Project development review

Digital Cabin (D-Cab) development for the ITESM Electrobus Initiative

Guillermo Sáenz-González [1], Diego A. Santisteban-Pozas [2,*], and Diego M. Botín-Sanabria [3]

[1] Tecnológico de Monterrey; A00823049@itesm.mx
[2] Tecnológico de Monterrey; A01154423@itesm.mx
[2] Tecnológico de Monterrey; A00825786@itesm.mx
* Correspondence: A01154423@exatec.tec.mx

Abstract: This paper covers the development of a digital dashboard prototype for the Electrobus cabin, that includes a circuit that simulates communication between a vehicle and the system through a real CAN network, and an Android Automotive OS based interface capable of displaying received information on a basic GUI. The circuit was built using an Arduino UNO running a simple program to emulate a vehicle ECU sending messages, two CAN controllers, and a Raspberry Pi functioning as the brain of the operation. The graphic user interface was developed in Android Studio as an android application, capable of connecting to the underlying car service through the AAOS specific Car API. Also, the installation of AAOS on a Raspberry Pi device was possible by booting through a USB flash drive instead of the most common method using an SD card. Communication between simulated ECU and RPi was achieved, and custom messages could be received and decoded correctly. The display of one type of sensor on the interface was achieved, and its necessary permission and connection operations were identified. Connection with the Car API, development of the interface, and testing of the connection's integrity were achieved, while a connection between layers, and field implementation and testing will be left for future work.

Keywords: CAN; Control Area Network; AAOS; Android Automotive OS; Electrobus; Android API; Raspberry Pi; Digital Dashboard; Cabin

1. Introduction

The Electrobus initiative was created with the aim of providing a sustainable solution to urban mobility. Its objective is to link students with five different research and industry companies to design and deliver a fully electric and connected city bus for the Circuito Tec transport. The Digital Cab is one of six projects that emerged from this initiative; with the support of the company Questum, it focuses on developing a fully digital instrumentation panel for the driver, as well as allow for the implementation of other smart solutions, such as a Digital Twin simulation. As the Electrobus is a complex project that involves many stakeholders, it is expected to be completed in the second semester of 2022. However, as of now, the associated companies and the technical teams in Tecnológico de Monterrey are working towards developing the systems and having them ready to be implemented.

Literature on this subject is sparse, but not inexistent. Last year, Volvo released its Polestar 2, the first electric car that implemented an infotainment and dashboard system natively based on Android Automotive OS (AAOS) [5], and it is one of the very few examples of an application of this exact technology. Apart from this, there are some developments that approach this solution, although most of them without the use of AAOS. Systems developed by Varga, A. [1] and Morgan, R. [2] served as a basis for the

development of the circuit, and the image utilized for the prototype was provided by Snapp Automotive, which are pioneers in the development of automotive infotainment systems.

Section 2 of the paper begins with a simple review of CAN bus networks. It then covers the development of the prototype, the design of its circuit and the instrumentation involved, as well as a look into the hardware and software specifications and connectivity tests. Section 3 covers the development of the interface (app), software connection, GUI prototype, and a look into the testing environment, as well as how the Android Automotive Operating System was implemented in the hardware. After this, in the discussion section, some of the literature for both the hardware and software is reviewed and compared to the developments of this project. At the end, the main conclusions of this project and future work are stated as bullet points.

2. Circuit design and instrumentation [Diego]

2.1 Controller Area Network (CAN bus)

The Controller Area Network, or CAN for short, is used to ensure reliable communication between vehicle systems. In its most basic setting, the CAN protocol uses a pair of wires, named CAN-High and CAN-Low, and serialized electrical signals to transport all information needed. Systems can send their status and other useful information through this network, and these signals can be read by any other system or device connected to it. One example of such systems is the electronic dashboard that displays basic information to the driver, such as vehicle speed, RPM, fuel/battery gauge, and distance travelled. This dashboard is connected via CAN protocol to the rest of the vehicle, through which the other systems provide the information the dashboard needs.

2.2 Circuit design (prototyping)

These characteristics are also present in the development of the Electrobus, the only exception being that, as for the requirements of this project, its dashboard will be replaced by a fully digital TFT display connected to a Raspberry Pi running Android Automotive. This dashboard will work and display the exact same information as a standard one and will also be connected to the vehicle network through a CAN controller, but it will have the added flexibility granted by running an image of Android. Due to the unavailability of a running vehicle for tests, all testing of our system had to be carried out using a simulated CAN network. This network used an Arduino UNO board to emulate messages just as if they were sent by the vehicle. It did so by running code that periodically sent a formatted custom message to a CAN controller board (MCP2515 controller with a TJA1050 transceiver), which in turn sent it to an identical CAN controller board through a twisted pair of wires (CAN-High and CAN-Low). The second controller was then connected to a Raspberry Pi 4, which ran a python program able to decode the message and display it. The schematic of this circuit is shown in Figure 1., and a diagram showing the connections and relationship between components is shown in Figure 2.

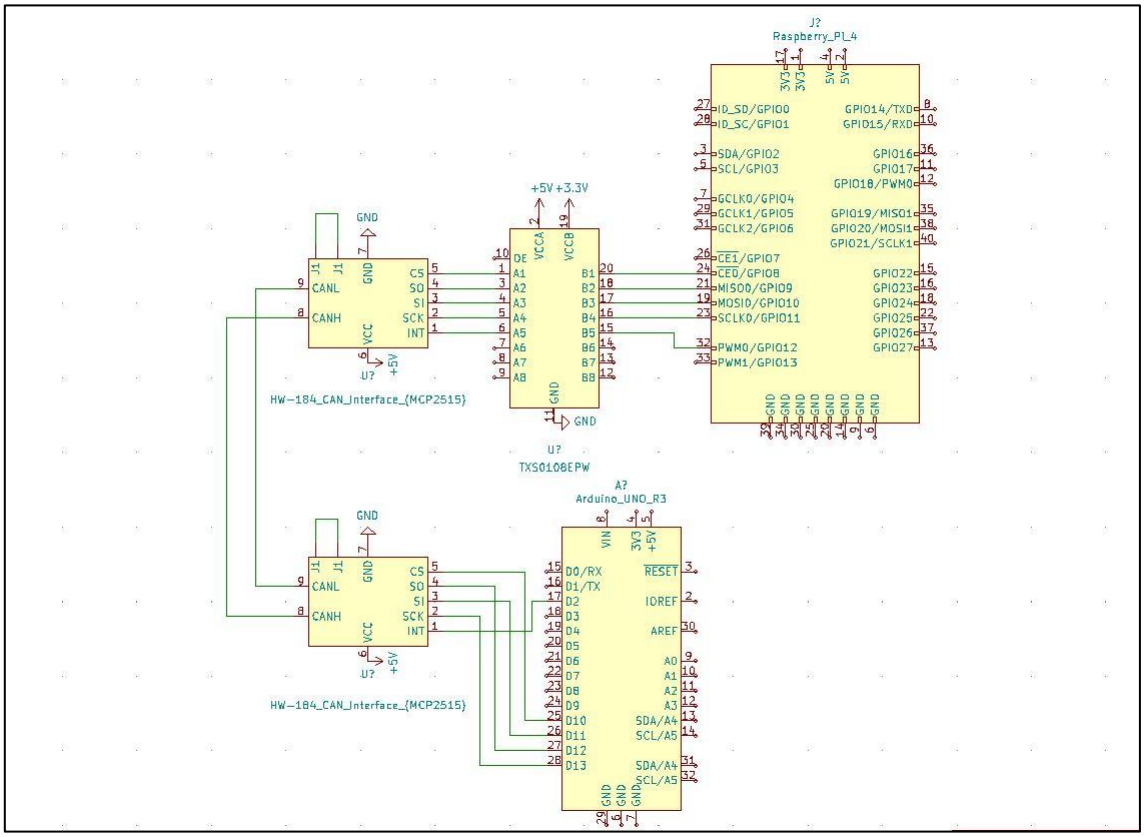

Figure 1. Electronic schematic of test circuit (made with KiCAD)

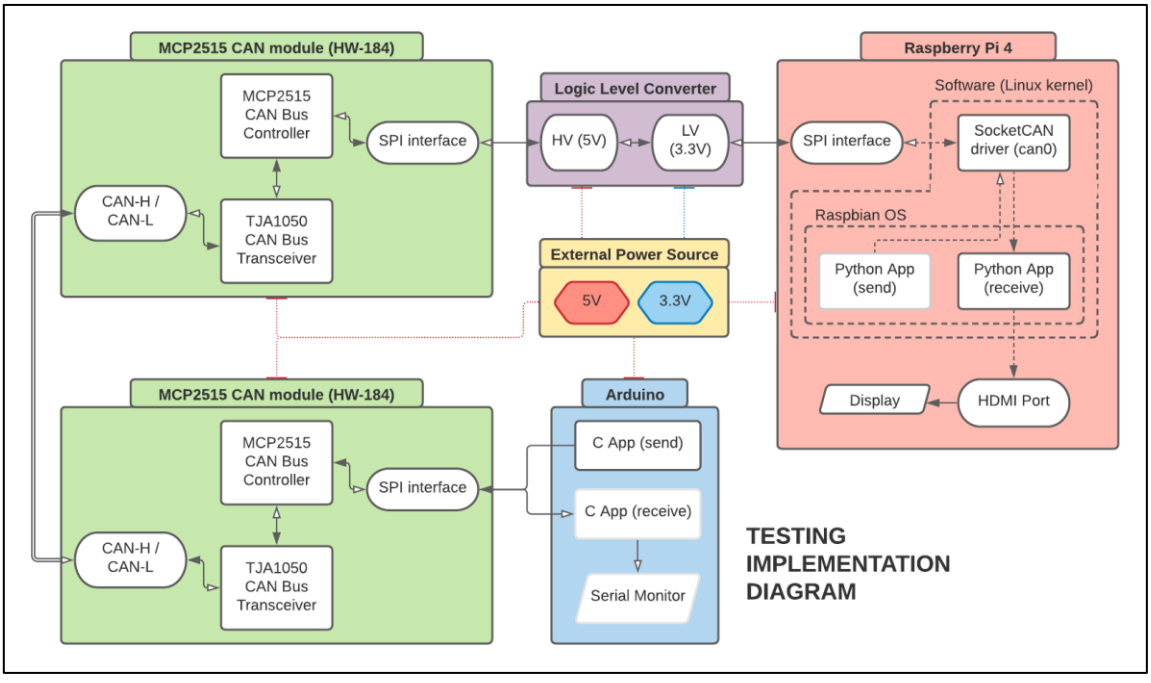

Figure 2. Connection diagram for prototype

Several components were used to build this test circuit, so a bill of materials (Table 1.) was created to keep track of all materials required to replicate this project.

Table 1. Bill of Materials of prototype

Elemento	Descripción	Cant.
Protoboard	Board para prototipado de circuitos	1
Convertidor de Nivel Lógico (Bi-direccional)	Convertidor Logico Nivel 8 Canales 5v a 3.3v Txs0108e.	1
Modulo CAN MCP2515	Modulo Can Bus Spi Mcp2515 Tja1050 51 Mcu Arm	2
Jumpers macho/macho/hembra	120 Piezas Cables Jumpers Dupont H-h, M-m, H-m 10cm Arduino	1
Pantalla	Pantalla LCD 10" para integración con Raspberry Pi	1
Arduino UNO	Microcontrolador programable Arduino UNO	1
Modulo fuente de 5V/3.3V	Modulo Fuente Para Protoboard Mb102 Arduino 3.3V/5V	1
Eliminador Fuente de Poder	Eliminador Fuente Poder 5v 2a Arduino	1
Raspberry Pi 4 B	Microcomputadora programable Raspberry Pi 4 B	1
Cable HDMI Micro/Standard	Cable para HDMI enviar imagen, debe ser de HDMI Micro a Standard	1
Memoria USB 3.0	Memoria flash drive USB (de preferencia 3.0, por velocidad de lectura)	1

2.3 Connectivity testing

The code used in Arduino was based on the "mcp_can" and "SPI" libraries, and the can-utils (socketCAN) packet was installed in the Raspberry Pi to help with the decoding of the signals. A custom device tree overlay was loaded in the RPi configuration to help drive the MCP2515 controller, and a channel (can0) was opened within the RPi terminal to allow applications to read and write messages from and to the controller. It was through this system, in conjunction with an external monitor and a computer, that most connectivity tests were carried out.

```
Code: Select all

dtparam=spi=on
dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=12
dtoverlay=spi-bcm2835-overlay


sudo apt-get install can-utils


sudo ip link set can0 up type can bitrate 500000
```

Figure 3. Commands for configuration of com-channel in RPi (Raspbian OS)

Results obtained with the test setup were positive overall. A stable connection between transmitter (Arduino) and receiver (Raspberry Pi) was successful, and CAN

messages were correctly coded, sent through the twisted pair, decoded, and displayed. This proved the integrity of the signal between controllers and the capacity of the circuit to relay emulated messages. Due to the closeness between our test circuit and actual CAN system, adapting it to carry out a real application shouldn't pose much of a problem.

Message	Signal	CAN ID	Startbit	Length	Endian
Motor_Status	Motor_Speed	0x205	40	16	Motorola

		ID		Motor_Power [U]		Drive_Current [S]		Motor_Speed [S]		Motor_Torque [S]	
Motor_Status	MIN	02	05	00	00	00	00	00	00	00	00
	MAX	02	05	00	00	00	00	00	00	00	00

Figure 4. Example of CAN message frame format (frame data provided by QUESTUM)

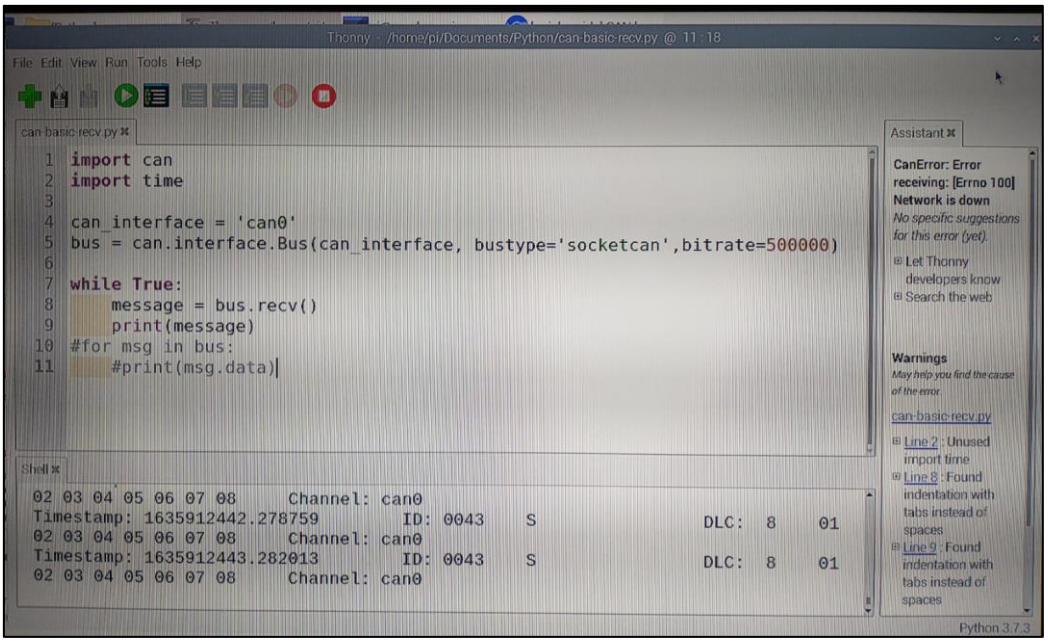

Figure 5. Screenshot of RPi receiver program reading messages sent via CAN bus.

Another successful result was obtained after testing CAN messages that constantly changed values. The Raspberry Pi was able to read and display a message with a certain ID and a shifting signal value corresponding to a change in motor speed. This is important for the next step in the building of the final prototype, which is using the decoded signals to move the simulated digital gauge of the display, thus presenting this information to the driver.

3. Programming and interface [Guillermo]

3.1 Interface (app)

The most convenient way of implementing an interface is through an Android Application. To communicate with the vehicles systems, the first step is to establish a connection between the application and the AAOS car service. This service is the one meant to connect to and interpret the car network messages (flowing through the CAN bus). In terms of connectivity, the International Organization for Standardization (ISO) has proposed a six-layer model. The Car API (application programming interface) is the one that allows the connection between CAN bus and AAOS service from the application layer.

Taking this into consideration, the prototype application was developed in Android Studio. It can retrieve the speed value from de Car Service API and then display it on the Android Studio debug log, as well as in the graphical element. Said value was controlled through the Android Studio simulated sensors, and the application was executed on the Android Studio emulator, through a Virtual Device which simulates the capabilities of a real AAOS device. The testing environment and prototype application can be seen in Figure 5.

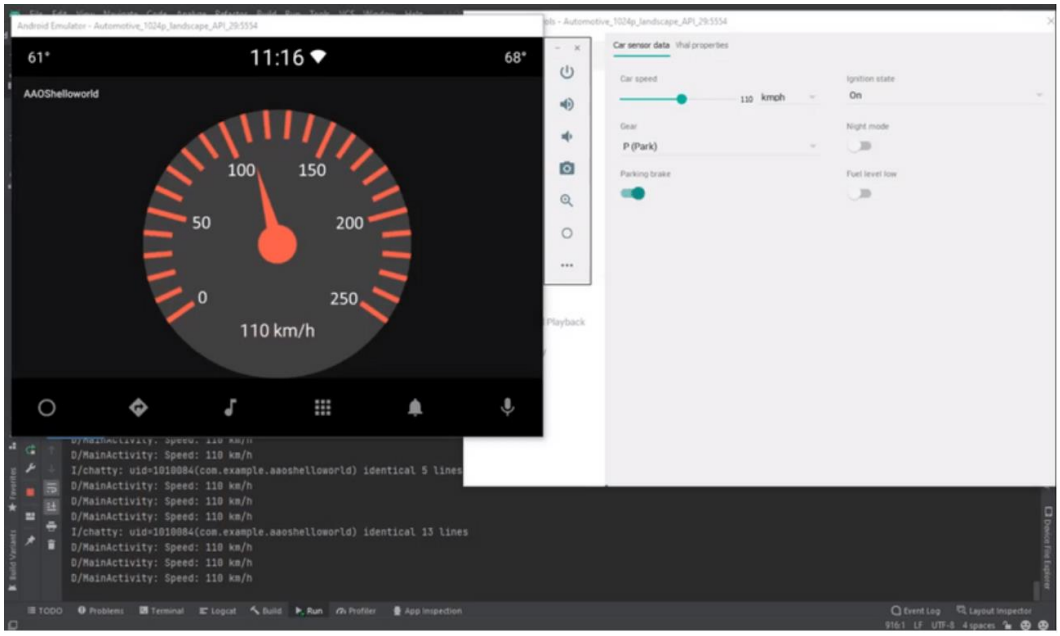

Figure 6. Testing environment of the interface.

3.2 Android with raspberry PI.

The vehicle cabin display is intended to run on the Android Automotive OS (AAOS). This is a variation of Google´s Android, designed to run on vehicle dashboards. Unlike the previously launched Android Auto, AAOS is a full operative system meant to run on the vehicle's device and interact with its subsystems. A benefit of implementing AAOS is that it is open source which means that there are no fees involved with its use and distribution. But most importantly, it can be customized to align with a company's style and made to include the company's applications by default. This allows significant customization and allows developers to work easily with the platform.

The goal of this project is to run said operative system on the RPi and, enabled by this environment, the digital cabin display. The installation on the RPi is possible by leveraging existing open-source projects dedicated to port Android to different devices. One such example is the "android-rpi" project, in conjunction with the modifications that the Snapp Automotive company provides for AAOS. The latter was used to install android in the raspberry pi. By burning a pre-built image of AAOS (distributed by Snapp) onto a USB, and then booting the raspberry pi from said USB, it was possible to run android from this device. It was discovered that USB 3.0 is the type of flash drive that works best when attempting this.

The physical device connections can be seen on Figure 7, while Figure 8 displays pictures of the android system running on the device.

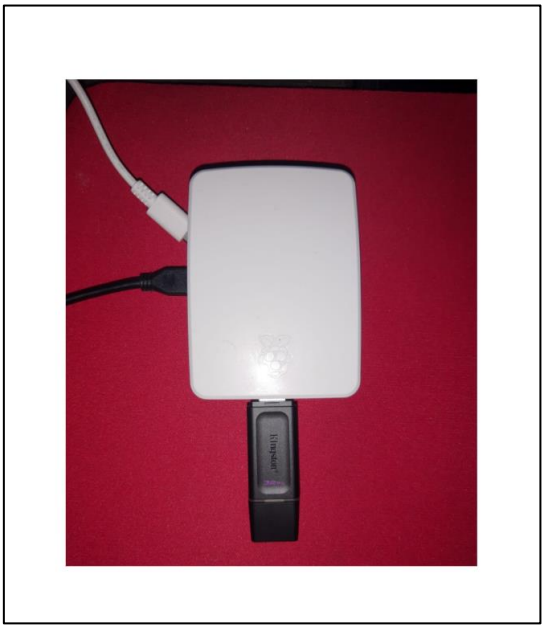

Figure 7. Raspberry Pi connections.

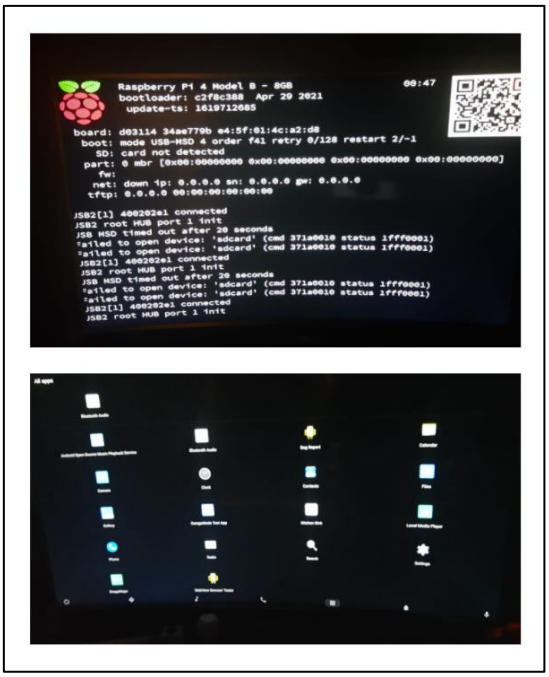

Figure 8. Android running on Raspberry Pi

4. Discussion

Most of the circuit was developed based on several tutorials found on the YouTube platform and the Raspberry Pi Forums, but there are some key differences with these circuits and the one built for this project. One of these differences is the use of Logic Level Converters that convert signals based on 5V to 3.3V, to avoid damaging the Raspberry Pi GPIOs. This solution bypasses the need for the additional component soldering step found in saper_2's guide [3] and simplifies the build of the circuit. Another difference from Varga's [1] and Morgan's [2] builds is the implementation of a fully simulated CAN bus network that uses true-to-application values. These values were provided by the involved company (QUESTUM), and allowed us to develop a much more realistic, and more

easily adaptable prototype. The simplicity of the programs and the documentation created also facilitates the continuation of this project in the future.

The Android for Developers website contains a lot of information on how to develop applications for the Android Automotive OS. While it contains all of the necessary documentation to use all of the APIs available, it only presents tutorials for media, entertainment and navigation applications. A proper tutorial on accessing the CAN services was found in the blog of siili_auto, but it was written in the kotlin language, with no graphical elements, only debug messages. It is a popular language for applications, but not as well spread as java, nor are there nearly as many tutorials for other aspects of app development. That's why the whole interface was developed in Java instead. Including the graphical elements of the application. That's how we differed from the Android for Developers and Siili_Auto documentation and examples.

On the other hand, the Snapp Automotive image for raspberry pi was used with no modifications, because we wanted to test the possibility of installing android on this development board, not customize it yet.

5. Conclusions

Project Achievements:

- Development of initial interface
- Successful installation of android automotive on Raspberry Pi
- Stable connection between Arduino and Raspberry Pi
- Successful communication through a CAN network
- Creation of simple programs that facilitate send/receive communication
- Positive results after realistically testing the hardware (circuit) and software

Future work:

- Installation of application on Raspberry's android
- Connection between Raspberry's ports and Android Car Service API
- Development of a more robust GUI
- Addition of a use-ready display to the circuit
- Increase robustness of circuit by adding better components, such as a PiCAN board
- Full integration of the circuit to the vehicle (for power, and communication)
- Run final tests of implementation

Acknowledgments: For the development and implementation efforts, a total of 5 associated companies are involved. Tecnológico de Monterrey and this work's authors are currently collaborating with an automotive bus manufacturer, a bodywork manufacturer, a public transport agency, and two EV conversion companies. Special thanks to Questum, for their guidance and frame data provided.

Conflicts of Interest: The authors declare no conflict of interest

Appendix A

Google drive folder: https://drive.google.com/drive/u/1/folders/1457YAOmm8JtustHb3iGiR5jwexg7Kvpv

References [APA]

[1] Adam Varga. (2020, March 19). How to hack your car | Part 1 - The basics of the CAN bus [Video]. YouTube. https://youtu.be/cAAzXM5vsi0
[2] Rhys Morgan. (2019, December 9). Canbus hacking guide part 1: How to set up pican2 and log or decode canbus messages [Video]. YouTube. https://youtu.be/XK1qhMWP3-g
[3] saper_2. (2014, August 3). [quick-guide] CAN bus on raspberry pi with MCP2515. Raspberry Pi Forums. https://forums.raspberrypi.com/viewtopic.php?t=141052
[4] Sundar19. (2020, December 25). [quick-guide] CAN bus on raspberry pi with MCP2515 and Arduino. Raspberry Pi Forums. https://forums.raspberrypi.com/viewtopic.php?t=296117
[5] Hawkins, A. (2020, September 4). Driving the Polestar 2, the first electric car with a brain by Google. The Verge. https://www.theverge.com/21365032/polestar-2-hands-on-first-drive-electric-android-automotive-photos-range
[6] Android (N. A.). Automotive. Android Open Source Project Documentation. https://source.android.com/devices/automotive
[7] Siili_Auto (2019, May 14). First Steps With Android Automotive Car API. Siili_Auto Blog. https://auto.siili.com/blog/first-steps-with-android-automotive-car-api
[8] Android (N. A.) android.car. Android for Developers Reference Documentation. https://developer.android.com/reference/android/car/package-summary
[9] Sutton, A. (2020, May 7). Android Automotive OS 11 USB image for the Raspberry Pi 4B. Medium. https://medium.com/snapp-automotive/android-automotive-os-11-usb-image-for-the-raspberry-pi-4b-34efe1119356