Development of Embedded Software for Data Acquisition and Calibration of Automotive ECUs

Sergio Arribas Garcia, Thiago Nogiri Igarashi, João Paulo Bedretchuk, Miguel Suchodolak, Anderson Wedderhoff Spengler, and Giovani Gracioli
Software/Hardware Integration Lab (LISHA)
Federal University of Santa Catarina (UFSC)

ABSTRACT

The automotive industry is one of the most demanding industries in the world. Companies in this sector have to follow high-quality standards and reduce the manufacturing cycle time, minimizing the final cost of their products. Specifically, measurements and calibrations of Electronic Control Units (ECUs) play an important role in improving fuel consumption, reducing maintenance costs, and optimizing engine performance.

However, the interaction with ECUs is neither easy nor cheap. It requires complex software, bound to costly hardware and licenses, that few highly qualified engineers are able to configure and use. In this article, we propose an embedded software able to acquire and write data from and to ECUs through the CAN bus, using standard communication protocols. The solution adds features not present in commercial solutions, such as online transmission of data to a cloud server, local data storage, and GPS integration.

Our proposal eliminates the expenses with licenses and reduces up to 90% of the hardware investment, and furthermore, it allows engineers to analyze the results of a car test during its execution, saving time and money in case an error is detected.

INTRODUCTION

The modern automobile industry has a quality-based background, processes, and products following strict controls to provide high-standard results. One of the technical improvements that supported the industry in recent years is the use of software in diverse manufacturing tasks like planning [1], 3D design [3], automation [2], or powertrain calibration and testing.

During the development life cycle of a vehicle, before releasing a new model, electrical, mechanical, and safety systems should be tested and optimized. The components of these systems, and frequently the overall system or car, are tested in laboratories or testing-tracks. When the car is close to being launched, it is tested on the streets and roads, measuring the vehicle's data to ensure safety and validate the defined calibration parameters. Some companies in the market are experts in delivering testing systems and testing tools. In the field related to measurement and calibration of electronic systems, some enterprises provide software and hardware that, after being connected to the ECUs of the car, can diagnose problems and optimize settings. These tools are complex, expensive, and dependent on dedicated hardware where the data extracted is stored and subsequently processed. As an example, using an ES581 and INCA, both products from ETAS GmbH, would cost around an initial U$ 2600.00 with an additional cost of U$1000.00 per year for the software subscription.

In this paper, we proposed a robust, compact, simple to use, and cheap solution able to calibrate and process the data extracted from ECUs in the cloud while using standard automotive communication protocols.

In the first part of the paper, we review the background and explain our motivations. In the second part, we present the Intelligent Acquisition and Analysis System for ECUs (IASE), giving details about its firmware, hardware, and User Interface (UI). In the third part, we conclude the document and show our conclusions.

BACKGROUND AND MOTIVATION

An ECU is an embedded controller used to gather information from sensors and control actuators. In modern vehicles, many ECUs are used to control different functionalities, such as the braking, engine, infotainment system, airbag, and powertrain, among other systems [5][6].

During the vehicle development and testing phase, communication with the ECU is used to measure and calibrate its variables, an important step to ensure safety and efficiency. Therefore, ECUs used during this period can have an additional bus or a debug port for this purpose. Once the ECUs are optimized and configured they are replaced with ECU versions that cannot be modified.

The Association for the Standardization of Automation and Measuring Systems (ASAM) has developed a standard for measurement systems interfaces that allow manufacturer-independent communication [7]. The communication between an ECU and a measurement and calibration system is defined by the ASAM MCD-1 CCP and ASAM MCD-1 XCP standards.

The CAN Calibration Protocol (CCP) was developed in the 1990s and it can be used to program the ECU, read its variables, and calibrate its parameters. CCP only supports Controller Area Network (CAN) as its transport layer [7]. The Universal Measurement and Calibration Protocol (XCP) is CCP's successor as it has the same functionality as

CCP but supports more transport layer protocols such as CAN, Ethernet, USB, SPI/SCI, and FlexRay [7][8].

Both CCP and XCP offer the same structure for event-based data transfer. Data acquisition lists (DAQ) containing ODTs (object descriptor tables) are set up, each list representing a different event, whether it is periodic or not. An ODT is composed of addresses pointing to up to 7 bytes of memory [9][10]. The maximum number of bytes an ODT can point to is 7, due to the fact that CCP and XCP data packets are 8 bytes long, and the first byte is the PID (packet identifier). When the DAQ list event is triggered, all of its ODTs are sent to the connected master.

ASAM also maintains the ASAM MCD-2 MC standard. It defines the contents of the A2L file, a file that describes the ECU configuration, such as the variables and parameters memory addresses, type, length, and conversion equations, among others [7].

Some of the member companies of ASAM develop products that are designed to communicate with ECUs. For instance, ETAS GmbH is a company that offers hardware and software solutions for communicating with ECUs. Among its products, they have INCA, a software that can be used to measure, calibrate, and program ECUs. The software reads an A2L file and allows the user to choose measurement variables and their acquisition frequency, as well as select parameters to calibrate during execution. INCA also supports recording of the values read and displays them on screen. ES581, a CAN-to-USB interface from ETAS costs around U$ 1000.00 and INCA is sold at a subscription model costing around U$ 1600.00 per year, therefore these solutions represent a considerable cost on the vehicle's development process.

Vector Informatik GmbH also offers hardware and software products that aid ECU development and testing. The company offers hardware interfaces that are used to connect the ECU to a computer, the CAN Application Programming Environment (CANape), among others solutions. CANape is a software with functionality similar to INCA, allowing variables measuring and parameters calibration, as well as supporting data analysis and automated processes.

The commercial solutions present some disadvantages like their license price, the complexity of use, the dependency on powerful PCs to run their software, and the necessity of offline data processing. Here we present IASE, a system designed to solve the drawbacks of traditional approaches. It is based on free software running on an optimized and low-cost hardware platform, allowing the integration with cloud technologies, GPS, 4G modem, and online data processing.

IASE

In this section, we describe IASE and its features. We start exposing the system requirements and hardware specifications, where we give details about all the components. In the second part, we continue presenting the software, where we divided the section into firmware and UI. Finally, we describe how the system is configured using experiment files.

SYSTEM REQUIREMENTS - The IASE requirements were specified based on the current commercial solutions, aiming to provide the same performance but with additional features and lower costs. In that sense, to have the same performance as the commercial systems, the IASE hardware must be capable of communicating with the vehicle ECU though CAN network or JTAG bus, process the gathered data and present it to the manufacturer specialists.

The additional features of IASE includes the use of real-time data processing, Internet of Things (IoT) paradigm, artificial intelligence (AI) algorithms and system interface through a smartphone application. To meet all the specifications, the hardware has the following system requirements:

1. Data acquisition through CAN network or JTAG interface.
2. Real-time data upload via 4G following a standardized data format.
3. Global Position System (GPS) localization.
4. Calibration and measurement control via smartphone application.
5. Visual or audio status indicators and start/stop commands.
6. Get the energy to feed the system from the vehicle.
7. Processing power to handle the entire data stream.
8. Memory capacity for data storage for a full experiment.

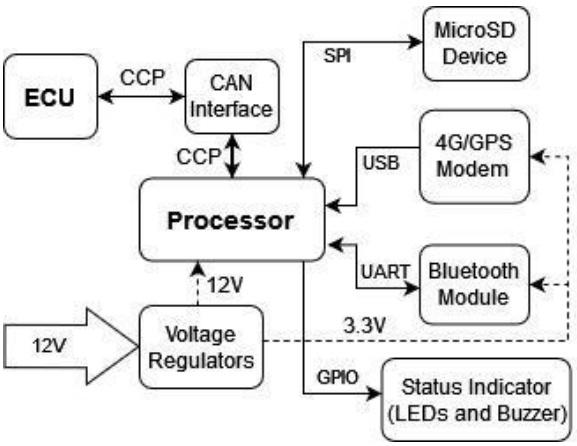
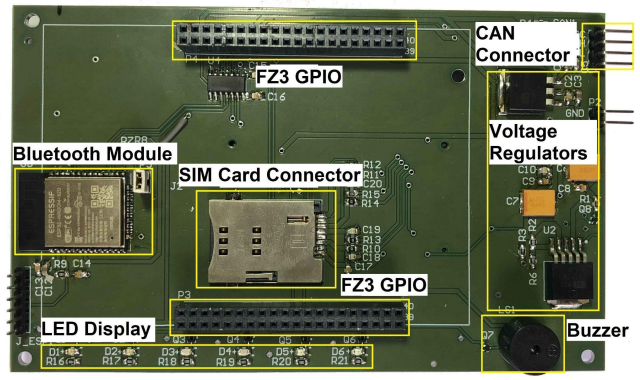


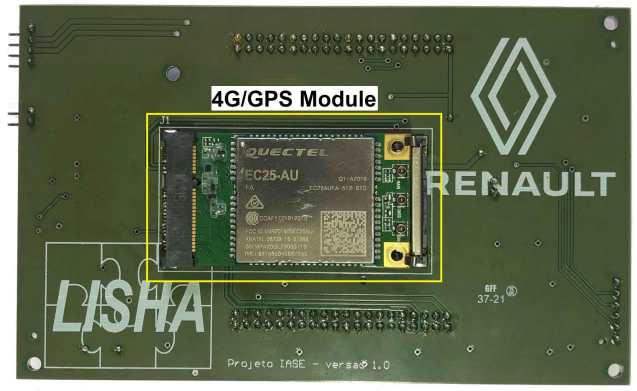
Figure 1. Block Diagram of IASE Hardware.

HARDWARE - The hardware developed to meet the system requirements is composed of a deep learning computing card and a printed circuit board (PCB), containing all the necessary modules and peripherals. Figure 1 shows a block diagram of the hardware with the seven main parts of its architecture.

The processor unit connects all the modules, peripherals, and the vehicle ECU itself. To comply with all the required protocols to connect every module on the board, IASE uses an FZ3 card, produced by Shenzhen Myir Technology. This card is a System-on-a-Chip (SoC) composed of a Programmable Logic (Zynq Ultrascale+ FPGA) and a Processing Subsystem (ARM quad-core Cortex-A53 and dual-core Cortex-R5). The communication with the ECU is made through CAN communication protocol and uses the FZ3 card CAN ports and DB9 connectors.



(a)



(b)

(c)

Figure 2. IASE Hardware: (a) top view, (b) bottom view, (c) top view with the FZ3 card.

In order to provide 4G network connection and GPS localization, the hardware uses a Mini PCIe LTE category 4, the Quectel's module EC25-AU. A SIM card of a local mobile phone operator is used to allow the 4G network access.

The IASE interface is provided by a smartphone application that communicates with the hardware through Bluetooth Low Energy (BLE). The Espressif Systems module ESP32-WROOM-32D is used to interface the communication between the smartphone and the main processor.

As a secondary way to communicate with the user, the hardware has an LED display, with six LEDs and three different colors that change according to the system status, and a buzzer that warns the user about status changes or errors.

A 16GB MicroSD card is used to store the operating system to boot the device and save the gathered information from the vehicle ECU.

The system power is provided by the vehicle battery, therefore a set of voltage regulator circuits are used to source each part of the system with an appropriate voltage level.

To put all the components, modules, and circuits together, a PCB was designed and manufactured. Figure 2a presents a top view of the PCB. Figure 2b shows the bottom view and Figure 2c shows the top view with the FZ3 card connected to it.

SOFTWARE - We organize the software in 3 main parts The first one focuses on the software running on the hardware connected to the ECU, the second one on the user interface that runs on an Android phone, and the final part explains how the experiments are configured in the system.

Firmware – The FZ3 board supports the Xilinx's Linux Board Support Package, named Petalinux, which automatically builds and generates a Linux kernel, including required device drivers. IASE uses Petalinux 2020.1 and the operating system (OS) was built using Petalinux Tools. The Linux distribution is based on Ubuntu. We enabled the USB driver, during the OS building process, to allow the EC25 module to connect to the FZ3. The same can be said about the PPP (point-to-point protocol) daemon, which was used for the configuration of the 4G network.

The firmware was programmed using C++11 and, because it is a well-known Object-oriented programming language, we use classes to organize the code. We modeled the firmware that runs on the FZ3 using Unified Modeling Language (UML). Firstly, we defined the use cases for the system and depicted them using a use case diagram. Figure 3 shows the final version of the diagram. The users of the system are depicted as humans on the left side, and in the square on the right side, we have the functions of the system. According to its access privilege, the user can

connect and activate the system, start the configured experiment or change the experiment.

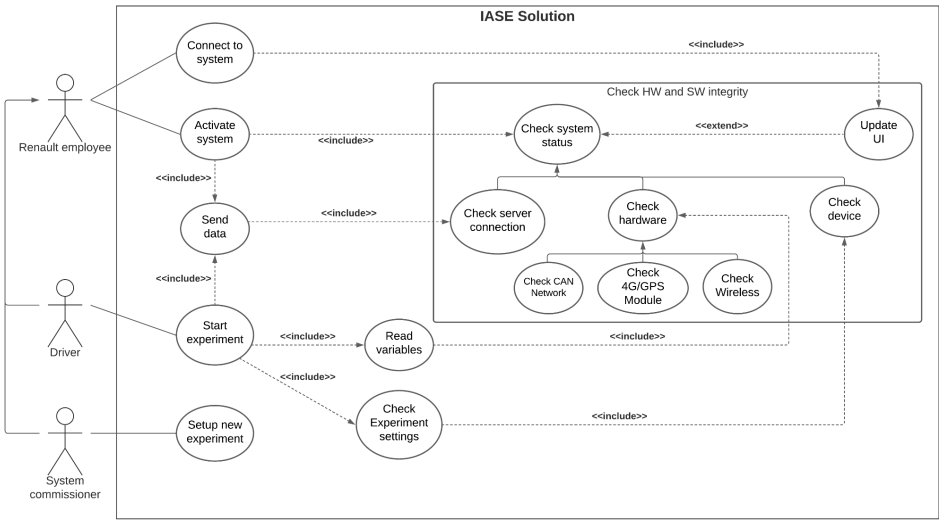


Figure 3. Use cases diagram

We designed the classes of the internal structure of the firmware using a class diagram. In Figure 4 we modeled the classes related to the sequence control of the firmware, state-machine classes. We grouped all the classes according to the part of the software they control:

- State-machine classes: control the sequence of the system. The class StateMachineBase is the base class for all the state-machines of the code. It creates a thread that executes a function depending on the current state, this allows us to have diverse processes running in parallel using all the cores of the FZ3. Inheriting from the base class state-machine we designed 3 classes that control the data management, communication with ECUs and communication with external devices and UI. The 3 state-machines run together and exchange data using mutexes to avoid synchronization problems.
- Experiment configuration classes: open the JSON file with the experiment configuration and prepare the data model to store and get variables from ECU. We have classes to store different kinds of variables, one class to define the variables and other classes responsible for the conversion of data to the international system, once the variables are read from ECU.
- Communication with ECU: A main class called ECUReader controls the reading and writing process from and to the ECU. We created several classes related to the communication protocol. In the current version of IASE only the CCP is used but the CCP class inherits from an interface class Protocol. This allows us to enhance the system with other protocols like XCP for example. We also have classes to control the CAN bus, because it is the bus used currently. The class inherits from an interface class also allowing further implementations of other buses like Ethernet.
- Data-management: Here we use a class called DataController to get the Smartdata generated by the ECUReader class, which is stored in a common circular buffer. The circular buffer has mutexes that avoid the writing and reading process to collide. Here we created an interface to manage the connection with the cloud server. At the moment, the EC25 chip is used to control the communication but a new class could be created inheriting from the interface to have other alternatives. IASE also has classes to compress the data that was already sent to the server and control the storage in the SD card.
- External devices communication: IASE has classes used to control a group of LEDs and a buzzer that we used to inform the user about the status of the system. The class activates the outputs of the FZ3 depending on the state of the process. We also created a class called ESP32 that inherits from an interface class that allows us to send information to the Android app. The ESP32 uses the UART of the FZ3 for the data transfer, but new classes could be created to send information using a WiFi card for example.

IASE uses state-machines to manage the executing tasks of the system, for example, configuring devices, and reading or writing to the ECU. The firmware has a main state-machine, which controls the overall system, and 2 other state-machines that are responsible for the data processing and acquisition and writing to the ECU.

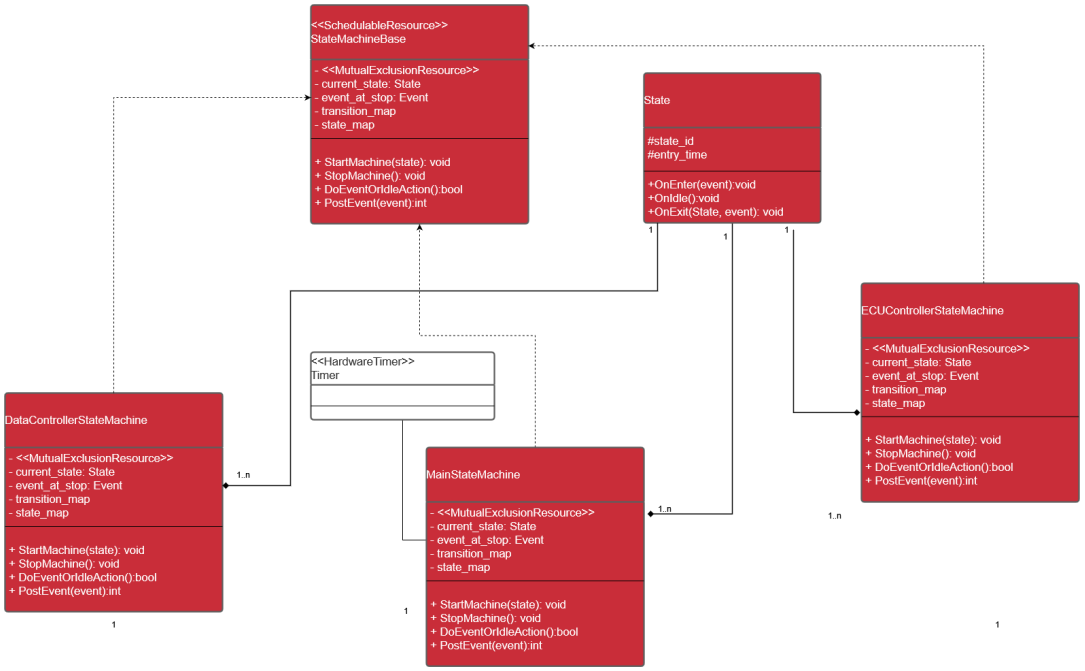


Figure 4. Class diagram - State-machines

Each of the state machines have a thread, but some can create a second thread for time sensitive functions or commands that require waiting for an response.

Figure 5 shows an overview of the states in the main state-machine. The orange figures represent the states and the orange lines the transitions between states. The main state-machine starts at the *Init* state, in this state it creates all other state-machines objects and changes its state to *Prepare DataController*. This state is responsible for starting the DataController state-machine and upon success, alters its state to *Prepare ECUReader*, which does the same as the last state, but instead starts the ECUReader state-machine. When the other state-machines are started, the main state-machine enters an *Idle* state and only leaves through user requests or if an error is detected. In the *Idle* state IASE updates the UI and informs the user about the status of the acquisition and calibration process.

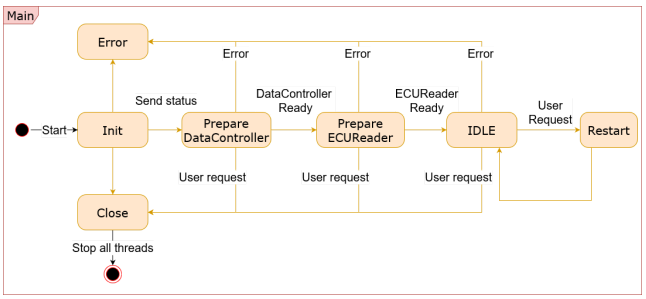


Figure 5. Main state-machine

The number of variables available in the state-of-the-art ECUs is in the order of thousands, but not all of them could be acquired simultaneously using CCP. The maximal amount of data is limited by the speed of the CAN bus, and for that reason, we have an experiment file that defines the variables the ECU will transmit to the IASE. We present the experiment file in the next subsection *Experiments*.

When IASE opens the experiment file and extracts the variables to be measured and calibrated, it establishes the communication with the ECU and configures it, sending the addresses of the variables and the individual acquisition period. IASE triggers the acquisition process by sending a command to the ECU and from that moment on, the current values of the selected variables are available in the CAN bus.

The acquisition of data is a critical process in the system, and it runs in a separate thread inside the code, this is controlled by the ECUReader class and the one of the state-machines. The same occurs with the transmission of data to the cloud or the internal processing of data, controlled by the DataController class and a state-machine.

As IASE captures the values of the variables from the CAN bus, it transforms them into Smartdata [4] and places them into a temporary buffer. Smartdata is an information structure that includes the necessary metadata making it self-contained regarding spatial location, timing, semantics, and truthfulness. It includes value, unit, the position where it was generated, the timestamp of the moment it was produced, and its validity. IASE generates Smartdata series for each variable. These series are groups of Smartdata that are transferred to the cloud and stored on the SD card of the FZ3 together. The use of series reduces the size of the Smartdata, because the common information of every single Smartdata is only saved once per series. Having a

standard data format allows the same processing algorithms to be applied in different car models, for instance.

The firmware utilizes another thread to read the temporary buffer to build the Smartdata series. This thread is responsible for calculating the temporal offset between data points, saving the data into the SD card, and preparing the data packets that will be sent to the cloud. The user could configure the data measurement from the ECU based on a regular sampling frequency or based on the position of the cylinders of the motor, which generates a data stream with irregular sampling times. In the latest case, the temporal distance between samples is calculated. The firmware parametrizes the number of Smartdata per Smartdata Series. When the series are completed, IASE moves them to the SD card and sends them to the cloud.

The Smartdata includes the position where it was generated, and this information is obtained using a GPS module attached to the FZ3. The firmware reads and updates its current position from the GPS and attaches it with the variables coming from the ECU. Afterward, the user could analyze the results of his experiments probing how the position affects the operation of the motor.

The SD card of the FZ3 contains the Linux Operating System, the firmware of the IASE and also the measurements of the experiments that were performed recently. During the experiments, IASE creates 2 types of files with the Smartdata and saves them in two folders, one type with data transmitted to the cloud and a second type with data not-transmitted to the cloud. The firmware processes those files separately. The files whose data is already in the cloud are stored in the SD card for a fixed number of days, the firmware checks the folder after its start and removes the old files. On the other hand, the folder with file not-transmitted to the cloud is preserved and checked constantly, when the firmware is active it tries to transfer the data of the files to the server, and only when this process is finished removes the files from the folder.

IASE commands a third thread that transfers the data to the server in the cloud. The thread gets the Smartdata series once they are completed and informs the server that the series are ready to be uploaded. After the server's confirmation, it begins the uploading process. The transmission uses SSL certificates to avoid unauthorized external interferences with the server. The server preprocesses the data when it receives it and gives feedback to the IASE indicating if the SmartData was stored in the server. If the feedback is negative, IASE will maintain the data in an output buffer and will try to upload it again. The non-transmitted files are created in the SD card with the content of the output buffer only at the moment the user switches off the IASE experiment.

The server which receives the data from the FZ3 handles the data using workflows. It can perform verifications and adaptation of the information in real-time. For instance, it could be configured to detect failures or malfunctions in components of the motor. Using AI algorithms it could predict problems and notify the user. Similarly, workflows could be configured to inform the user that a variable has reached values out of tolerance for example. The user can use this information to stop the experiment, saving time and avoiding damage to the car.

The feedback of the server is not the only interaction between the user and IASE, the firmware communicates with an Android application that we will present in the following subsection *User UI.* The driver of the car performing the experiment could start, stop and monitor the status of the system with the app. The FZ3 is not equipped with Bluetooth or WiFi technologies and the communication with the mobile phone and the application is done by an ESP32 module which is connected to the FZ3 via UART.

The firmware extracts the status of all processes, threads, and external devices and sends them continuously to the ESP32 using UART. The packet sent and received from the Bluetooth is configured to have 23 bytes, inside these bytes the firmware sends:

- Device control block (bidirectional): 1 byte containing the status of the experiment
- Device status block (IASE to app): 7 bytes containing the status of the ESP32, GPS, 4G, temporary buffer, SD Card, and ECU.
- Device calibration block (App to IASE): 7 bytes used to calibrate variables.

IASE calibrates the ECUs with the information received from the Android app. Software in the mobile phone or tablet is configured using the same experiment file that the FZ3 uses. As we will present in the subsection *Experiments,* the configuration file contains data about the variables of the ECU (address, name, type…). The user will select a variable and a value to be calibrated in the Android app, and it will send it to the IASE using BLE. Once the ESP32 receives the information, it transfers via UART to the FZ3. The state-machines in the firmware control the moment the variable could be calibrated and use the classes responsible for the communication with the ECU to write the new values on the CAN bus. After this process the firmware checks the ECU feedback verifying if the value was updated correctly. The confirmation is used to indicate to the Android application that the value was updated or informs that there was an error in the process.

User UI – IASE has an Android application, developed in Android Studio, that communicates with the ESP32-WROOM-32D through Bluetooth Low Energy (BLE) communication protocol. This application supports a variety of Android versions, although it stops getting support at Android Lollipop (5.0) and older versions. The purpose of this application is to monitor the IASE processes. It shows the information about the experiments and the status of the hardware, it also initializes and stops

the acquisition process from ECU, besides being responsible for the calibration of parameters.

One of the most important parts of the application is the graphical User Interface (UI). It can be divided into three main sections: Log in and Connectivity, Status Pages, and Engineer Interface.

The Login and Connectivity can be described in some specific pages shown in Figure 6. The first page of the software allows the user to choose between two profiles in the application: Driver and Engineer. The engineer profile requires a password to confirm the login. After login, the app leads to the Bluetooth page, which presents to the user a list of available devices to establish the BLE connection.

Figure 6. Login and connectivity screens

After getting a BLE connection, the application initializes the status pages. The first page appears when the system is preparing the data acquisition, after the configuration is completed, the system is ready to start and waits for the user to start the experiment.

Once the experiment starts running, the app changes to the status screen. In this screen, the user can see if the experiment is running, is stopping, finished, or presenting an error during the process, an example of this screen is depicted in Figure 7.

The activation or finalization of the experiment is triggered by the application. The button in the system status screen will send the starting/finishing command to the FZ3 when pressed.

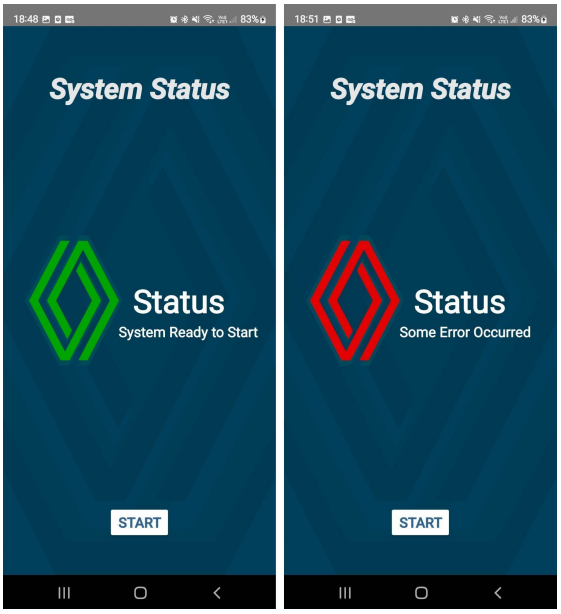


Figure 7. System Status screens

Besides the System Status screen, the application offers the Engineer Status screen. This screen is only available to the user if he logged into the system as engineer. The engineer profile has access to the modules tab, which shows the information about the 4G, GPS, Buffer, SD, and ECU status, the alarms tab, which explains the specific problem or warning about any status that needs attention, and finally the Calibration tab. The second screen in Figure 8 represents the Calibration tab, a tab that offers a list of all parameters that can be calibrated in the ECU. The parameters are acquired from the same experiment file used by the FZ3. Clicking on a parameter of the list, the application describes in detail the information of the variable giving its name, description, and current value, as shown in Figure 9. This tab is where the user calibrates the parameters and automatically adapts its aspect depending on the type of variable (boolean, numbers, or tables).

Experiments – During the vehicle's testing phase, experiments are done to validate its calibration and identify potential problems. Measurement variables are selected and used to inform the ECU which variables the user is interested in. In order to facilitate information handling, IASE uses JavaScript Notation Object (JSON) files, an easy-to-use data format that is supported by many languages and libraries.

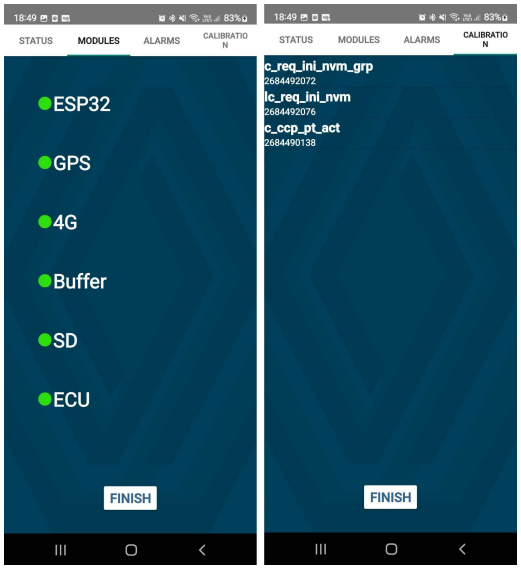


Figure 8. Engineering aspects

Figure 9. Calibration variable tab.

The ECU's A2L file is converted to a JSON file and used by IASE's experiment generator interface. This interface reads the JSON and allows the user to select the desired measurement variables, with their acquisition period, and the calibration parameters. The interface then generates another JSON file, the experiment JSON, that is uploaded to the FZ3 and an android smartphone.

The experiment JSON contains all selected measurement variables, calibration parameters, and conversion information necessary. Each measurement variable and calibration parameter contains its name, data type, memory address, address extension, and the conversion identifier, all of them from the A2L. Variables also have the acquisition period selected as well as the unit, dev, and workflow, numbers used to identify the variable in the server. Figure 10 shows a JSON structure for a measurement variable.


```
{
  "name": "Vbx_igk",
  "longIdentifier": "Ignition key signal",
  "datatype": "UBYTE",
  "conversion": "utCpcna_olpdfvb_iFvAkza_nC___01",
  "unit": "0x00000001",
  "ecuAddress": 3489673766,
  "ecuAddressExtension": null,
  "source": 30,
  "dev": 1,
  "workflow": 0
}
```


Figure 10. Experiment JSON structure used for measurement variables

Values received from the ECU are first converted to physical values and then, if desired, are converted to their SI unit equivalent. IASE supports both rational functions and tables as conversion methods.

Both the FZ3 and the smartphone must receive the same JSON experiment. The FZ3 uses it to communicate with the ECU and the smartphone uses it to know which calibration parameters were selected and display them on screen.

In the IASE version for CCP, which uses CAN as its transport layer, the system is limited by the protocol bandwidth. To avoid reaching the limit, the ECU is configured to allow a maximum number of bytes transferred for each acquisition period (also called raster). Each raster contains a field called length, which is the amount of ODTs dedicated to it. Since each ODT can contain up to 7 bytes of data, multiplying the length by 7 gives the maximum number of bytes that a given raster can contain. It is important to emphasize that since variables can contain different sizes in bytes, how they are organized in ODTs matters and can influence the real maximum number of bytes transferred.

RESULTS AND VALIDATION

This section presents the results of tests done to validate IASE's performance and requirements.

In order to confirm that IASE can collect the same amount of data as a commercial system, an experiment was created using almost all of the bandwidth allowed by the ECU. Table 1 shows the amount of variables assigned to each raster, being the variables 1, 2 or 4 bytes long.

Acquisition period (raster)	Number of variables
Cylinder 1	23
Cylinder 2	23
Cylinder 3	23
Cylinder 4	23
4 ms	28
5ms	24
10ms	60
100ms	54

Table 1. Amount of variables per raster on high load experiment

From the used ECU A2L file, it is known that all rasters can each contain up to 7 ODTs, equivalent to 49 bytes of data, with the 10 and 100 millisecond rasters each containing up to 15 ODTs, equivalent to 105 bytes. This means that at maximum, 504 bytes of variables can be configured per experiment.

This experiment used 477 out of the 504 bytes possible, which is equivalent to 94.6% of the total allowed bandwidth. IASE was able to read and send the data to the server without delays, and without consuming much of the board's processing power and memory.

Figure 11 shows the update rate of the gathered data, in SmartData format, to the IoT server over time for the tests made with the presented experiment. The average measured transfer rate was 726 kB/s and its related number of variable values was 23131 elements/s. This result shows that the system is capable of sending all the collected data for a full experiment.

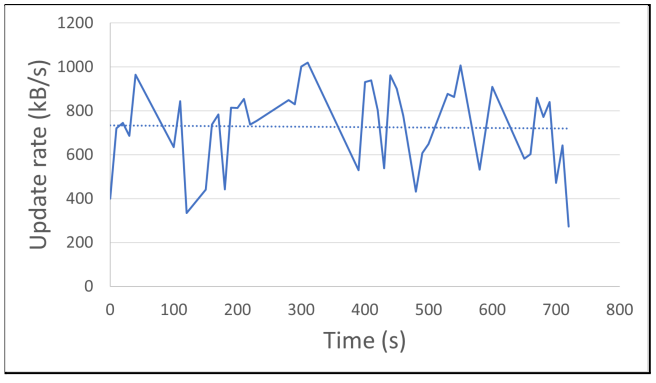


Figure 11. Average data transfer speed over time.

After updating the vehicle data to the IoT server, it can be processed by AI algorithms, acquired through some software or monitored in web applications. For instance, Figure 12 shows a screenshot of the vehicle engine speed data, acquired, processed and uploaded by IASE.

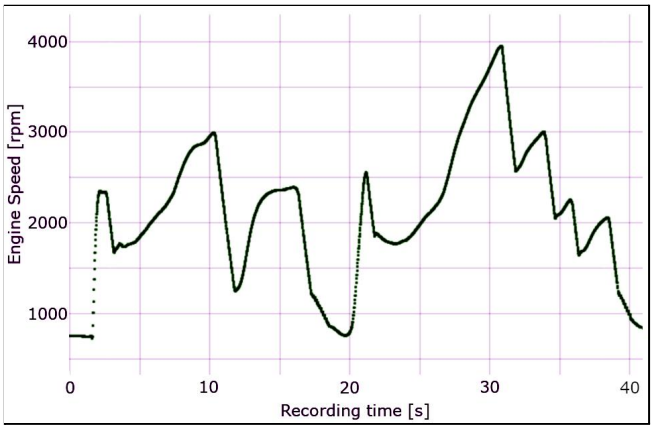


Figure 12. Vehicle engine speed acquisition and processing results for the IASE.

The results presented by IASE were validated using the INCA software. Comparing the data acquired and processed by both systems, no divergences were detected, showing the reliability of the proposed solution.

Costs – The estimated cost of the IASE system is around $430 considering all the components, wires, and protection cases. It does not require a license and has no cost related to software. According to our research in the market, compared with other commercial systems, it represents more than 90% of savings, since the average cost of the commercial solutions is $5000 and the license of the software should be renewed yearly being this cost around $1500. These commercial solutions run in PCs that should be carried in the car during the experiments, they monitor the data and store the raw data to be analyzed.

CONCLUSION

This paper has presented IASE, a system designed to measure and calibrate ECU variables using CCP protocol. Its main features are the integration of a 4G network and GPS and the integration of the system with a cloud server which stores and processes the data assisting the user in the processing of data in real-time.

The IASE system, when compared with commercial solutions, is a compact and robust alternative that does not require additional hardware like computers or expensive tablets to be configured. It is controlled using a mobile phone with Android, that can be used to calibrate and monitor values of variables in real-time. Beside that, the cost of the proposed solution showed to be less than 10% of the current average cost of commercial systems.

In addition to this, the cloud server in which IASE stores the SmartData extracted from the cars, processes the information generated in real-time. The processing of data during the experiment reduces time losses and saves money

by avoiding performing experiments when malfunctions or bad configurations are detected. In other solutions used in the automobile industry, the experiments are concluded and the data is stored on expensive computers. This data is transferred later to an IT team that will process it and will provide the results. In many cases, experiments, where the car is driven for hours, could be invalid or useless and should be discarded and repeated.

We leave for future work some improvements of the systems as the optimization of the hardware. The current processing unit, the FZ3, only uses 5% of its processing power, indicating that it could be replaced with a simpler and even cheaper unit.

IASE is able to communicate with ECUs compatible with CCP, but the firmware is prepared to implement further protocols like XCP in future works.

ACKNOWLEDGEMENT

This work was supported by Fundação de Desenvolvimento da Pesquisa - Fundep Rota 2030/Linha V 27192.02.01/2020.09-00. We would like to thank Renault do Brasil for the support of this project.

REFERENCES

[1] F. Biesinger, B. Kraß and M. Weyrich, "A Survey on the Necessity for a Digital Twin of Production in the Automotive Industry," 2019 23rd International Conference on Mechatronics Technology (ICMT), 2019, pp. 1-8, doi: 10.1109/ICMECT.2019.8932144.

[2] Hajarnavis, V. and Young, K. (2008), "An investigation into programmable logic controller software design techniques in the automotive industry", Assembly Automation, Vol. 28 No. 1, pp. 43-54. https://doi.org/10.1108/01445150810849000

[3] Zimmermann, P. (2008). Virtual Reality Aided Design. A survey of the use of VR in automotive industry. In: Talaba, D., Amditis, A. (eds) Product Engineering. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8200-9_13

[4] A. A. M. Fröhlich. Smartdata: An iot-ready api for sensor networks. Int. J. Sen.Netw., 28(3):202–210, jan 2018.

[5] M. S. U. Alam, "Securing Vehicle Electronic Control Unit (ECU) Communications and Stored Data." Order No. 10999576, Queen's University (Canada), Ann Arbor, 2018.

[6] M. Faizan and A. S. Pillai, "Dynamic Task Allocation and Scheduling for Multicore Electronics Control Unit (ECU)," 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), 2019, pp. 821-826, doi: 10.1109/ICECA.2019.8822086.

[7] M. Naserian and K. Krueger, "A test-bed for verification of vehicle safety communications applications," 2009 IEEE International Conference on Electro/Information Technology, 2009, pp. 327-331, doi: 10.1109/EIT.2009.5189637.

[8] Deshpande, Uttam U., et al. "Data acquisition and Calibration of ECU using CAN protocol." International Journal of All Research Education and Scientific Methods (IJARESM), 2017, pp. 2942-2950.

[9] Muresan, M., Pitica, D., & Chindris, G. (2008, May). Calibration parameters principles for MATLAB S-functions using CANApe. In 2008 31st International Spring Seminar on Electronics Technology (pp. 105-110). IEEE.

[10] He, Y., Sun, X. (2009). An XCP Based Distributed Calibration System. In: Ślęzak, D., Kim, Th., Kiumi, A., Jiang, T., Verner, J., Abrahão, S. (eds) Advances in Software Engineering. ASEA 2009. Communications in Computer and Information Science, vol 59. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10619-4