system comprises a low cost hardware design and monitoring algorithms designed to optimize the number of variables collected and their collection frequency. The algorithm aims at collecting a high quantity of Battery Management System (BMS) data in electric vehicles together with power-usage data to enable short and long term estimation for battery state of health (SOH) and state of charge (SOC). The proposed system has been implemented and tested on a Nissan Leaf and lead to the acquisition of 1.7 million records over 120 hours of driving.**

***Index Terms*—OBDII, 4G, Electric Vehicles, SOH, SOC**

I. Introduction

Modern cars and electric vehicles embed thousands of sensors to provide several functionalities and features. More and more cars manufacturers include new and useful utilities in their new models to enhance the driver's comfort and its driving experience [1]. For example, the Advanced Driver Assistance Systems (ADAS), the Anti-lock Braking System (ABS), the lane-keeping assist, or the cruise control are easy to find even in the cheapest market sector. On the other hand, new functionalities require the car to orchestrate a huge number of signals between different components called Electronic Control Units (ECUs). An ECU is an embedded system that exchanges data at low latency and high bitrate which connecting all the car's components all together or controls a specific vehicle part, such as:

- Vehicle Control Module (VCM), that acts as a gateway to control all the units on the controller area network (CAN bus);
- Li-ion battery controller (LBC), that tracks the battery pack in the EVs;
- Heating, Ventilation and Air Conditioning (HVAC), to control airflow and temperatures inside the cabin;
- ABS actuator and electric unit;
- ADAS, and many more...

Automakers constantly invest globally more than 100 billions USD into the research and development departments to enhance vehicle safety and transportation efficiency [2]. Moreover, autonomous cars constantly scan the environment with their sensors, exchanging information with other vehicles (V2V) and the infrastructure along the roads (V2I). Multiple actors are also interested in monitoring every variation and load change inside the vehicle (e.g. drivers, producers, or leasing companies), together with the possibility of orchestrate a wider smart and connected vehicles moving around creating an Internet of Vehicles (IoV)[3].

In 1996, the OBD2 [4] specification is made mandatory for all cars sold in the United States while the European Union did the same for all the gasoline vehicles (2001) and diesel (2004) sold in Europe. But it was not until 2011 that the Chinese market adopted the ISO 15765-4 signaling standard proposed by the United States.

Along with the ISO standard definition and adoption, many commercial OBD-II scanners have been sold to diagnose engine problems or general issues specified by the Diagnostic Trouble Codes (DTC). The most famous is the ELM-327 [5], a small microcontroller that communicates with the OBDII port relying on USB, Wi-Fi, or a Bluetooth connection. It normally requires a dedicated Android/iOS application to read the data. Since the communication is bidirectional, some scanners can also send CAN messages into the CAN-bus; generally, they clear the DTC warnings or they can modify some hidden settings up to forge the odometer.

However, nowadays the ECU data is encrypted for safety reasons. For example, Nissan have added the ECU gateway to filter out all the unauthorized messages. The Data monitoring is possible only by knowing the proprietary CAN message protocol. Due to this policy, most of the commercial OBDII microcontrollers are stopping working. In this paper, we propose a system that consists of a 4G LTE data logger which helped us to retrieve several variables from a Nissan Leaf 2018 Acenta 40kWh despite its OBDII CAN-bus was controlled but the aforementioned gateway.

The paper is organized as follows: section II is an overview of the data logger, the communication to the server along with its architecture; section III focuses on the implementation of the microcontroller firmware, and the reverse-engineered CAN protocol explanation; section IV describes the platform validation running on a Nissan Leaf 2018; finally, some future enhancements are provided in the final section.

II. System architecture

The design of an appropriate monitoring system is crucial to correlate and study all the signals and measures while the car is moving. The architecture is the sum of three main components:

1) The LilyGo TTGO T-SIM7600E-H 4G LTE [6], a standalone IoT device for ECU sensing and data validation shown in the figure 1.
2) A headless Amazon EC2 instance for data aggregation from multiple vehicles;
3) An Amazon Aurora MySQL Database for data storage and data analysis.

The LilyGo TTGO T-SIM7600E-H produced by Shenzhen Xin Yuan Electronic Technology Co., Ltd is a modern wireless networking module that relies on the ESP32 WROVER-B microcontroller unit (MCU); the clock speed is 240Mhz along with a 4MB flash memory (extended with an external microSD card), and an 8MB pseudo SRAM; by design, it also embeds an Assisted-GPS unit inside that supports GPS and GLONASS, a slot for the microSD card, and a slot for the SIM card. Its slim size (110*30*29 mm) is suitable for a compact allocation under the steering wheel without interfering with the driver usage; the LilyGO is also connected to an MCP2515 SPI/CAN controller that reads and writes the CAN message to the ISO 15765-4 OBD2 HIGH (6) and LOW (14) pins; finally, a DS3231 real-time clock (RTC) is used for the fine-time granularity CAN messages requests. The CAN-bus transmission rate can reach up to 1Mbps exchanging a huge number of 8 bytes messages. The board is powered with

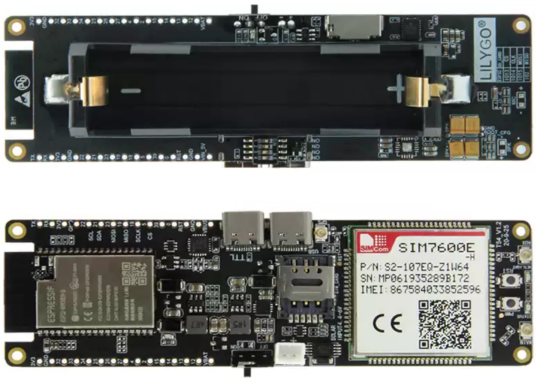

Fig. 1. LilyGo TTGO T-SIM7600E-H 4G LTE CAT4 USB dongle [6]

the OBDII pin number 16 which has constant +12V DC power from the battery pack; the pin is protected by a 10A fuse. Since the LilyGo operates with 3.4V-5V, we added a 12V-5V step-down converter between the OBDII port and the board.

Once the SIM7600E-H networking module has established a 4G cellular connection with the cell tower, it gets an IP address and it starts sending all the variables to a cloud server. The server is hosted into an Amazon AWS EC2 t2.micro instance based in Frankfurt. The instance runs on a Linux Ubuntu 14.04.6 LTS with a 3.13.0-170-generic Linux kernel. The t2.micro machine has one vCPU and 1GiB RAM, with 32GiB SSD. The database is distributed among two Amazon Aurora instances which formed a regional cluster based in Frankfurt. Two db.t2.small instances offer 1 vCPU core and 2GiB RAM, with low-medium throughput. One reader and one writer instance work together to prevent DDOS attacks and, in case of failures, they can be restored with fall-back recovery. Here our MySQL database runs on 5.6.mysql aurora.1.22.2 engine.

Many OBDII dataloggers suffer from a power usage drawback: they constantly drain the electric energy from the internal 12V battery [7]. Therefore, we added a battery controller between the OBDII power pin and the LilyGo, to program its switching on and off as the figure 2 shows.

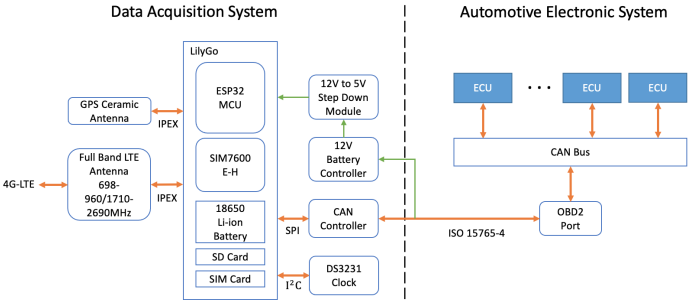

Fig. 2. CAN Data Acquisition Architecture

III. Implementation

The University of Bologna has at its disposal a fleet of 30 electric Nissan Leaf 2018 Acenta. The goal of this project was to retrieve all the High-Voltage battery and engine information from the different ECUs, monitoring all the fleet. Despite the OBDII parameter IDs (PIDs) are well defined by SAE standard J1979 [8], we could not fetch the data requested out of the box, because of the CAN gateway behind the OBD2 port. This is why we adopted a programmable board instead of exploiting the commercial OBDII scanners on the market. With a custom board and a CAN controller we could (i) exploit the SIM7600E-H for the 4G Internet connection, (ii) set custom CAN module settings to match with Nissan's CAN-bus, and (iii) we could send custom CAN requests instead of the hardcoded ones into other scanners. Although a custom OBD-II bluetooth device can be used to scan the car's CANbus, we preferred to avoid the driver's active involvement in data logging; this ensures that no user will alter the data and it will no be distracted by the use of the smartphone.

We programmed the LilyGO's ESP32 using Arduino IDE, adding the 'ESP32 Dev Module' capabilities with the 'Board Manager' and following the build instructions provided by the LilyGO's developers. Firstly, we use the LilyGO's API and TinyGSM library to interact with the SIM module; the library tries to establish a connection to the Internet Service Provider (ISP) using the AT commands. Once the module obtains a valid IP address, we synchronize the external DS3231 RTC to get the right time from the NTP local server; next, we take advantage of the LilyGO's two cores, exploiting ESP32 FreeRTOS to separate the workflow into multiple threads (xTasks):

- One task (CheckConnection) cyclically checks the cellular signal quality (in dBm), the GPRS status along the connection availability. While moving, the connection is

intermittent sometimes but it can be reestablished once the module lost the IP address;
- a GPS task (checkGPS) interacts with the SIM7600's multiple satellites high accuracy positioning GNSS system, to get the current latitude, longitude, and altitude;
- a CAN reader task (checkCAN) cyclically sends the CAN requests to get access to all the ECU's variables described in section IV. In case the connection is missing, the task stores the variables into the 32GB microSD for a delayed re-transmission.

A UDP socket establishes the connecting endpoints between the board and our Amazon EC2 thanks to some adjustments we made on the TinyGSM library. We edited TinyGsmClientSIM7600.h file in which particular modemConnect(), modemSend(), and modemRead() to comply with the right AT commands CIPOPEN, CIPSEND, and CIPRXGET according to the 'SIM7600 Series TCPIP AT Command Manual V1.0' by the reason of TinyGSM has not implemented the UDP context yet but the SIM7600 actually supporting UDP while we are writing this. Due to some cost constraints with the ISP provider and the intermittent nature of the V2I connectivity, we decided to send UDP packets to the cloud server, avoiding any re-transmissions in case of failures. Then we tested the absence of missing packets from the server perspective, even with high-speed tests on the road. The message protocol comprises two kinds of string requests and responses as follows:

- {SEND_CODE, VEHICLE_ID, DATETIME, GPS_COORDINATES, [VARIABLES]}
- {ACK, VEHICLE_ID, DATETIME, [ERROR]}

'SEND_CODE' can take only two different values: (0) if we want to add a new vehicle into the database, or (1) if we want to send a new packet that contains all the variables parsed from the CAN-bus. If the 'SEND_CODE' is zero, 'VARIABLES' must be empty. The 'VEHICLE_ID' is the car's identification number composed of 17 characters (digits and capital letters), also known as VIN; the VIN serves as the car's fingerprint, as no two vehicles in operation have the same id. 'DATETIME' is the timestamp of the current cycle of the variables stored in the payload; as timestamp, we use the epoch (i.e. how many seconds have passed since January 1, 1970 00:00:00 UTC). The 'GPS_COORDINATES' are the longitude, latitude, and altitude mentioned above, while 'VARIABLES' is a list of all the ECU's parameters we asked to the CAN gateway during a cycle. The couple 'VEHICLE_ID' and 'DATETIME' is used as MySQL key in the data table. Before sending a value, a conformity check is applied: all the variables out of their operative range are discharged.

On the server-side, a UDP socket is listening to the port 55555 for new incoming packets. The new sample is then parsed to check its correct format and integrity, and finally stored into a dedicated MySQL table. Therefore, a thorough modeling of the time series is possible in order to predict the battery's SOH in correlation with its temperature and voltage. The server replies with an acknowledge response with 'K' as 'ACK' in case the packet has been correctly received or 'E' that stands for error, with a detailed description in 'ERROR' (e.g. a badly formatted packet from LilyGO, or a duplicate sample).

IV. Evaluation

To evaluate the system, we drove more than 120 hours throughout the road of the city of Bologna, during the last months. As stated in the previous sections, we used one of the Nissan Leaf 2018 Acenta from the University of Bologna fleet. In order to overcome the 'gateway problem', our first step was to study how CAN parsing was possible on previous Nissan Leaf models. In fact, before 2018 all the ECUs broadcasted all the CAN messages into the bus passively. Lots of variables were found by enthusiasts and researchers, correlating the values changes among different driving scenarios. Many tools (e.g. PEAK PCAN-View [9]) highlight which bytes, with the same CAN ID, have changed during the scans.

Following a similar approach, we identified all the ECU ids (i.e. CAN message ids) that the gateway allowed to be questioned, i.e. we flooded the bus with all the combinations of 8 bytes with 0x7XX IDs until we got a CAN message which does not match with CAN error (0x7XX 03 7F 21 11 FF FF FF FF). The Nissan Leaf uses the 11-bit CAN frame exclusively, so the identifiers could range from 0 to 0x7FF; not all the identifiers can be queried, i.e. they neither reply with an error message nor any message at all.

TABLE I
Nissan Leaf 2018 ECU IDs

ECU	ID Query	ID Response
Vehicle Control Module (VCM)	0x797	0x79A
Body Control Module (BCM)	0x743	0x763
AntiBlockierSystem (ABS)	0x740	0x760
Li-ion Battery Controller (LBC)	0x79B	0x7BB
Traction Motor Inverter (INV/MC)	0x784	0x78C
Meter	0x745	0x765
HVAC	0x744	0x764

Every ECU has a predefined id and it replies with a different, but consistent id accordingly. Nissan Leaf's ECU IDs are reported in above table I. In the payload, the unused bytes are usually 0xFF. To comply with the Nissan protocol, the message query must also follow these rules:

- The first byte corresponds to the length of the payload (usually 02 or 03), e.g. not FF bytes.
- The second byte is usually 0x21 for a multi message response or 0x22 for a single message response;
- The third and fourth bytes are the PIDs that we reverse-engineered.

For instance, this is how we ask for the car's gear:

R: 0x797 03 22 11 56 FF FF FF FF

A: 0x79A 04 62 11 56 01 FF FF FF

Thus, 0x79A is the VCM's answer message ID, 04 is the payload byte length, 62 corresponds to 22+40 (it is always the third question byte plus 40), 11 and 56 is the gear PID; 01 is the gear value, i.e. 1=Park, 2=Reverse, 3=Neutral, 4=Drive.

In some cases, the ECUs reply with many messages to a single request; if it happens, the first byte correspond to the message ordering. It is possible to request the whole

response once we get the first 8 byte response; we usually wait at least 0.5ms and then we send a new query as follows:

0x7XX 30 00 00 00 00 00 00 00

Multiple next readings of the CAN buffer are required to get all the answers. All the PIDs we found, and related variables, are reported in the tables II.

TABLE II
NISSAN LEAF 2018 PIDS

Variable	PID	ECU
Tires Pressure	0E 25-28	BCM
RPM	12 55	BCM
Torque	11 15	INV
Wipers-Lights	09	Meter
Speed	12 1A	VCM
Motor Power	11 46	VCM
Range	0E 2E	BCM
Brake pedal	12 09	ABS
Acc. pedal	11 15	ABS
12V Bat Current	11 83	VCM
12V Bat Voltage	11 03	VCM
Odometer	0E 01	BCM
Temperature	11 5D	VCM
AC Power	12 61	VCM
Quick Charges	12 03	VCM
L1 L2 Charges	12 05	VCM
SOH	61	LBC
HV Battery Temperature	04	LBC
Battery Serial	84	LBC

Variable	PID	ECU
ePedal	13 1A	VCM
Heater Power	12 62	VCM
AUX Power	11 52	VCM
Motor Temp	11 21	INV
OBC Power	12 36	VCM
ECO Mode	13 1B	VCM
AC	11 06	VCM
Gear	11 56	VCM
Plug State	12 34	VCM
Charging Mode	11 4E	VCM
12V Bat Voltage	11 03	VCM
Rear Heater	11 0F	VCM
Fan speed	10	HVAC
Power (ON/OFF)	13 04	VCM
VIN	81	VCM
HV Battery Current	01	LBC
HV Battery Voltage	01	LBC
SOC	01	LBC
HV Cells Voltage	02	LBC
HV Battery Capacity	01	LBC

V. CONCLUSION

This paper presents a ESP32-based 4G OBDII platform for vehicular data acquisition which requires a SIM card only to work. The system can be placed under the steering wheel and powered directly from the OBDII port. Nevertheless, to the best of our knowledge, this is the first OBDII scanning attempt via CAN bus reverse engineering, since Nissan does not publish its CAN message protocol. As the OBDII standard has become mandatory in USA, Europe, and Asia, along with the global rise of the electric vehicles, self-diagnosis systems and vehicular data acquisition systems are getting much attention from automakers, researchers and enthusiasts. Despite the SAE standard J1979 defines many OBDII PIDs, many vehicles manufacturers are hiding them behind more robust CAN-bus gateways and proprietary CAN message protocols. Furthermore, almost all [10]–[12] the OBDII scanners on the market relies on dedicated mobile apps and they communicate real-time via bluetooth, USB, or via WiFi.

Ultimately, extensive evaluation carried out show how fine-time BMS monitoring represents a promising enabling technology for the battery's health estimation. Real-time data along with a thorough analysis of collected samples will give us valuable insights into the battery pack degradation and driver's behaviors.

ACKNOWLEDGEMENTS

This work was supported in part by the Macao Polytechnic Institute – Edge Sensing and Computing: Enabling Human-centric (Sustainable) Smart Cities (RP/ESCA-01/2020). This work has received funding from the LiBER project under PORFESR programme by Emilia Romagna Region, years 2019-2021.

REFERENCES

[1] A. K. Basu, S. Tatiya, and S. Bhattacharya, "Overview of electric vehicles (evs) and ev sensors," in *Sensors for Automotive and Aerospace Applications*, S. Bhattacharya, A. K. Agarwal, O. Prakash, and S. Singh, Eds. 2019.

[2] "Automotive r&d," *2019 Global R&D Funding Forecast*, vol. Winter 2019, p. 17, Oct. 2019.

[3] A. Bujari, O. Gaggi, C. E. Palazzi, and D. Ronzani, "Would current ad-hoc routing protocols be adequate for the internet of vehicles? a comparative study," *IEEE Internet of Things Journal*, vol. 5, pp. 3683–3691, 2018.

[4] *United states environment protection agency (epc). vehicle emissions on-board diagnostics (obd).* [Online]. Available: https://www.epa.gov/state-and-local-transportation/vehicle-emissions-board-diagnostics-obd.

[5] *Elm327 - elm electronics, "on board diagnostics (obd) ics*. [Online]. Available: https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf.

[6] *Lilygo ttgo t-sim7000g module*. [Online]. Available: https://github.com/Xinyuan-LilyGO/LilyGO-T-SIM7600X.

[7] *How to solve elm327 drain power problem?* [Online]. Available: http://blog.obdii365.com/2014/12/08/how-to-solve-elm327-drain-power-problem/.

[8] SAE. (2017). "SAE J1979/ISO 15031-5: E/E Diagnostic Test Modes," [Online]. Available: https://www.sae.org/standards/content/j1979_201702/.

[9] PEAK. (2021). "PCAN-View: Windows Software for Displaying CAN and CAN FD Messages," [Online]. Available: https://www.peak-system.com/PCAN-View.242.0.html?&L=1.

[10] J. Palomino, E. Cuty, and A. Huanachin, "Development of a can bus datalogger for recording sensor data from an internal combustion ecu," in *2021 IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics*, 2021.

[11] M. Amarasinghe, S. Kottegoda, A. L. Arachchi, S. Muramudalige, H. M. N. Dilum Bandara, and A. Azeez, "Cloud-based driver monitoring and vehicle diagnostic with obd2 telematics," in *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, 2015, pp. 243–249.

[12] J. Moniaga, S. Manalu, D. Hadipurnawan, and F. Sahidi, "Diagnostics vehicle's condition using obd-ii and raspberry pi technology: Study literature," *Journal of Physics: Conference Series*, vol. 978,