Universidade de Aveiro Departamento de Engenharia Mecânica
2022

Miguel Francisco De Amaral Pinto

Dashboard Interativa do Estado Global do ATLASCAR2

Interactive Dashboard with the Global State of the ATLASCAR2

Universidade de Aveiro Departamento de Engenharia Mecânica
2022

Miguel Francisco De Amaral Pinto

Dashboard Interativa do Estado Global do ATLASCAR2

Interactive Dashboard with the Global State of the ATLASCAR2

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado C/ Agregação do Departamento de Engenharia Mecânica da Universidade de Aveiro.

O júri / The jury

Presidente / President	**Prof. Doutor Miguel Armando Riem de Oliveira** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Rui Manuel Escadas Ramos Martins** Professor Auxiliar da Universidade de Aveiro
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado C/ Agregação da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Em primeiro lugar, gostaria de agradecer aos meus pais e irmã que estiveram sempre ao meu lado nos momentos mais difíceis desta longa jornada de 5 anos.
Ao Professor Vítor Santos pela paciência, disposição, orientação e apoio ao longo deste trabalho, assim como ao Eng. Heitor e a todos os elementos do LAR pela disponibilidade e prontidão para ajudar.
Agradeço aos meus companheiros do 3ºY pela colaboração e amizade mostrada em todos os momentos.
E por fim ao meu grupo de amigos e em especial ao R.A. , pela boa disposição, companhia e por me terem ajudado a crescer nesta etapa da minha vida.

Palavras-chave

ATLASCAR2; ADAS; Arquitetura ROS; Dashboard; Controller Area Network; Display;

Resumo

A indústria automóvel tem desenvolvido inúmeros esforços para tentar tornar a nossa experiência de condução mais segura e confortável. Atualmente, uma das soluções desenvolvidas são "dashboards". Estes dispositivos são Sistemas Avançados de Assistência ao Condutor que permitem aos utilizadores obterem todas as informações relativas ao estado do veículo que os transporta através de um "display" dinâmico. No âmbito do projeto ATLAS, esta dissertação tem como objetivo criar uma dashboard para o ATLASCAR2. Tendo em vista este projeto, primeiro foi instalada uma nova solução de energia para a unidade central de processamento do veículo, responsável pelo funcionamento dos equipamentos instalados. O antigo quadro elétrico do carro apresentava algumas limitações. Por essa razão, foi instalado um novo quadro e colocado no porta-malas do veículo. Em seguida, o ATLASCAR2 foi equipado com um inversor que retira energia da bateria de chumbo do veículo para alimentar o computador. Numa segunda fase, foi criada uma nova rede de informação baseada numa arquitetura ROS que fornece o estado dos sistemas integrados no carro ao display da dashboard. O barramento Controller Area Network do veículo foi utilizado para este fim. Este trabalho apresenta a solução desenvolvida e todas as funcionalidades nela incorporada. Por fim. foi realizado um teste que auxiliou na avaliação da usabilidade da nova solução.

Keywords ATLASCAR2; ADAS; ROS Architecture; Dashboard; Controller Area Network; Display;

Abstract The transportation industry has deployed new efforts to make our driving experience safer and more comfortable. Nowadays, one developed solution points to dashboards. These devices are an Advanced Driver-Assistance System that allows the users to check information regarding the vehicle that transports them through a dynamic display. Within the ATLAS project, the present dissertation aims to create a dashboard for the ATLASCAR2. Given this need, a new power solution for the central process unit responsible for booting all external installed equipment was installed first. The electric board already in place presented some limitations. Therefore, a new one was installed and placed on the vehicle's trunk. Next, the car was equipped with an inverter that withdraws energy from the vehicle's lead battery to feed the computer. Then, an information network built upon a ROS architecture had to be created to feed information from the car's in-built systems to the dashboard display. The Controller Area Network bus of the vehicle was used for this purpose. This work presents the developed solution and all features embedded in it. In addition, a field test was performed, which helped to evaluate the new solution's functionality.

Contents

List of Tables

Intentionally blank page.

List of Figures

Acronyms

AC Alternating Current.

ADAS Advanced Driver-Assistance System.

CAN Controller Area Network.

DC Direct Current.

ECU Eletronic Control Unit.

GPS Global Positioning System.

GUI Graphical User Interface.

HMI Human-Machine Interface.

LAR Laboratório de Automação e Robótica.

LIDAR Light Detection And Ranging.

OBD-II On-Board Diagnostic II.

Radar Radio detection and ranging.

ROS Robotic Operating System.

UPS Uninterruptible Power Supply.

Intentionally blank page.

Chapter 1

Introduction

With the recent technological advances, the transportation industry constantly demands continuous car performance progress. Therefore, customers have now built unprecedented expectations about their vehicles, which triggered brands worldwide to focus on making our driving experience as safe and comfortable as possible.

One crucial factor is the ability of drivers to gather as much information as they can regarding their car. The solutions developed are based on human-machine interfaces Human-Machine Interface (HMI) called dashboards which allow drivers and passengers to easily engage with the vehicle and the outside world. They can be embedded with navigation systems, control panels, built-in screens, buttons, and driving assistance. Dashboards turns a vehicle into an ecosystem of interconnected parts that work together to make our driving experience more convenient, personalized, and enjoyable.

This dissertation aims to develop a fully functional dashboard for ATLASCAR2, an autonomous research vehicle from the University of Aveiro, by displaying data regarding the vehicle's global state on a dynamic display.

1.1 Context on the ATLAS project

The ATLAS project was created in 2003 by the Laboratório de Automação e Robótica (LAR) of the Department of Mechanical Engineering of the University of Aveiro, focusing mainly on the research and development of robust sensory systems [1]. During the first years, the group focused on creating a series of autonomous robots (Figure 1.1), thus advancing with its attendance at the Portuguese Robotic Open.

(a) (b) (c)

Figure 1.1: ATLAS autonomous robots [1].

The participation was such a success that in 2009 the development of a fully autonomous vehicle, the ATLASCAR1 (Figure 1.2), began. This prototype had a series of embedded equipment, which collected all the internal information of the car and its surroundings. Subsequently, the sensors sent the acquired data to a central computer, which returned a series of orders for the actuators to act as intended. A Ford Escort Station was equipped with the most diverse instruments to be as reliable as possible, being systematically improved year after year.

Figure 1.2: The ATLASCAR1 vehicle [1].

Over time, the team decided to switch to another vehicle. The ATLASCAR2 (Figure 1.3) is a fully electric Mitsubishi i-MiEV and on it were incorporated all the progress and results obtained so far. Therefore, the vehicle is more technologically advanced than its predecessor and facilitates the research work carried in it.

1.2 Problem description and objectives

The ATLASCAR2 focuses on becoming the driving experience more comfortable, autonomous and mistake-free. Therefore, many Advanced Driver-Assistance System (ADAS), such as sensors and Light Detection And Ranging (LIDAR), were embedded in the vehicle. These devices are gaining more and more attention as a critical technology to increase driver awareness and safety. The implementation of a dashboard also serves this purpose. This hardware will keep the user updated by displaying all data related to the vehicle on a dynamic screen, enhancing the driver's sense of security and joy while driving the ATLASCAR2.

In order to install the dashboard on the vehicle, a new electric power supply must first be installed. This task will be done on top of the current installation. As researchers developed work on ATLASCAR2, an outside power source was added to feed the vehicle's computational system, which is the car's central processing unit. Throughout the years, this solution has become less suitable for the team's demands due to a lack of autonomy, and future work on the car can be compromised as we overload its power supply.

Figure 1.3: The ATLASCAR2 vehicle.

The data related to the car status is present in the control units of the vehicle and it can be accessed as a result of past developments. Due to security reasons, these messages are encrypted and they are hard to trace. Nevertheless, the ATLASCAR2 model is commonly used by investigators worldwide for autonomous driving projects, and much work has been done in this field. The dashboard must be capable of reading the messages received from the automobile and processing them, so a software application must be developed. Most information will be related to the car's original features, such as autonomy and velocity.

As the software feeds the dashboard with information, an interactive display will keep the driver updated on the vehicle's current state. The user must be capable of easily communicating with the machine by incorporating a user-friendly interface. To summarize, the following topics are the main objectives of this dissertation:

- Increase the power autonomy of the computational system;
- Create a software infrastructure with the global status of the vehicle in real-time;
- Development of an interactive display;

1.3 Document structure

This document is divided into seven chapters. The first and second focus on introducing the problem, describing the main objectives and explaining the basis behind this work. The third is related to the experimental infrastructure. On the one hand, it

analyses relevant features and characteristics of ATLASCAR2. On the other, it covers all software used for this project. Chapter 4 discusses the power solution to extend the autonomy of the processing unit implemented on the vehicle. Chapter 5 discusses the process of receiving information from the car. Chapter 6 includes all aspects related to the dashboard and display. Finally, the conclusions and proposals for future works are presented in the last chapter.

Chapter 2

State of the art

The following chapter addresses the state of the art. The section 2.1 gives some relevant details on the car's architecture and electrical systems. Section 2.2 analyses Advance Driver-Assistance Systems, some of which are already implemented on the ATLASCAR2. Section 2.3 gives context on features to implement on a dashboard and section 2.4 mentions important work related to this project developed in LAR and worldwide.

2.1 Background on vehicle architecture and their eletric systems

In the context of this work, it is crucial to address the current vehicle architecture to understand where the displayed information comes from. Nowadays, modern vehicles are managed by a series of networked controllers, which allow easy communication among all in-car hardware.

One feature that has shaped current automobiles is the Eletronic Control Unit (ECU). These devices were created in 1970 by US manufacturers and are embedded systems used to monitor and control specific devices of vehicle units [2]. Modern cars have various ECU's, each associated with a specific device such as the brakes, lights, or steering.

In 1986, Bosh created the Controller Area Network (CAN) and found a way to facilitate data transmission between ECU's [3]. They are connected through a "pathway" inside the car composed of semiconductors called CAN bus, where other units are listening. CAN is a serial multimaster protocol which means that any ECU can send a message to the bus when it is free. All nodes connected to the bus can receive frames and process them. The node that sends the data is called the transmitter and the node that receives it is called a receiver. This bus can be accessed using the On-Board Diagnostic II (OBD-II) port, usually situated under the vehicle's steering wheel. Nowadays, road entities force car manufacturers to implement this robust feature on their products, as they are crucial for drivers to overwatch malfunctions.

2.2 Advanced Driver-Assistance Systems

Fast growth in ADAS development has been seen worldwide over the last years. Due to recent technological advances, car manufacturers have embedded them in their vehicles as they play a crucial role in road safety. According to [4], in 2015, 94 % of US car crashes occurred due to human error. Thus, their main purpose is to reduce this statistic by assisting drivers in gathering important information about the car and the external world. The driver's comfort is another aspect, as they provide a better on-road experience.

ADAS are composed of a wide set of instruments (Figure 2.1). Sensors measure the distance from surrounding obstacles. Radio detection and ranging (Radar) allows monitoring of the road ahead and is particularly helpful if a vehicle hides behind other obstacles such as corners or other cars. Assisted by artificial intelligence and deep learning, cameras allow recognition of different objects in our driving experience, from pedestrians to traffic signs. These gadgets can be combined with embedded actuators that have the power to control crucial ECU's of the vehicle, such as the break paddle or the steering wheel. Adaptive Cruise control, automatic parking, driver monitoring system, and the ability to dodge obstacles are a few examples of this development. Another aspect that must be highlighted is consumption. Intelligent Global Positioning System (GPS) can advise the driver on more efficient and economical driving practices [5].

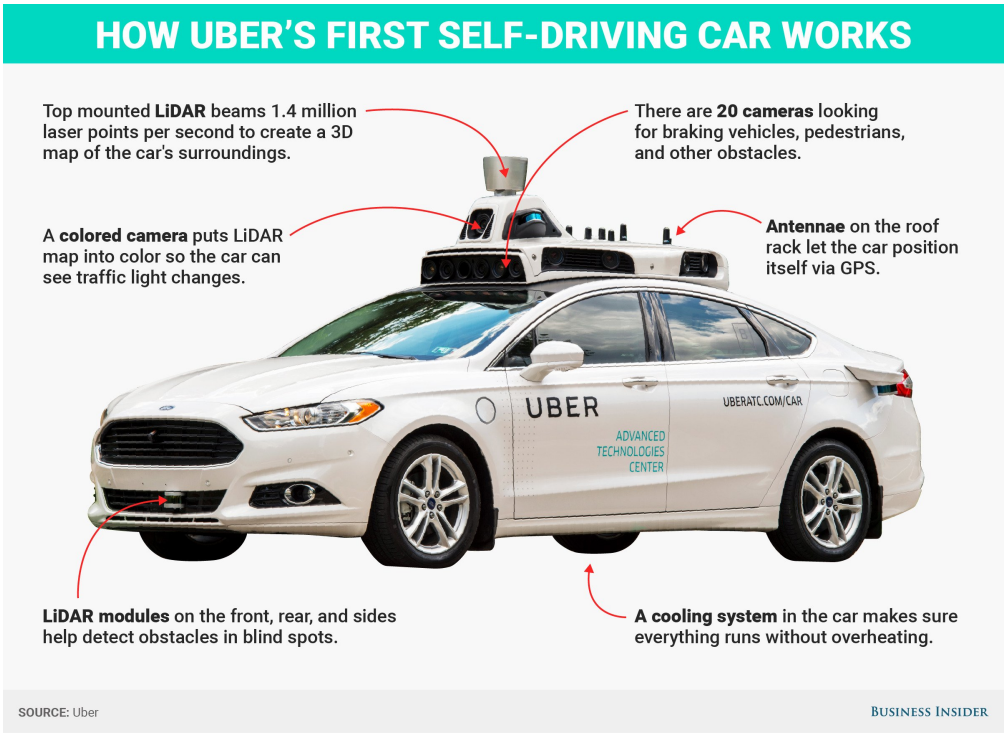


Figure 2.1: Examples of ADAS instruments [6].

There are two types of ADAS: Passive and Active [7]. The passive ones have the job of informing about certain situations as the driver must then act to correct them. The battery or low tire pressure warnings are two examples. In this case, a warning

light flashes in the vehicle's control panel when a malfunction occurs. They can also be associated with internal gadgets such as rear cameras or parking sensors. With active devices, the vehicle takes direct control of its systems, as they act directly on the car's ECU's and are activated when the system feels necessary. Self-parking or lane-centering assistance are some cases.

Due to past developments, ADAS are already part of the ATLASCAR2 (Figure 2.2). To accomplish their full purpose, the driver must visualize the information collected by these systems in real-time. Therefore, dashboards were also designed to address this issue. A deepened analysis of this instrument is provided in the next section.

(a) (b)

Figure 2.2: Examples of the ATLASCAR2 ADAS.

2.3 Context on dashboards

A dashboard is a HMI. It consists of a panel that enables communication with a machine. On its screen, touch buttons are presented so the operator can easily navigate through its display. They are a considered passive ADAS since they are used mostly to check information and do not have a direct interference in the driving experience. It is important to dive into some aspects of this hardware to understand how it operates.

Displayed information

Dashboards transmit all kinds of data to the driver. Since most information is related to the car's ECU's, dashboards must be connected to the CAN bus. For instance, they can show the car's velocity, autonomy or the state of headlights or blinkers. The study by Figueiredo [8] describes many ways to engage with it. The OBD-II provides a direct access point to the CAN bus. Another option is to get indirect access to it using entertainment systems such as a CD player, USB port or an iPod port. Lastly, dashboards can also use wireless channels such as Wi-Fi or Bluetooth to engage with the CAN bus. Many perform a two-way communication since they can access ECU's to adjust some parameters according to the driver's desire. The air conditioning temperature and radio volume are a few examples.

Dashboards also show ADAS data on their display. They use data provided by sensors and LIDAR's to create virtual representations of the real environment, helping drivers with maneuvers like parking their vehicles. The GPS is a common feature as well. It provides drivers with information related to their location and driving directions. Dashboards can be connected to Web services to verify updated data regarding outside temperature, weather or current traffic conditions.

Warnings

Dashboards provide several warnings to inform the driver of numerous occurrences related to external agents or the car itself. They can come in text form by using pop-ups and notifications. Flashing icons can also be used to indicate some events. An audible indication can follow both, which makes the task of identifying messages more intuitive and direct. Car warning messages can be classified in several ways. According to Laux & Meyes [9], they are divided into three distinct categories:

- Level 1- Information/Advisement: These messages are mainly used to inform and help the user during his driving experience. The information may be ignored since they do not oppose any danger to the driver or the vehicle;
- Level 2- Potential Danger: Occurrences that are dangerous to the driver if corrective action is not taken. A flat tire or low battery messages are two examples;
- Level 3- Urgent or Imminent/Emergency: High-importance warnings that alerts of an imminent threat to the driver's safety. They must be communicated to the driver immediately and should be the most direct possible;

Inputs

Drivers can engage with the HMI in multiple ways. The most common is to use a touchscreen or an in-built mousepad. R. Swette et al [10] studied which tool is the most efficient to access a display between a serial, circular or vertical mousepad or a standard touchscreen. Therefore, the team conducted a survey where participants had to click on a series of icons using these four devices. They considered two aspects: the time each participant took to complete a series of tasks and the attention percentage based on eye awareness. The final result was that the participants performed better while using a touchpad.

Voice commands are also a possibility to be considered. T.A.Ranney et al [11] compared an interface based on voice with one using mainly the touch. The study asked a group of 22 participants to do a set of tasks in the fastest time possible while driving their vehicles. The conclusion was that adding this feature to some functionalities of the dashboard, such as remote phone access, helped the drivers to focus more on the road and reduce the distraction caused by accessing the dashboard by hand.

Eye-Gaze is the most innovative way of communicating with a dashboard. It allows the users to control the dashboard with their eyes by using a set of sensors and cameras. T.Poitschke et al [12] tested this system by comparing it to the touchscreen device. The research concluded that Eye-Gaze commands sometimes were inefficient and did not improve the driver's experience on the road.

Static vs Adaptive interface

The interface can be either static or adaptive [13]. The first has a low degree of customization, and the software developer's visualization mode is standardized. They present the information all at once and can hinder the task of the driver to check information. Adaptive ones, on the other hand, perform a more intelligent display organization and may give greater emphasis to more relevant data, depending on the moment or the driver's preference. Also, they can give the option to reorganize the information modules according to the user demand. They possess multiple windows and numerous tabs, providing a more intelligent data organization.

Solutions on the market

In the context of this work, it is important to analyze the current dashboards in the market. They are a standard widget of numerous car manufacturers worldwide, such as Tesla and Mercedes. The most common practice is to place these electronic devices in a central position on the car's instrument board, between the driver and the passenger seat. A different screen arrangement can be seen depending on the brand or model of the vehicle.

(a) (b)

Figure 2.3: Tesla Model 3 dashboard [14].

The Tesla Model 3 dashboard (Figure 2.3) offers an interactive layout composed of multiple windows that can be closed and opened anytime [14]. The touchscreen allows the users to navigate through the screen, access media, use entertainment features and customize Model 3 to suit their preferences. On the top end of the screen are the time and outside temperature. Meanwhile, touch icons in the bottom bar replace traditional in-car physical buttons. For instance, the driver can control cabin heating temperature, air conditioning and headlights. The main screen is shared by many widgets, from a virtual representation of surrounding objects to information regarding the door locks. In the background, the GPS can be seen.

Similar functionalities can also be found on Mercedes vehicles, as in the Mercedes Benz EQS 450 (Figure 2.4). The screen has an adaptive layout equipped with many windows. The top and bottom tabs are configured similarly as in the prior case [15]. By clicking on the "home" icon, the drivers access a menu with the navigation, phone,

radio, media and features. In both cases, hand-free access is available by selecting the voice option. A parking assistance menu pops up when the user toggles the reverse shift.

Figure 2.4: Mercedes Benz EQS 450 dashboard [15].

2.4 Related work

2.4.1 LAR projects

Several LAR projects have been developed around the ATLASCAR2 during the past years. Therefore, it is essential to cover some assignments that will take a crucial role in the context of this work.

One work aimed to develop an OBD-II interface to monitor the ATLASCAR2 state [16]. To receive and read CAN packets, the author developed an Arduino solution that enabled communication between the OBD-II port of the vehicle and the external devices, like a laptop or flash drive. After establishing the connection with the CAN bus, the author decoded several encrypted messages. As a result, messages with data related to some of the vehicle's internal systems can now be understood and analyzed. By accessing the data from the brakes, steering and accelerator ECU, Figueiredo [8] was able to hack the car systems and construct a remote-control solution.

Another work documented the electric board aboard the ATLASCAR2 [17]. Pereira studied different power solutions that could feed the additional external hardware implemented on the vehicle. The author conducted an experiment where he studied the possibility of withdrawing energy directly from the main battery using a DC-DC converter. The conclusion was that it was possible to do so since the vehicle did not report any malfunction signal on this panel. Correia built an electrical circuit fed by this energy source that powers external hardware on the vehicle [18].

2.4.2 Similar projects

Researchers of the University of Berlin designed an HMI for two vehicles used for autonomous driving research called "Spirit of Berlin" and "Made in Germany" [19]. With it, the team could remotely command the vehicles and check real-time data received from

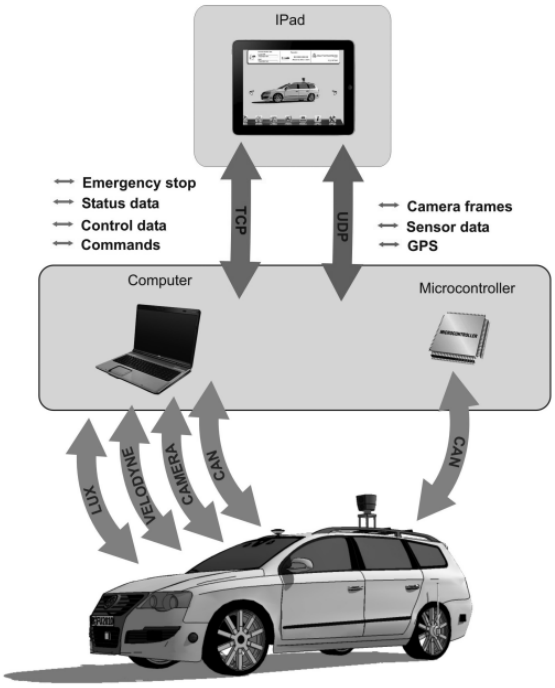


Figure 2.5: "Spirit of Berlin" architecture [19].

the sensors (Figure 2.5). The base platform for this work was a multitouch Apple iPad tablet connected to a server and microcontroller simultaneously. The iPad received data from the microcontroller over Wi-Fi using UDP packets. It acts as an emergency backup system and is directly connected to CAN. Meanwhile, the server sends information from the car's cameras, LIDAR's and ECU's via a TCP/IP connection to the tablet. This component is connected to the vehicle via ethernet to retrieve messages from the CAN bus. This configuration provides a more mistake-free design since the other is operational if one fails. The developers used an iOS programming language based on C called Objective-C to design the display (Figure 2.6). The top bar shows status information regarding time and Wi-Fi connection with the server. The screen shows a 3D Model of the vehicles with a short description of all hardware. At the bottom, a scroll view with various icons gives access to all the implemented features.

A team from the Politecnico de Milano created an interactive dashboard interface for an autonomous shuttle (Figure 2.7) [20]. The objective was to design an HMI that not only served to command the shuttle but also could be used to visualize the states of the various ECU's. The inputs used to control the vehicle could come from either a joystick, the HMI, or the autonomous driving logic. A PeakCan USB hardware was used to communicate with the shuttle's CAN bus. Robotic Operating System (ROS) network connected all the systems and each system had an associated node that published its information on a topic. An additional node that subscribed to all topics was created to accommodate the interface designed in a Simulink-based CAN interface. From this node, CAN packets can be sent to the shuttle's bus to control the vehicle remotely.

Figure 2.6: "Spirit of Berlin" dashboard [19].

Figure 2.7: "Politecnico de Milano" autonmous shuttle [20].

2.5 Summary

Dashboards present a variety of information to the driver. This information can come from web connections, the vehicle ECU's, or implemented ADAS. In the context of this work, it is also essential to study how this device works, what functionalities must be added and how the information modules should be organized and displayed. For this purpose, other dashboards were analyzed.

Intentionally blank page.

Chapter 3

Experimental infrastructure

The following chapter details the test platform of this dissertation. Section 3.1 presents the main ATLASCAR2 characteristics, explains the current power architecture and describes the equipped hardware. Section 3.2 specifies the infrastructure used to communicate with the car's CAN bus. Section 3.3 and 3.4 overviews the software, libraries, and tools used.

3.1 The ATLASCAR2 vehicle

3.1.1 Characteristics

Figure 3.1: Mitsubishi i-MiEV 2015.

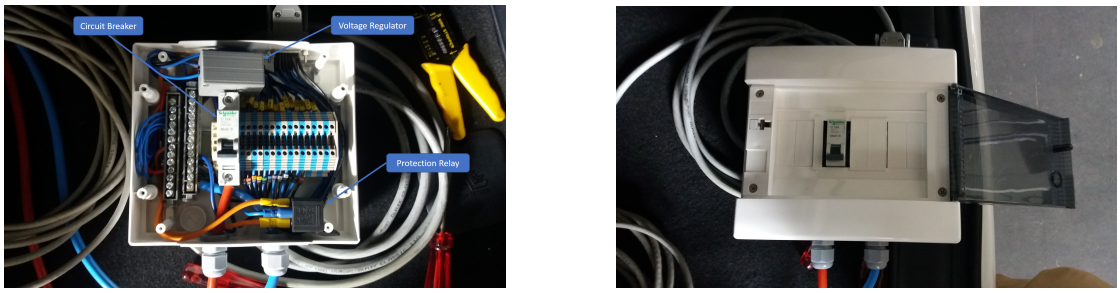
As mentioned in Section 1.1, the ATLASCAR2 is a Mitsubishi i-MiEV 2015 (Figure 3.1). This fully electric vehicle has a 16 kWh lithium-ion battery with a rated voltage of 330 V Direct Current (DC) [21]. It has a 160 km cruising range and can be charged with a 230 V Alternating Current (AC) outlet, taking up to 10 hours to fully recharge. By using a fast-charging station, the stoppage time is reduced up to 30 minutes. The 49 kW electric motor that powers the rear wheels is connected to the main power supply. The car is also equipped with a lead 12 V battery. Its two main functions are to feed small gadgets, such as audio systems, headlights and windshield wipers, and protect the 16 kWh battery through its isolating relays. The main electric source powers the small battery through a DC-DC converter when the ignition is on [22].

3.1.2 Onboard power distribution

ATLASCAR2 is a vehicle for studies on autonomous driving and for that reason, many sensors, LIDAR and cameras have been added to it. Thus, researchers had to equip it with a small circuit powered by the 12 V DC battery at the vehicle's front end. Since the electric board is placed in the vehicle's rear, cables were added to connect these two components. The orange wire corresponds to the positive pole and the blue one to the negative pole (Figure 3.2).

The circuit is protected by a 16 A DC circuit breaker that acts as a switch for the board. Then, the current flows through a relay which guarantees that the car's gadgets are powered only when the main battery is on. A MC7812SCT 12 V/1A regulator was added to the switch input to protect the board from voltage fluctuations. This device is stored in a custom 3D printed case inside the board.

Initially, a DB 15 female connector was added. It plugs a cable with twelve connectors that comes from the vehicle's hood and is used for two purposes: to power the embedded gadgets and to activate the relay. Further work added a pair of female banana connectors on the board's top and two hardware plugs on each side (Figure 3.3).



(a) (b)

Figure 3.2: ATLASCAR2 original electric board [18].

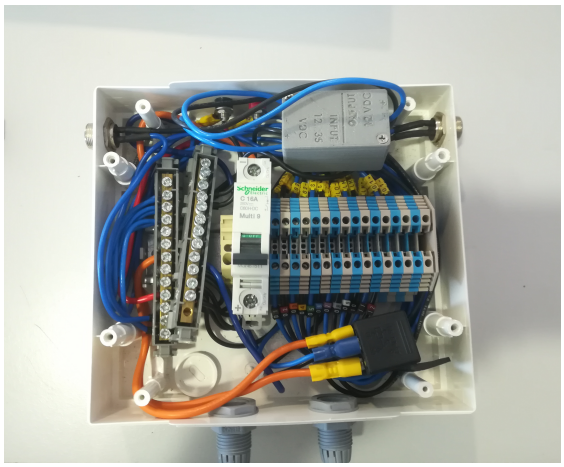


Figure 3.3: Eletric board expansion.

3.1.3 Auxiliary equipment

The added devices are connected to a processing unit. This computer is the master brain of the car since all the programs addressed to the vehicle hardware are stored in it. The computer needs a 230 V AC plug and the only two power sources inside the vehicle were the 330 V DC main battery and the small 12 V DC battery. Therefore, an Uninterruptible Power Supply (UPS) was installed in the vehicle's trunk (Figure 3.4) alongside the central processing unit. When the UPS is turned on, it powers the central computer, booting all ATLASCAR2 systems simultaneously.

Figure 3.4: ATLASCAR2 processing unit and UPS

APC Smart-UPS 1500 VA

The APC Smart-UPS 1500 VA (Figure 3.5) is the central unit's power supply. It has a 230 V AC nominal output voltage and a 1500 VA maximum power output. The technical specifications of the APC Smart-UPS are shown in table 3.1.

Figure 3.5: APC Smart-UPS 1500 VA [23].

Table 3.1: APC Smart-UPS technical specifications [23].

Main Input Voltage	230 V AC
Main Output Voltage	230 V AC
Power Output	1500 VA
Batteries	Lead-Acid
Weigth	24.09 kg
Outlets	8

Battery MZ690402

The ATLASCAR2 lead battery (Figure 3.6) has 12 V DC and is situated on the car's hood. The main characteristics are specified in Table 3.2.

Figure 3.6: Battery MZ690402 [24].

Table 3.2: Battery MZ690402 technical specifications [24].

Model	MZ690402
Voltage	12 V
Maximum current	300 A
Capacity	35 Ah

3.2 Accessing the CAN bus

3.2.1 CANalyze

Having a reliable source of information is crucial for this thesis. As mentioned in section 2.1, ECU information is present in the CAN bus of the vehicle. To access that data, a CANalyzer was used (Figure 3.7). This hardware is an open-source, native CAN interface for Linux that uses SocketCan to link the CAN bus with other devices. By enabling communication with outside sources, it allows reading and recording of all relevant messages.

To set up the network, the CANalyzer should be linked with the OBD-II port placed next to the foot pedals of the vehicle (Figure 3.8). A standard OBD-II to DB9 cable must be used to ensure proper transmission of messages. A USB 2.0 type A to type B cable is also required to connect this gadget to the computer.

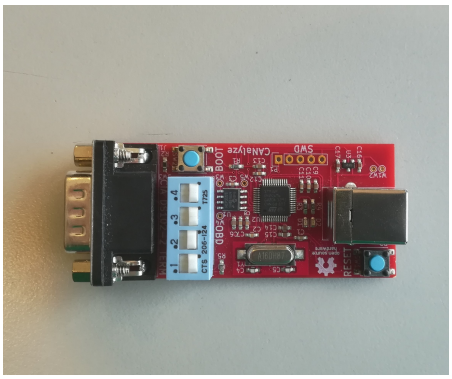


Figure 3.7: CANalyzer.

Figure 3.8: OBD-II Port on ATLASCAR2.

3.2.2 SocketCan

The SocketCan is a set of open-source CAN drivers for Linux. By providing a socket interface built upon the Linux network layer, it allows communication between

the CANalyzer, the device described in section 3.2.1, and the CAN bus. After setting up a socket, this package binds it to an interface like in a TCP/IP network. Once bound, the socket is ready to be used. The SocketCAN offers three types of CAN interfaces [25]:

- Native: CAN interfaces associated with real hardware, such as a USB-CAN adapter, like in this dissertation. Usually, they are entitled "can0", "can1," and so forth;
- Virtual: Interfaces that are not associated with any real device. They are named "vcan," followed by the identification number;
- SLCAN: Serial interface compatible with the Slcan-Interface. They are associated with the Slcan ASCII protocol and are typically named "slcan," following the rule as in prior cases;

For this work, only the native and virtual interfaces were needed. The first was used to communicate with the vehicle to get real data on the car's current state and the second served to conduct tests and experiments in a virtual environment without really accessing the ATLASCAR2.

Depending on the type of interface that will be linked, an initial configuration is required. For the native interface, a configuration of the CAN interface with the required bitrate must be done:

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

A virtual CAN driver for testing purposes can be loaded and created in Linux with the commands below. Loading the modprobe module may be needed in case the driver is still not loaded:

```
$ sudo modprobe vcan

$ sudo ip link add dev vcan0 type vcan

$ sudo ip link set up vcan0
```

The command "ifconfig" is used to verify if the connection is set up:

```
$ ifconfig can0
```

In either case, the presented code should enable interaction between a CAN bus and a Linux environment. However, to read and process the received messages, an additional SocketCan package has to be installed: can-utils.

can-utils

The can-utils packate is a series of SocketCan utilities that help programmers with CAN communication inside the Linux operating system. These tools can be accessed using a standard Linux terminal. With it, users can create, receive and send custom CAN packets to the vehicle [26]. The main tools used in the context of this work were:

- cansend: Sends a CAN message to the network. It is used mainly for experience purposes in a virtual environment since the internal security system of ATLAS-CAR2 does not allow external messages to be sent into the CAN bus of the vehicle. The frames are composed of a three-digit identifier followed by a cardinal digit and 8 bytes of data, as in the following example:

  $ `cansend can0 ABC#1122334455667788`

  In this case the message identifier is 0xABC and the data bytes are 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77 and 0x88. Futher analyses on CAN messages will be given in Section 5.2;

- cangen: Generates random CAN packets. It is used mostly in virtual CAN interfaces for testing since the user has no control over the messages sent to the network;

  $ `cangen can0`

- candump: Prints all received messages by the CAN interface to the terminal (Table 3.3). As explained before, the correct bitrate must be set while connecting with the CAN bus. In this case, the network operates with a 50000 bitrate, making it difficult for investigators to identify any particular CAN message. The data presented at the console is very cluttered, making it hard to trace by the human eye. Therefore, a Linux command-line tool was used to help filter these messages. The `grep` filter searches a file for a particular pattern of characters and displays all lines that contain that pattern in the terminal (Table 3.4). In this case, inputting a specific identifier to the grep tool makes it possible to print the desired set of CAN messages;

  $ `candump can0`

175	[8]	12	A3	E0	19	FF	56	AF	A5
184	[8]	16	00	00	00	0D	16	12	12
235	[8]	01	23	71	19	14	57	23	BE
457	[8]	45	58	00	65	14	D3	54	14
579	[8]	12	A6	04	12	14	5A	92	0A
987	[8]	A4	03	87	14	14	00	00	00
1A3	[8]	E4	00	45	1F	00	56	21	42
2E5	[8]	12	A3	23	55	14	00	10	3A
4FA	[8]	12	1A	D3	11	10	56	73	BA

Table 3.3: Dumping all CAN messages.

235	[8]	01	23	71	19	14	57	1F	BE
235	[8]	00	23	71	19	14	57	1F	BE
235	[8]	01	23	71	19	14	57	1F	BE
235	[8]	00	23	71	19	14	57	1F	BE
235	[8]	01	23	71	19	14	57	1F	BE
235	[8]	01	23	71	19	14	57	1F	BE
235	[8]	00	23	71	19	14	57	1F	BE
235	[8]	01	23	71	19	14	57	1F	BE
235	[8]	00	23	71	19	14	57	1F	BE

Table 3.4: Dumping all CAN messages using grep tool.

- cansniffer: It helps with message identification by only displaying changing CAN packets (Table 3.6) . For instance, if the driver presses the brake, opens a door, or turns on the air-conditioner, cansniffer will print to the console the messages with the 0x208, 0x424 and 0x3A4 ID's;

  $ `cansniffer can0`

175	[8]	12	A3	E0	19	FF	56	AF	A5
184	[8]	16	00	00	00	0D	16	12	12
235	[8]	01	23	71	19	14	57	23	BE
457	[8]	45	58	00	65	14	D3	54	14
579	[8]	12	A6	04	12	14	5A	92	0A
621	[8]	65	E3	56	AA	00	56	09	F3
987	[8]	A4	03	87	14	14	00	00	00
1A3	[8]	E4	00	45	1F	00	56	21	42
2E5	[8]	12	A3	23	55	14	00	10	3A
4FA	[8]	12	1A	D3	11	10	56	73	BA

Table 3.5: Dumping all CAN messages.

175	[8]	12	A3	E0	19	FF	56	AF	A5
184	[8]	16	00	00	00	0D	16	12	12
208	[8]	01	33	15	A5	BB	75	24	BE
235	[8]	01	23	71	19	14	57	23	BE
424	[8]	22	32	11	75	AA	BB	A2	F1
579	[8]	12	A6	04	12	14	5A	92	0A
682	[8]	01	23	71	19	14	57	1F	BE
987	[8]	A4	03	87	14	14	00	00	00
3A4	[8]	00	03	03	05	24	99	64	2A
4FA	[8]	12	AA	D3	11	10	56	73	BA

Table 3.6: Dumping all CAN messages using cansniffer.

3.3 ROS - Robotic Operating System

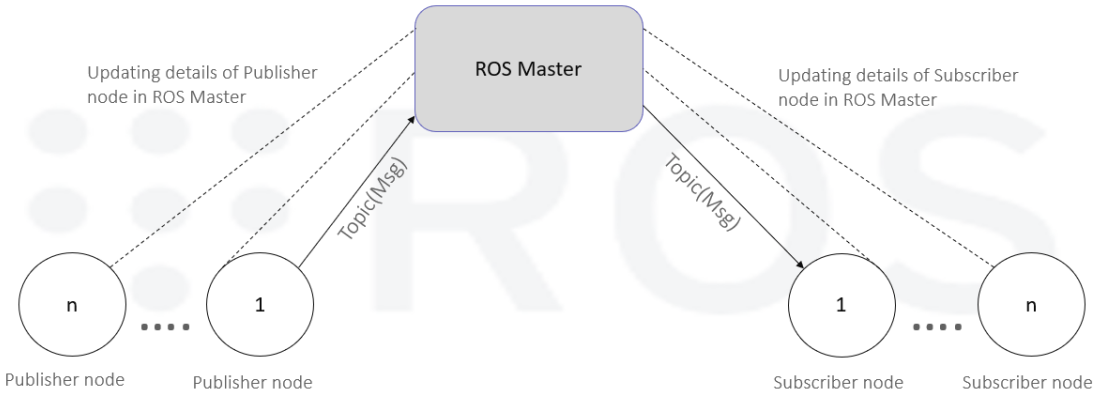


Figure 3.9: ROS architecture [27].

Over the last years, LAR´s researchers have implemented a communication architecture based on ROS at ATLASCAR2 (Figure 3.9). This open-sourced framework was created in 2007 and provides a common platform for robotic applications. By easily handling large volumes of data, it allows smart communication between multiple systems and offers the opportunity to create complex and efficient networks of information. Currently, ROS has over 3000 collections of tools and libraries written in C++, Python, and LISP, mainly supported by Linux [27].

Three ROS network objects were used for this work: Nodes, Topics and Messages. The following topics will explain these concepts, followed by a description of Rqt and RosLaunch, two ROS packages used during this project.

Nodes

Nodes are structures that perform computation. They send messages to topics that are then subscribed by other nodes to communicate. After a node subscribes to a topic, all messages are delivered to the node that requests it. When a message is received, they process the data as intended. These two entities usually are not aware of each other's existence.

Topics

Topics are modules in which nodes can exchange messages. They work on a publisher-subscriber logic as they serve as a communication channel between different nodes. A topic is defined by its name and a message definition, which is the data structure of the information nodes send. All messages on the same topic must be of the same data type and there can be multiple subscribers to a topic at once.

Messages

A message consists of a data structure that carries information. They can be composed of other messages, arrays or standard primitive types such as integers, floating points and booleans and can be either standard or custom. The structure of the custom message is defined in a text file stored in the msg subdirectory of a package.

Rqt

Rqt is a Graphical User Interface (GUI) framework designed to support ROS users with various tools in the form of plug-ins. With it, users can trace the messages being sent, providing them with a useful tool for monitoring malfunctions in complex networks. Rqt can also create virtual nodes that publish messages to a certain topic at a specific rate. A schematic representation of all running nodes and topics is provided as well.

Roslaunch

Roslaunch is a tool for quickly launching multiple ROS nodes locally. It includes the option to respawn processes that have already died. Launch files are of the format .launch extension and use a specific XML format that specifies the parameters to set and the nodes to launch.

3.4 Kivy

The chosen software to conduct this work had to meet some specifications. The software had to be supported in Linux and open-source. In addition, it must be compatible with python and have a broad set of online documentation. Kivy was the chosen software to develop the dashboard display.

Kivy is a multitouch application development software supported on Linux, Windows, OS X, Android, iOS, or Raspberry Pi platforms. It is based on an object-oriented format and gives users the option to add all the core functionalities of a standard application to their project. The software provides a wide range of tools to work with

since it supports input devices such as a keyboard or a mouse and allows the addition of external gadgets like cameras and GPS tracking systems. It contains a graphic library using only OpenGL ES 2 based on Vertex Buffer Object [28].

A Kivy application is divided into two files. The primary coding is done in a standard python script (Code Listing 3.1). Thus, the user can access multiple libraries that ease the development of applications. An independent text file written in Kv language (Code Listing 3.2) is responsible for customizing the app layout. It manages all app objects, such as text boxes or buttons and offers the possibility to specify the size, position, or events associated with them. Each widget has an associated ID. Kivy collects all the IDs and places them in an instance of the app class. Thus, the programmer can call to the script any object and update its features anytime. Also, Kivy supports music tracks, custom letter fonts and videos by sorting them in the app's directory.

Kivy has a vast collection of add-ons and frameworks to improve the GUI app's performance and design quality. For this dissertation, two were used: KivyMD and Kivy-Garden.

```
1
2  import kivy
3  from kivy.app import App
4
5  class MyDashboard(Screen):
6
7      #Change the text of the label
8      def change_label(self):
9          self.ids.label.text = 'Button Pressed'
10
11 class ExampleApp(App):
12
13     def build (self):
14         Builder.load.file('KvFile.kv') #Import Kv File
15         return MyDashboard() #Inicialize App
16
17 if __name__ == 'main'
18     ExampleApp().run()
```

Code Listing 3.1: Main script example.

```
1
2  <screen>:
3
4      Button:  #Create left button
5          id:button
6          size_hint:0.5,0.5
7          pos:0.25,0.25
8          text: 'Example Button'
9          on_press: root.change_label
10
11     Label:   #Create right label
12         id:label
13         size_hint: 0.5, 0.5
14         pos:0.75,0.75
15         text: 'Example Label'
```

Code Listing 3.2: Kv file example.

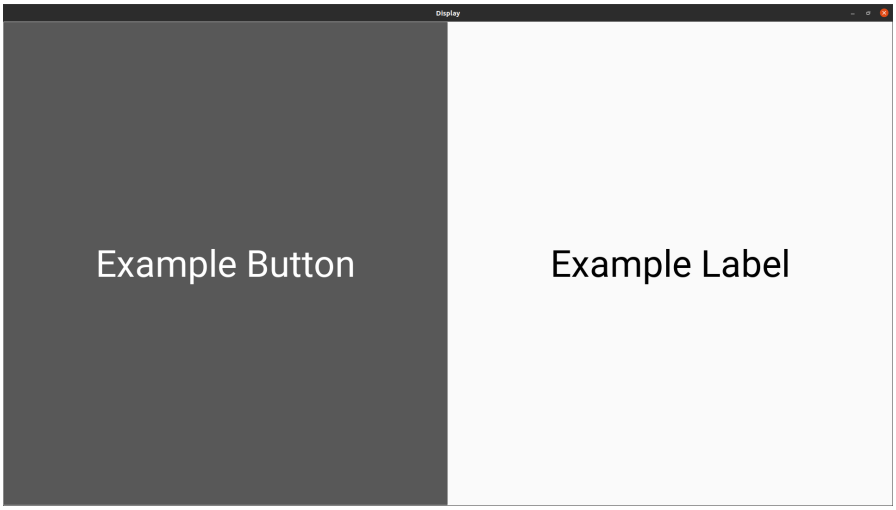
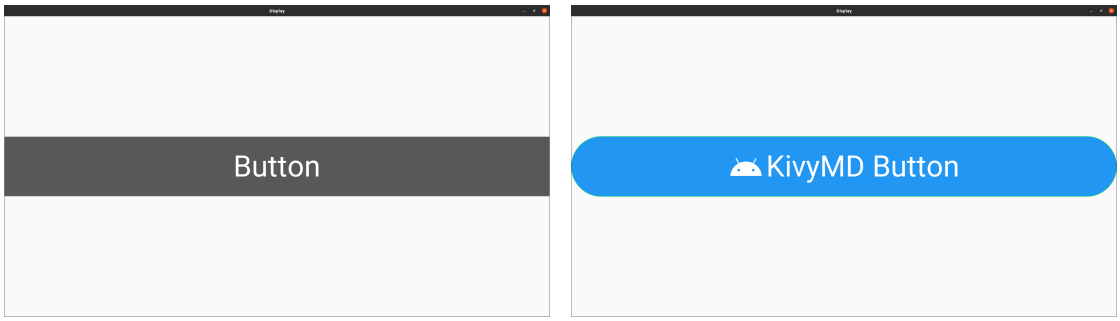


Figure 3.10: Example app.

KivyMD

The standard Kivy library provides a set of unappealing widgets that lack visual design. Therefore, Kivy developers created a library built on top called KivyMD. It gathers a collection of Material Design widgets that provides the GUI with a more attractive look, improving the visual outline of the project (Figure 3.11). In addition, this expansion offers programmers various objects, from appealing buttons to custom tabs [29].



(a) (b)

Figure 3.11: Comparison between Kivy and KivyMD buttons.

Kivy-Garden

Garden is a project used to centralize add-ons for Kivy. It gathers a series of projects created by standard users in one place, as developers can access it and upload their work there. Furthermore, this package equips the application with more sophisticated tools using a straightforward approach. All widgets are available in the Kivy-Garden Github repository and can be imported into any project [30].

3.5 Summary

This chapter describes the hardware and software infrastructure used during this work. CANalyze was used to access CAN messages by connecting it with the OBD-II. The CanSocket is responsible for establishing communication between the CAN bus and external computers using this hardware. The operating system was Linux OS as Python was the opted language to code all applications. The library used for the display design was Kivy and the communication framework picked was ROS.

Intentionally blank page.

Chapter 4

Solution to extend the processing unit autonomy

The first objective of this dissertation is to install a new power solution for the computational system of the ATLASCAR2. Currently, the vehicle control unit is only operational for 15 minutes and by adding the dashboard, this time can be decreased even more. Therefore, the intervention must increase the time span on which these components are available. This chapter details the approach used during this work, explains all added components and describes the electric board's final setup.

4.1 Solution overview

As described earlier, the vehicle's control unit needs a 230 V AC plug. However, the only compatible source available onboard the vehicle is the UPS which only provides energy for a short period.

Devices on the car are connected to two types of sources. On the one hand, some devices can withdraw energy from in-built hardware aboard the ATLASCAR2, like the 330 V DC or the 12 V DC lead battery. On the other, they could be fed by an external device not connected to the vehicle and has to be recharged from time to time, such as an UPS. As mentioned in section 3.1.2, the car possesses a 12 V DC electric board connected to the lead battery that feeds embedded systems such as cameras and sensors. In turn, the main battery is linked to the lead battery and is charged when the ignition is on. One option was to install another UPS more powerful than the old one. However, this solution was ruled out since it is expensive. The second was to convert the 330 V DC to 230 V AC, but no compatible inverter was found. Therefore, transforming the current from the main battery would require a DC-DC and a DC-AC conversion units. This solution was ruled out since it was not affordable and the electrical signal quality might be compromised. Considering all facts, using the 12 V electric board presents the best option to develop this work.

The proposed solution (Figure 4.1) sets on expanding the current board as multiple components had to be acquired and several circuit components rearranged. To convert the 12 V DC into a 230 V AC current, a DC-AC inverter was used. The hardware will withdraw energy from the circuit as it powers the UPS. This device will then feed the computer. This UPS setup allows that, if a battery malfunction occurs, the user will be able to adequately shut down the control unit and save all work. A security fuse will

protect the circuit from high currents and a circuit breaker acts as a switch. The box that held the circuit was cluttered and lacked the proper space to add the instruments required. For that reason, the montage had to be removed from the old box as a new one was acquired. The new board was also designed to accommodate future expansions, providing additional plugs for external hardware. The banana plugs and the electrical buses used to feed the positive and negative pols were also replaced.

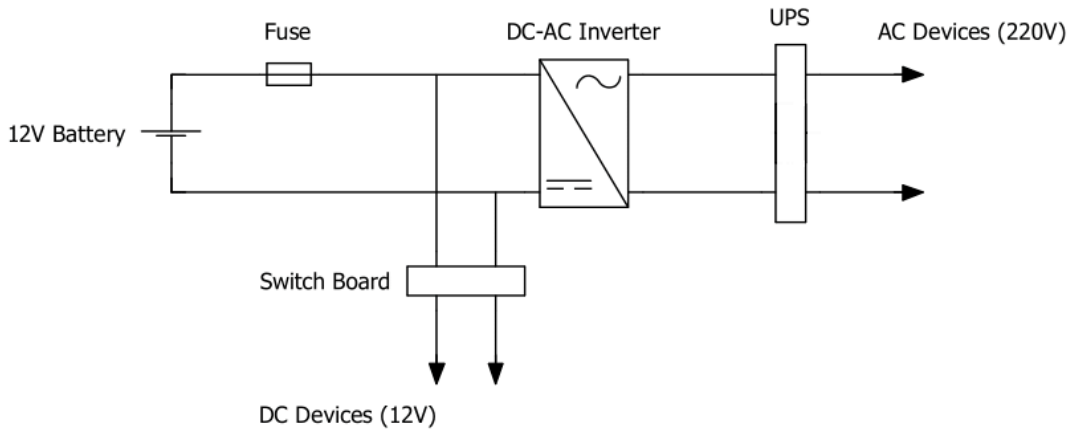


Figure 4.1: Circuit representation of the solution.

4.2 Hardware

4.2.1 Fundamentation

The components were chosen according to the board characteristics. Thus, some previous conditions had to be previewed in order for the circuit to work properly. The first component to be analyzed is the inverter. As mentioned in 3.1.3, the UPS has a 1500 W maximum output. Therefore, it must generate the same amount of power as the UPS. However, the top load will never be achieved as the inverter is slowly recharging this device. Nevertheless, the estimation presented in Table 4.1 overestimates the working conditions of this hardware. The UPS feeds the central unit of the vehicle that consumes 600 W [31]. The dashboard was also considered since it will be added in future works. The dashboard's power output was chosen according to standard devices on the market.

Considering all facts, the power output of the selected inverter had to be 1000 W. The second component was the fuse which was chosen according to the calculations present in the table (4.2) using the following expression:

$$P = V \times I \tag{4.1}$$

The fuse mounts in series with the components of the board. Therefore, the added currents of each hardware are the minimal current that the component has to support. Usually, when the hardware turns on, it consumes more power than usual. Thus, the

fourth column is an estimation value for when all equipment powers on simultaneously. Work from Correia [8] specifies all the characteristics of the old board components.

Table 4.1: Used calculations for the inverter design.

Hardware	Quantity	Power (W)	Power Surge (W)
Nexus P-2308H4/HR4	1	600	680
Dashboard	1	35	36
Total		635	716

Table 4.2: Used calculations for the fuse design.

Hardware	Quantity	Current (A)	Current Surge (A)
Sick LMS 151	2	0.67	2
Sick LD-MRS 400001	1	0.67	2
Camera	1	0.67	2
Point Grey ZBR2-PGEHD-16	1	0.67	2
Inverter	1	84	92
Total		87,35	102

4.2.2 Components definition

This section describes the components added to the original circuit. Most were acquired through Universidade de Aveiro's partners. Others were reused from past projects or were already available in LAR.

Inverter

A DC-AC 1000 W inverter from RS Components (Figure 4.2.2) was bought to convert the signal. To avoid damaging the UPS, a crucial component for the ATLASCAR2, the inverter has a pure sine output wave. Also, LED indicators on the front panel lights up when a failure occurs. Table 4.3 shows the main specification of this hardware. A pair of wires equipped with a ring terminal connect the inverter to the primary circuit.

Figure 4.2: Inverter for the power expantion [32].

Table 4.3: Inverter technical specifications [32].

Rated power	1000 W
Surge power	2000 W
Input voltage	12 V
Output voltage	230 V
Output waveform	Pure sine

Fuse and Fuse holder

A Jasco fuse (Figure 4.3) protects against spikes in the current and can support a rated current up to 100 A. It has 22x58 mm, a 500 V rated Voltage and a 100 kA breaking capacity. A Legrand fuse holder (Figure 4.4) has the function of mounting the fuse into the circuit. While open, no current passes through it. The component has a voltage of 500 V and a 100 kA breaking capacity.

Figure 4.3: Fuse [33].

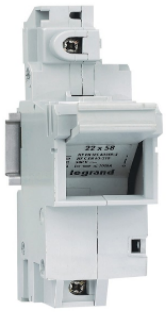


Figure 4.4: Fuse holder [34].

Cable glands

The cable glands (Figure 4.5) were attached to secure the end of the electrical cable to the box. Also, they act as a sealing device to protect all electrical equipment inside the board from outside agents. The board has two pairs of glands. An IP68 - PG 11 pair secured the wires connected to the inverter and an IP68 - PG 13.5 supported the battery cables. PG stands for the gland's internal diameter.

Figure 4.5: Cable glands [35].

Plugs

Five 3-pin GX 16 plugs feed the external devices as they operate at 12 V. While the female connectors (Figure 4.6) were added to the box, wires from the camera and sensors had to be attached to the new male adaptors (Figure 4.7).

Figure 4.6: Female plug [36].

Figure 4.7: Male plug [36].

Buses box

The previous grey buses that distributed the power supply through all devices lacked proper protection since a short circuit could happen if the buses got in touch with one another. The new electric bus (Figure 4.8) has two drawers. Each has five 5.3 mm diameter entrances and four with 7.3 mm. The top drawer is connected to the positive pole and the bottom to the negative.

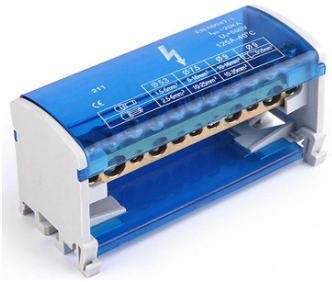


Figure 4.8: Buses box [37].

Circuit breaker

The component that acts as the switch is a YRO circuit breaker (Figure 4.9). It has two poles, a 125 A nominal current and a 550 V rated voltage.

Figure 4.9: Circuit breaker [38].

Board

A new 400x400x130 box (Figure 4.10) was acquired. It has two gutters. Two screws hold the gutters on each side and can be withdrawn anytime. The top gutter will accommodate the electric buses and electrical bornes. The fuse box and circuit breakers will be on the bottom gutter.

Figure 4.10: Box for the new board [39].

4.3 Board installation

The new board had to be adapted to support the new circuit. Two pairs of holes were drilled to accommodate the cable glands. One with 11.5 mm in diameter and the other with 13.5 mm. The first is used to feed the inverter by attaching the negative and positive borne to their correspondent electrical bus. The second was used for wiring the 12 V lead battery with the fuse holder. 16 mm holes were added to accommodate the 3-pin GX 16 female plug. An 8 mm hole was drilled to add an existing connector from the old box. Since they were new, the electric cables of the new connectors had to be welded. The plugs were isolated with rubber lids. A trapeze-shaped hole was drilled to accommodate the DB 15 female connector (Figure 4.11).

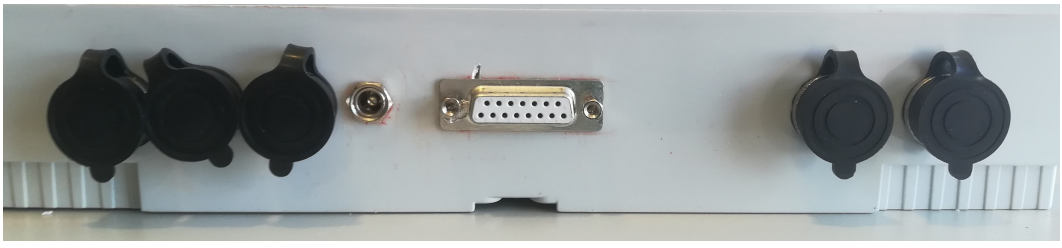


Figure 4.11: Side view of the board.

A fuse is linked to the power source to limit the current inside the circuit to 100 A. The circuit breaker that acts mainly as a switch is cabled to the fuse. This component is directly connected to the inverter and the relay. The negative pole of the inverter

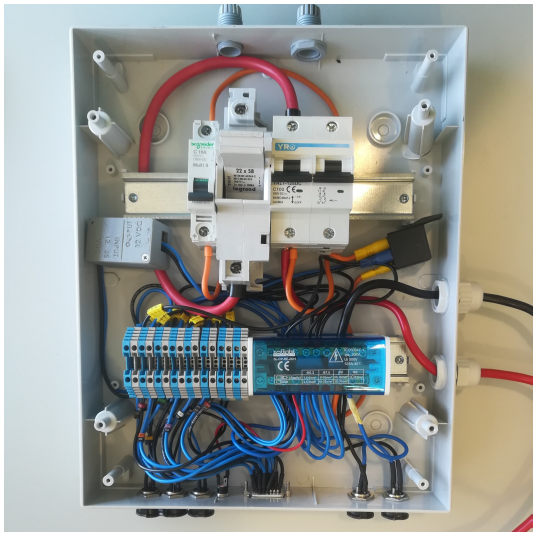


Figure 4.12: New circuit.

Figure 4.13: Closed Board.

was wired to the negative bus. Since the new circuit preserved all critical components from the initial installation, the configuration stayed the same from this point forward. The box opening now presents three switches, one for the 16 A circuit breaker that acts as a switch just for the sensors and cameras, the fuse and the 120 A general circuit breaker. The operator must use the inverter's in-built button to turn off the equipment and only power the outside gadgets. While the 3-pin GX 16 female plugs were added to the box, the wires from the camera and sensors had to be attached with the new male adaptors. This thesis also aims to improve the security and quality of the circuit already implemented. For that reason, the original wires were resized and equipped with tips. The final result can be seen in Figures 4.12 and 4.13. A schematics of the implemented circuit is provided in Figures 4.14 and 4.15. The instructions to use the new board are detailed in appendix A.1.

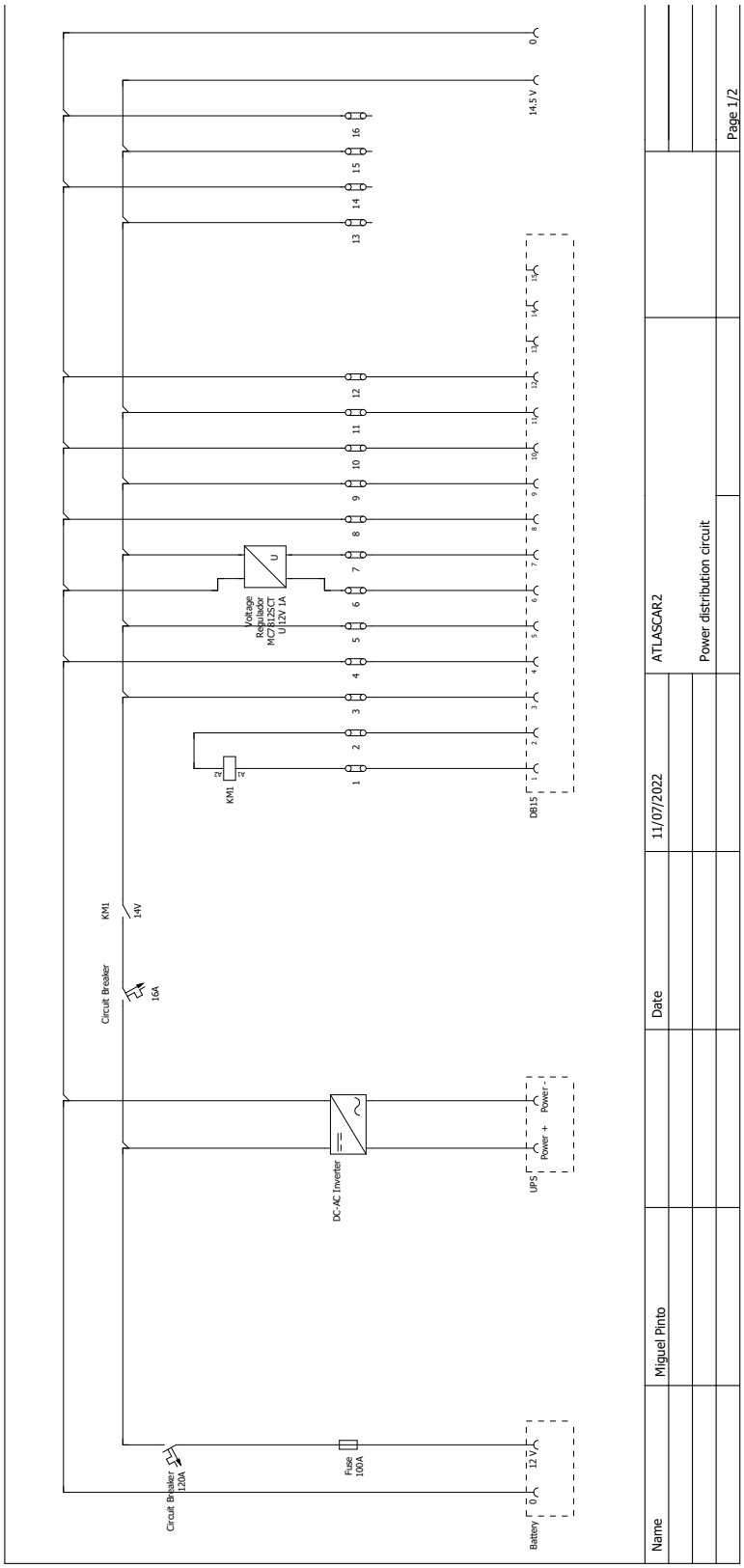


Figure 4.14: Power distribution board.

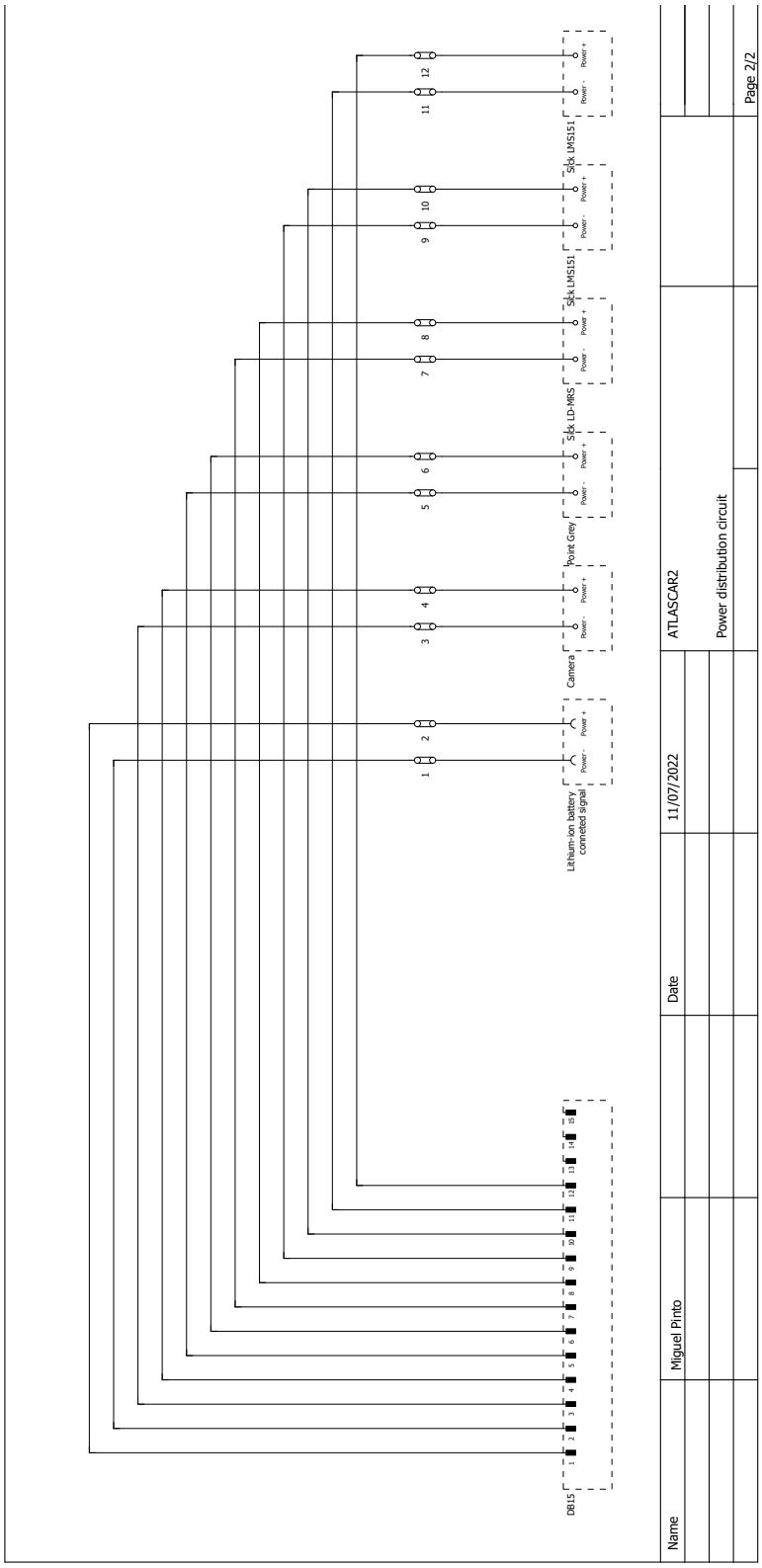


Figure 4.15: Power connections.

4.4 Expected autonomy

The first objective of this work is to increase the ATLASCAR2 processing unit power autonomy. Therefore, it is essential to understand how this intervention affected the vehicle's power supply. The 12 V battery has 35 Ah and the circuit needs 87.35 A to feed all systems. The time the equipment is available is calculated by dividing the consumption by the current. For that reason, with the current configuration, if the ignition is off the board enables the hardware to work for 24 minutes (Table 4.4).

Table 4.4: Time span of the UPS while using the main battery.

Voltage (V)	Capacity (Ah)	Current (A)	Time (minutes)
12	35	87.35	24

If the ignition is on, the main battery recharges the lead battery. Therefore, the autonomy of the central unit becomes the same as the vehicle. To calculate the expected total autonomy, three different consumption scenarios were assumed: One where the vehicle was travelling in a city with an average velocity of 40 km/h, the second on a highway at 110 km/h and the third in a midterm condition where the car is cruising at 75 km/h. By multiplying the average speed by the consumption, the power is obtained. Then add the maximum power of the equipment connected to the UPS, which is 0.635 kW, and divide the value by the battery capacity, which is 16 kW/h. The final result is present in Tables 4.5 and 4.6. Information regarding car consumption can be checked on [40].

Table 4.5: Relation between consumption and the driving scenario.

Situation	Average speed (km/h)	Consumption (kWh/km)	Power (kW)
City	40	0.107	4.28
Highway	110	0.197	21.67
Mid-term	75	0.145	10.86

Table 4.6: Autonomy with and without the UPS turned on.

	Power (kW)	Autonomy (hours)
Without UPS	4.28	3.75
	21.67	0.74
	10.86	1.47
With UPS	4.92	3.25
	22.31	0.71
	11.51	1.39

4.5 Summary

In this chapter, the new ATLASCAR2 electric board components were detailed and the final configuration was explained. Since the present work aimed to increase the autonomy of the central unit, predictions of the expected time span were made. The calculations assumed that the lead and main battery were new and followed the manufacturer's specifications. That is not the case since, at the time of this work, the vehicle had six years. Therefore, it is hard to know the current condition of those hardwares. Nevertheless, when the ignition is not switched, the autonomy of the processing unit is increased by 10 minutes. On the road, assuming that the ATLASCAR2 is a research vehicle mainly tested in urban areas, it is expected that the new configuration will provide power for 3 hours and 15 minutes which means a loss of 13%, or 30 minutes, in the car's full autonomy. It is also important to refer that the systems should not be turned on all at once since it could produce a 102 A current which burns the fuse.

Intentionally blank page.

Chapter 5

Car status monitoring

The information displayed on the dashboard comes from the CAN bus of the ATLASCAR2. For this work, it is relevant to understand how the data is created and sent. Thus, section 5.1 gives an introduction to the CAN protocol. Section 5.2 addresses message identification and section 5.3 analyses the software application developed to receive and process CAN packets.

5.1 Understanding CAN protocol

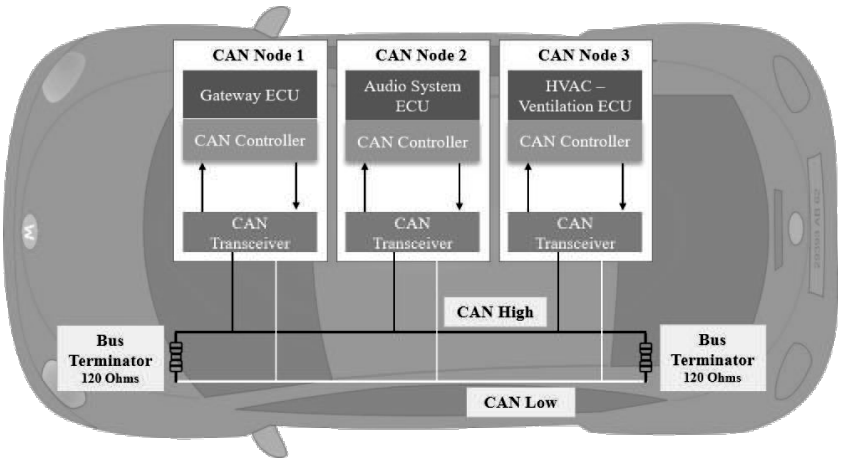


Figure 5.1: CAN bus representation [41].

First, to understand where the information displayed on the dashboard comes from, it is important to cover the background of the CAN protocol. As mentioned in Section 2.1, the CAN bus allows communication between different vehicle ECU's. It consists of two wires: CAN high (CAN H) and CAN low (CAN L). A 120 Ω resistor terminates each wire (Figure 5.1). The ECU's generate messages using a voltage differential created between the two lines. On the one hand, when the bus is clear, both cables carry 2.5V and control units assume a bit-value of "1". On the other hand, if an ECU broadcasts a message, the CAN high line goes to 3.75 V and the CAN low drops to 1.25 V, creating a 2.5 V voltage differential. This differential has a bit-value of "0". The "1" bit is called dominant and the "0" bit is called recessive. This process creates binary messages

broadcasted to the bus [40].

The created CAN data frames ensure the message exchange between ECU's. They are composed of seven fields (Figure 5.2) [41]:

- SOF– Bit that synchronizes the nodes on a bus after being idle. They indicate the CAN data frame beginning;
- ID- Unique 11-bit data frame identifier;
- RTR– The single remote transmission request (RTR) is a bit that indicates if a node is sending data to or requesting data from an ECU. It is dominant when information is required from another node;
- Control- The control field indicates the message's frame type and amount of data. It is composed of 6 bits;
- Data- Field where the data is sent. Up to 64 bits of information may be transmitted;
- CRC- A 16-bit group used for error detection. The Cyclic Redundancy Check contains the number of bits transmitted and checks the message's data integrity;
- ACK- It is a 2-bit which acknowledges that the CRC found no issues with the data;
- EOF- Stands for "End of Frame."It signals the end of transmission;

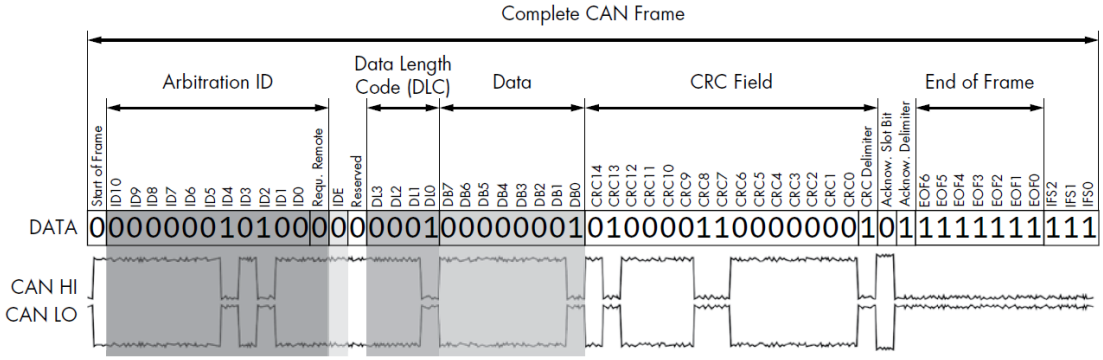


Figure 5.2: Format of standard CAN packets [42].

5.2 Decoding the Mitsubishi i-MiEV CAN mensages

Decoding CAN messages is highly significant for this thesis since all displayed information comes from this step. First, the CANalyzer must be connected to the OBD-II port to receive messages on a laptop. Then, `can-utils` enables CAN packets to be analyzed on the computer terminal. Each message contains seven fields. However, this library only prints to the terminal the messages ID, the total number of received bytes and the data field (Table 5.1). This syntax allows for easier comprehension by the user

since it sends the most relevant information. Each printed number is expressed in a hexadecimal digit. A hexadecimal number represents 4 bits and therefore, each 8-bit byte is a 2-digit hex number.

Table 5.1: Terminology of can-utils messages

Message Identifier	Total Bytes	Data Bytes							
ID	TB	B0	B1	B2	B3	B4	B5	B6	B7

CAN messages can have up to 8 bytes of data. The most common way to identify each byte is to name it with a capital "B" followed by the number of his position, from 0 to 7. Meanwhile, bits are named using a lower-case "b". The rightmost bit is bit 0 and the left-hand one is bit 7.

Message identifiers are utterly unknown to the general public since car manufacturers do not provide any information. Thus the most effective method to decode IDs is to do reverse engineering. The idea is to activate a specific device or sensor of the vehicle and then inspect the network for message changes. For example, if the operator wants to determine the ID of the door locks, he should close one and sees if this action kicks off changes in CAN frame data fields. Since ECU's are constantly sending messages to the bus, the `grep` and `sniffer` tools function as a filter that allows only the changing packets to be printed to the terminal. Still, reverse engineering is an arduous process that may take a while. Nevertheless, the Mitsubishi i-MiEV is commonly used by investigators worldwide for autonomous driving projects, and much work has been done in this field. Some message IDs and data frames are described in [8], [16] and [43].

The decoding messages process unveiled much information regarding the car's ECU's. The main parameters displayed on the dashboard are described next.

- **Autonomy**: Information regarding vehicle autonomy is present in the messages with ID 0x346 and 0x374. The first expresses the autonomy in km and can be read directly from Byte B7 of that message. Meanwhile, the second holds the data in percentage on Byte B1 and follows equation 5.1:

$$Autonomy = \frac{B1 - 10}{2} \quad [\%] \tag{5.1}$$

- **Battery status**: Message 0x373 holds the general status of the battery. The current value ranges from -164.18 to 76.54 and is obtained using equation 5.2. If the current is negative, that means that the car is charging. Equation 5.3 holds the value of the Voltage which varies in the interval [343.2, 389.7];

$$Battery\ Current = \frac{(B2 \times 256) + (B3 - 128) \times 256}{100} \quad [A] \tag{5.2}$$

$$Battery\ Voltage = \frac{B4 \times 256 + B5}{100} \quad [V] \tag{5.3}$$

- **Speed and total traveled distance**: The message with the identifier 0x412 holds information regarding the vehicle's speed and total traveled distance. The first is read directly in byte B1. The second follows equation 5.4;

$$Total\ km = (B2 \times 65536) + (B3 \times 256) + B4 \ \ [km] \tag{5.4}$$

- **Shift position**: Byte B0 of the message with ID 0x418 holds data related to the shift position (Table 5.2);

Table 5.2: Byte value acording to each shift position.

B0	Shift Position
0x44	Drive
0x4E	Neutral
0x50	Park
0x52	Reverse

- **Seat belt, windshield, blinkers, doors, lights**: The message with the ID 0x424 transmist information related to the instrument panel and some other vehicle sensors (Table 5.3). Bit b7 from Byte B0 tracks all open doors excluding the one from the driver. The same happens regarding the belts warning;

Table 5.3: Relation between variables of the instrument panel and the message 0x424.

Byte	Bit	Variable
B0	b6	Driver's belt warning
	b7	Passengers‘ belts warning
B1	b0	Right blinker
	b1	Left blinker
	b3	Front windshield
	b4	Back windshield
	b5	Medium lights
	b6	Maximum lights
B2	b6	Driver's door
	b7	Passengers' doors

- **Air conditioner**: All information regarding the air conditioner buttons of the vehicle is present in the message with the ID 0x3A4 (Table 5.4). The A.C. temperature and intensity vary from 0 to 15;

Table 5.4: Relation between the air conditioner buttons and the message 0x3A4.

Byte	Bit	Variable
B0	b0-b3	A.C. temperature
	b5	Max. temperature button
	b6	Recirculation button
	b7	Max. intensity button
B1	b0-b3	A.C. Intensity

During this process, other messages were identified. In this work's context, they are not relevant since the data do not pay any interest to the driver. Nevertheless, they are mentioned in the following list to help future work on the dashboard.

- **Igniniton key**: Information regarding the state of the ignition key is found in the byte B0 of the message with the ID 0x101 (Table 5.5);

Table 5.5: Byte value acording to the ignition key state.

B0	Ignition State
0x00	ON
0x04	OFF

- **Breake pedal's position**: The brake pedal position is measured in percentage and can be read in bytes B2 and B3 of message 0x208 following equation 5.5. When pressed, byte B4 of the message 0x231 changes from 0x00 to 0x02;

$$Break\ Pedal\ Position = \frac{(B2 \times 256 + B3) - 24576}{640} \times 100 \ \ [\%] \tag{5.5}$$

- **Accelarator pedal's position**: The information is present in message with ID 0x210 on byte B2;

$$Accelarator\ Pedal\ Position = \frac{B2}{250} \times 100 \ \ [\%] \tag{5.6}$$

- **Steering wheel angle**: Message with the ID 0x236 holds the Steering Wheel Angle on bytes B0 and B1;

$$Steering\ Angle = \frac{(B0 \times 256 + B1) - 4096}{2} \ \ [^\circ] \tag{5.7}$$

- **Eletric motor**: Information regarding some aspects of the electric motor can be found on the message 0x298. The motor's temperature can be accessed by using equation 5.8. Equation 5.9 holds the electric motor revolutions;

$$Eletric\ Motor\ Tempereture = B3 - 40 \ \ [^\circ C] \tag{5.8}$$

$$Eletric\ Motor\ Revolutions = B6 \times 256 - 1000 \ \ [rpm] \tag{5.9}$$

5.3 CAN software application

The developed python program receives raw packets from the CAN bus and then processes them in order to be understood by the display software. Therefore, it is the cornerstone of the dashboard since it feeds the display with information.

Initially, the script checks for any received messages on the bus. If any is available, the program reads it and then associates each field with a separate variable. The result

is three distinct variables: One called "id" that stores the message identification in hexadecimal form; A integer variable named "nd" that keeps the number of data bytes; And finally, a byte array called "data" that holds the data field. This array is then processed to create two variables: one that stores each byte and another that stores the byte converted into bites. This step is essential since some messages are read directly in the byte and others in the bit. They are named the same as before. A byte is named with a capital "B" followed by the number of his position and a bit with a lower-case "b" where the rightmost bit is bit 0 and the left-hand one is bit 7.

The next step is identifying the messages according to the findings of section 5.2. The program compares the message ID with every known identifier. It grabs the information from the desired byte or bit if it finds a match. In the event that the read information is an integer, the value has to be converted from hexadecimal to decimal. For instance, if the program receives the message: " 412 [8] 23 1E 45 23 55 A3 FF 00". First, it will compare the ID 0x412 with every known message ID and then the script identifies the data it wants to access. According to byte B1, the message signals that the car travels at 30 km/h.

The program stores the recently discovered data in a variable named according to each parameter. By predefinition, every variable is set to "0" until its value is updated. The variable is overwritten if the script receives a message with more recent information. Table 5.6 shows all the parameters used for this work. It is essential to mention that the final output of the shift position is a string with "Drive, Neutral, Park or Reverse ". The air conditioner intensity and temperature variables are an integer that can range from 0 to 15.

Table 5.6: Data used for this work.

Message ID	Description	Measurement unit
0x346	Autonomy	km
0x373	Battery current	A
0x374	Autonomy	%
0x412	Velocity	km/h
	Total covered distance	km
0x418	Shift position	-
0x424	Car instruments	ON/OFF
0x3A4	A.C. buttons	ON/OFF
	A.C. intensity and temperature	-

Finally, the script publishes the processed messages to a topic explained in the next chapter and helps the operator to check the messages being sent by creating a dictionary that is printed on the terminal. Figure 5.3 provides a general diagram of the script. The script and the instructions to use it are present in appendix A.2.

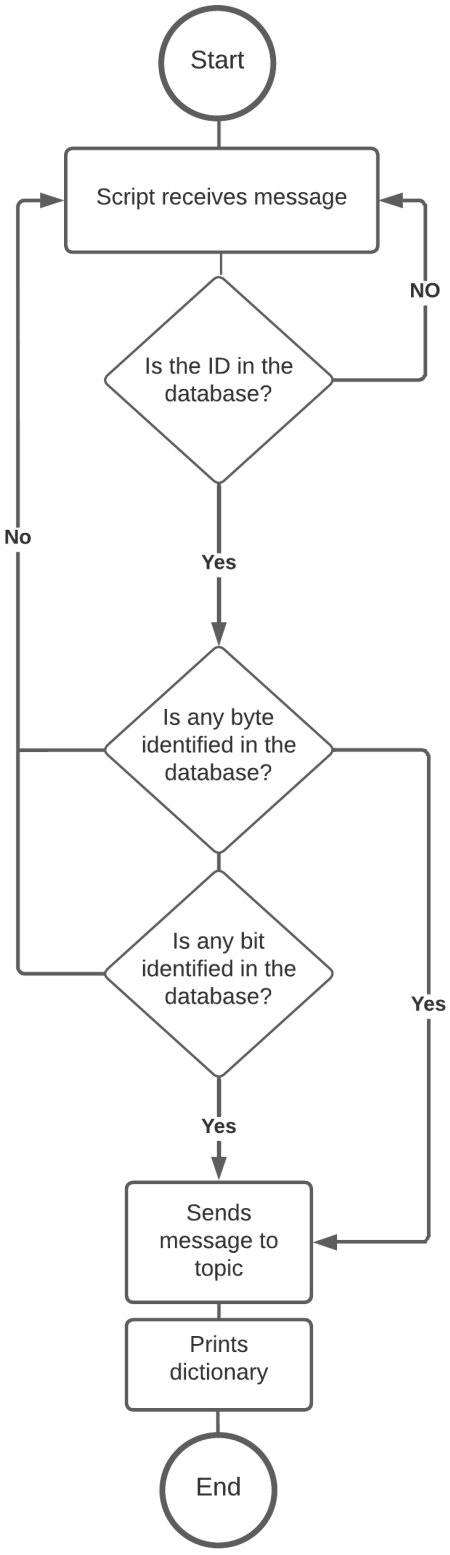


Figure 5.3: Driagram of the created Python script.

5.4 Summary

The most relevant messages on the ATLASCAR2 CAN bus are now available due to the developed application. In total, the following twenty parameters are now ready to be sent to a ROS network that supports the software of the dashboard:

- Autonomy in percentage and km;
- Battery current;
- Velocity;
- Total covered distance;
- Shift position;
- Driver and passengers' belt warnings;
- Driver and passengers' door warnings;
- Front and back windshield;
- Medium and maximum lights;
- Temperature, intensity, recirculation and maximum intensity buttons;
- Left and right blinkers;

Chapter 6

Dashboard - Implementation and testing

Chapter 6 explains the architecture of the newly assembled network that allows CAN raw packets to be transformed into readable data. The final solution is presented and all features and functionalities embedded in it are also described. The dashboard functionality was evaluated with a test described in the final section.

6.1 ROS architecture

6.1.1 Overview

The next step is to create an infrastructure that allows the data to flow from the ATLASCAR2 CAN bus to the dashboard display. Since the vehicle already had a ROS network installed in the car's central unit, this work will follow the same policy. ROS was designed with distributed computing in mind. Therefore, this architecture will allow other projects already embedded in the car to be included in the dashboard infrastructure. Also, it will provide easy access to the developed work for future assignments on the vehicle.

The network is composed of three nodes and two topics. The `can_node` was created using the script mentioned in Section 5.3 and it is responsible for reading and processing the CAN messages. Therefore, it is a vital structure since it feeds information to the network. The node publishes data in a topic named `can_messages` which uses a custom message named `can_msgs`, created to facilitate communication between the different modules. The topic is then subscribed by two nodes: `warning_node` and `dashboard_node`. The first is responsible for generating the dashboard alerts. Their state is available in a separate topic called `warning_messages`, which also uses a custom message named `warning_msgs`. Finally, the `dashboard_node` subscribes at the same time the `can_messages` and `warning_messages` topics. It is responsible for gathering all parameters from the network and displaying them in a dynamic layout created with the Kivy library. Figure 6.1 shows a node graph of the network. The custom messages were constructed according to the sent parameters in each node. If a node sends twenty variables, the custom message has the same number of fields, each designed for a specific variable. The code of the dashboard and instructions to launch it are present in Appendix A.3.

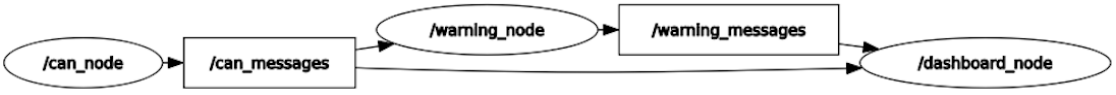


Figure 6.1: Node graph of the network.

6.1.2 Warning structure

The warning node is a python script responsible for managing the dashboard warnings, pop-ups and layout changes. It subscribes to the topic `can_messages`, processes the received information, and then publishes it into a new topic.

The script only updates its variables when the value of any parameter changes. Then it organizes the information by storing the data from each field in a separate variable. The generation process of warnings is explained in Table 6.1. The script compares the value of the variables with a database and then checks their state. The node creates eight warnings in total, which are sent in the `warning_msgs` custom message. While initializing, the variables are all set to false as default.

Table 6.1: Description of the warnings created.

Message field	Description
Doors	The warning is enabled when the driver or passengers' doors variables are activated and the car velocity is higher than 10.
Belts	It is activated when the variables of the driver or passengers' doors are activated and the car velocity is higher than 10.
Charging	The alert is activated if the vehicle is being charged and whether the car is moving or not in "Park".
Reverse	When the shift position is in reverse.
Low autonomy	Warning that is enabled when the vehicle has less than 15%.
Save energy	If the vehicle range is less than 30 km and the air conditioner is turned on.
Velocity	-
Close proximity	-

This work will open the door to future expansions on the dashboard display and network. Therefore, two extra variables were created. One is linked to a velocity warning and the other to a proximity warning. They are set to false by default and can only be activated using the `Rqt` tool. The idea behind the first is to access an online map and verify if the cruising speed exceeds the actual speed limit of the road the vehicle is currently in. For the proximity warning, it is expected that future works access the ADAS systems of the ATLASCAR2 and analyze external agents. The script activates an alert message if their position presents an impending danger.

The final result is a set of booleans variables that will be used on the display software.

6.1.3 Display strucuture

The `dashboard_node` is a python script responsible for receiving the information from the entire network and then showing it on a dynamic display. It processes variables from either the `can_node` or the `warning_node`.

The display was built on Kivy. The library provides a broad set of objects to work with, from standard labels, icons and cards to pop-ups, side-bars, and multi-screens. When the script receives data, it stores the parameters in separate variables and associates them with different objects. Icons were associated with boolean variables. They only can have two values: true or false. If a variable is true, then the icon flashes on the screen. If not, the icon disappears. The same happens with warnings, pop-us and layout changes. They are only activated when the state of the corresponding variable is true. Labels were used to show integers and strings. Table 6.2 shows all associations made between the variables and the created objects, which will be covered in the next section.

Table 6.2: Relation between the created objets and the received variables.

Variable	Data type	Object/ Functionality
Autonomy (Km)	Int.	Range label
Battery (%)	Bool.	Battery label and icon
Battery current	Bool.	Battery icon
Velocity	Int.	Velocity label
Total covered distance	Int	Odometer label
Shift position	String	Shift icon
Front windshield	Bool.	Windshield icon
Back windshield	Bool.	Windshield icon
Medium lights	Bool.	Headlights icon
Maximum lights	Bool.	Headlights icon
Right blinker	Bool.	Right arrow icon
Left blinker	Bool.	Left arrow icon
Passengers' doors	Bool.	Open door icon
Driver's door	Bool.	Open door icon
A.C. maximum button	Bool.	Maximum icon
A.C. recirculation button	Bool.	Recirculation icon
A.C. intensity	Int.	AC intensity label
A.C. temperature	Int.	AC temperature label
Doors	Bool.	Warning module
Belts	Bool.	Warning module
Charging	Bool.	Warning module
Reverse (warning)	Bool.	Change to reverse layout
Low autonomy	Bool.	Pop-up
Save energy	Bool.	Pop-up
Velocity (warning)	Bool.	Velocity module
Close proximity	Bool.	Change to close proximity layout

6.2 Display

6.2.1 Layouts

The display comprises three layouts: The main layout, the reverse layout and the close proximity layout (Figure 6.2). All three are interactive, meaning the user can access the displayed content by hand, using a touch screen. The first is the one displayed by default and has all the modules of information (Figure 6.3). The others are activated when the respective warning is true and are only used in certain events. They help the driver to focus on the task at hand by removing unuseful information and embedding ADAS on the dashboard display. On the one hand, when the reverse layout is activated, the display swipes to the left. Conversely, it swipes to the right when the close proximity layout is activated. The display shifts to the default position if their respective warnings are false. All have two bars. On the top, time and date are shown, followed by a side tab that allows the user to access a settings menu. The bottom bar is used to access the GPS window that can be toggled anytime.

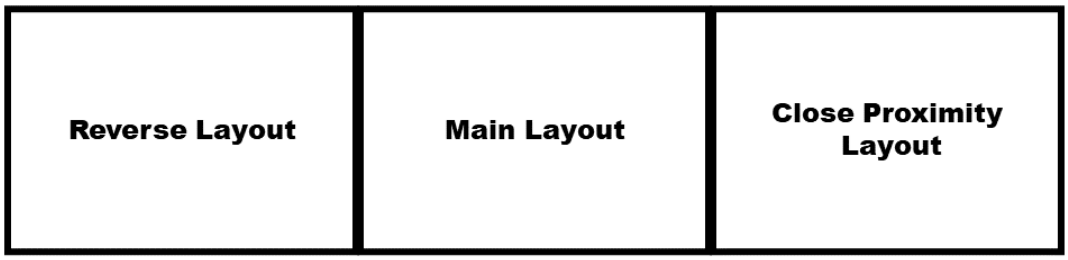


Figure 6.2: Representation of the three layouts.

The reverse layout (Figure 6.4) is only toggled when the car is in reverse. The top and bottom bars do not suffer any changes. However, the modules of information get rearranged. On the top left is given information about the battery autonomy and on the right, a sensor measures the distance between the vehicle and rear obstacles. The middle module shows the vehicle velocity. The particular feature of this module is the live feed from a rear camera.

The close proximity layout (Figure 6.5) follows the same configuration as the reverse. The main change is that the rear camera live feed gives place to a visual representation of the external world which helps the driver to dodge any obstacles on his way. This layout is activated when the close proximity warning is true. As in prior cases, the rear distance sensors, the live camera feed, and the visual representation are only demonstrations for future works, as they are not connected to any hardware.

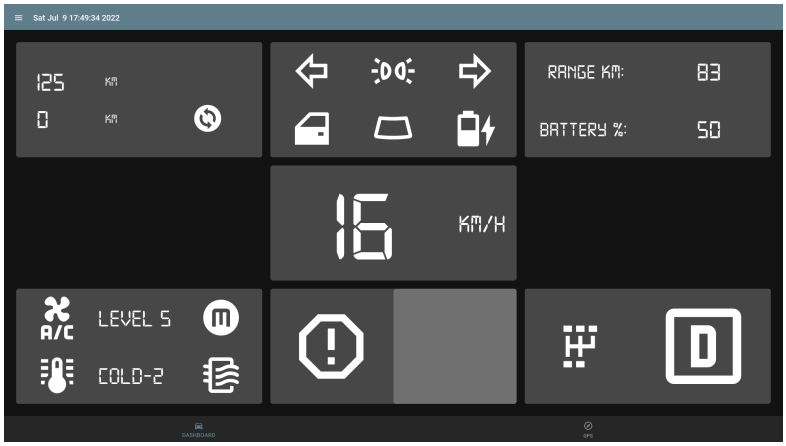


Figure 6.3: Main layout.

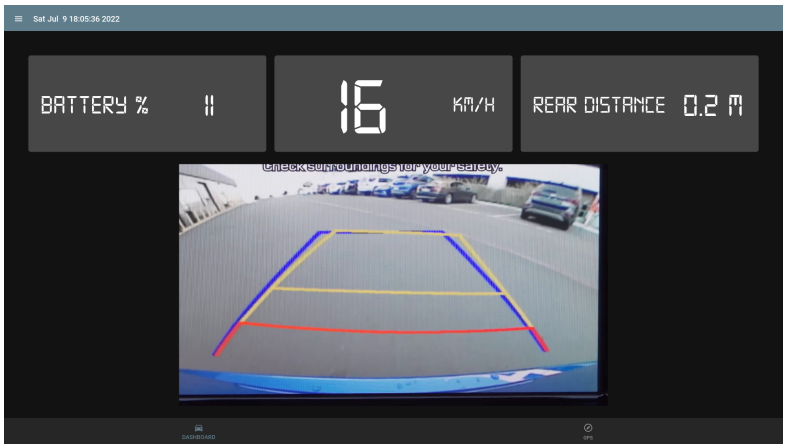


Figure 6.4: Reverse layout.

Figure 6.5: Close proximity layout.

6.2.2 Information modules and features

The display is equiped with ten information modules and features (Figure 6.6) which will be explained next.

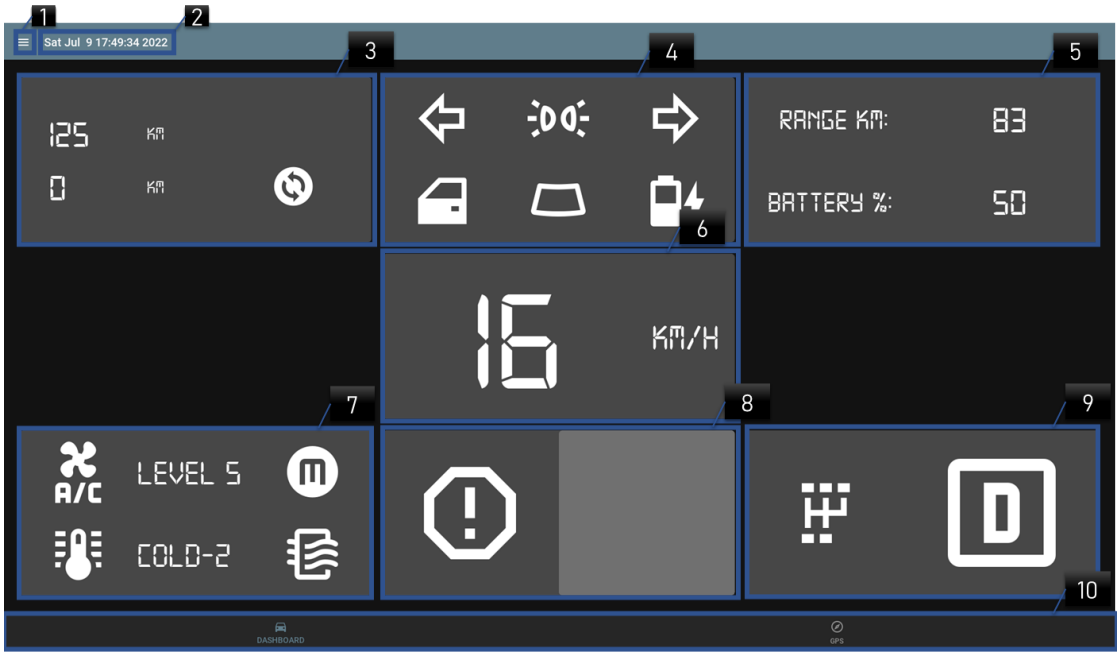


Figure 6.6: Modules of information.

1. **Settings**

   When the settings icon is triggered, a menu appears from the left side of the screen (Figure 6.8). Two slide bars are presented in it (Figure 6.7). The first allows the user to disable the warning and alert notifications of the dashboards. The second changes the color scheme from night (Figure 6.9) to day mode (Figure 6.10).

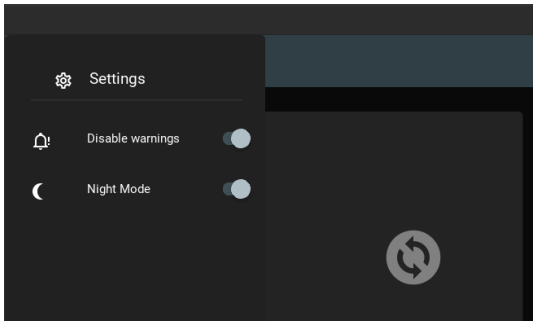


Figure 6.7: Setting slide buttons.

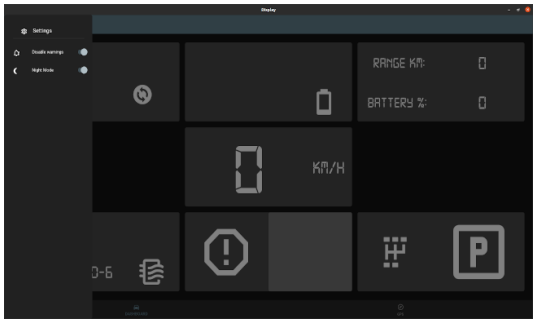


Figure 6.8: Settings tab.

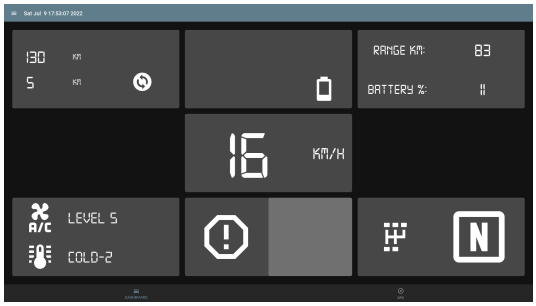


Figure 6.9: Night mode.

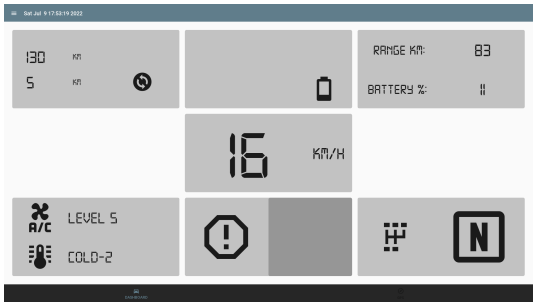


Figure 6.10: Day mode.

2. **Date and time**

   The top bar also provides updated data and time (Figure 6.11). The information is in the form of the day of the week, followed by month, time and year.

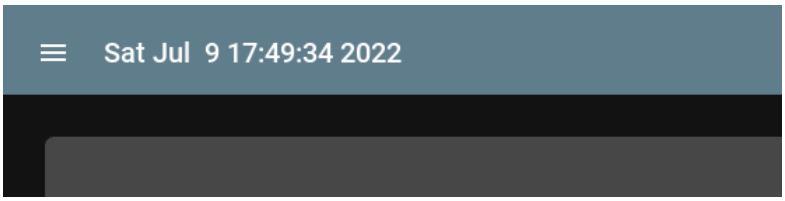


Figure 6.11: Date and time.

3. **Odometer**

   The odometer module (Figure 6.12) has two features. The top one is a standard odometer that measures the total traveled distance by the car. The second is a relative odometer which calculates the distance the car has traveled since the driver triggered the reset button. Every time it is pushed, the relative odometer sets to zero.

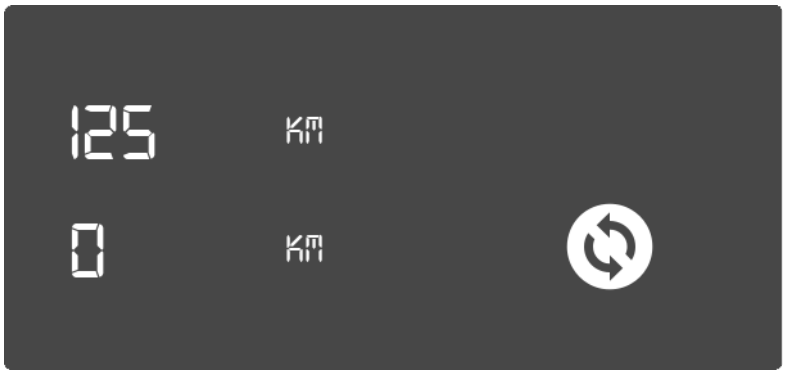


Figure 6.12: Odometer module.

4. **Car instruments**

   The fourth module shows the state of car instruments (Figure 6.13). When one is toggled, an icon lights up on the screen.

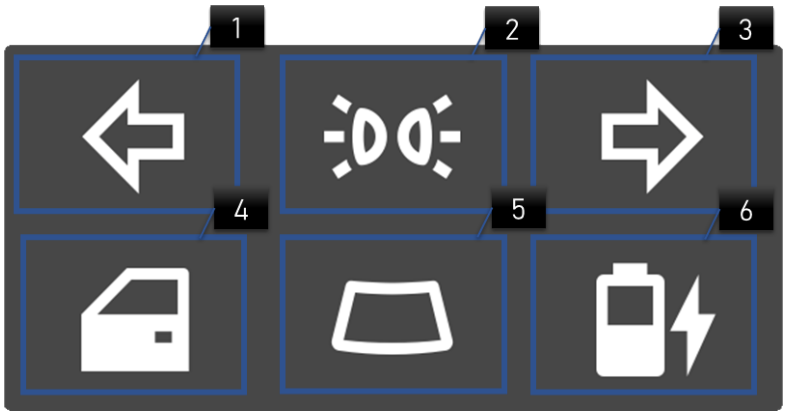


Figure 6.13: Intruments module.

   1. Left Arrow: Indicates if the left blinker is on;
   2. Headlights: Toggled when the medium or maximum lights are lighted up;
   3. Right Arrow: Indicates if the right blinker is on;
   4. Open Door: Flashes when the driver or passengers' doors are not closed;
   5. Windshield: Toggled when the front or back windshield is toggled;
   6. Battery: Dynamic representation of the battery autonomy. The icon updates according to the percentage of battery it has left. When the battery is charging, a visual indication is also provided;

5. **Range and battery autonomy**

   The fifth module (Figure 6.14) informs the driver of the current battery autonomy and tells the expected range of the vehicle.

Figure 6.14: Range and battery autonomy module.

6. **Velocity**

   The sixth module (Figure 6.15) informs the driver of the current cruising velocity of the vehicle. It is associated with the velocity warning. When the warning is true, the module's background color turns red followed by an audio indication, which signals that the driver is overspeeding (Figure 6.16).

Figure 6.15: Velocity module.

Figure 6.16: Overspeeding warning.

7. **Air conditioner**

   The driver can verify information regarding the air conditioner in the seventh module (Figure 6.17). The intensity level is presented on the top and on the bottom the temperature. The two icons on the right flash when the maximum intensity button or the recirculation button are toggled, respectively.

Figure 6.17: Air conditioner module.

8. **Warnings**

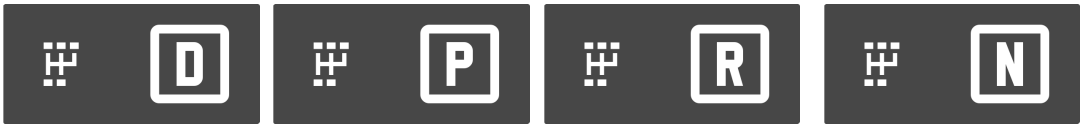
   The warnings module (Figure 6.18) notifies the driver of events that can present an impending danger to himself or the car. They can not be averted and the driver must act to turn them off. When the alerts are activated, an audio indication is toggled, the right side of the module flashes red, and a text indicates the occurrence.

(a) (b) (c)

Figure 6.18: Warnings module.

9. **Shift position**

   The final module (Figure 6.19) indicates the current position of the shift. It has four positions: Drive, Park, Reverse and Neutral.



(a) (b) (c) (d)

Figure 6.19: Shift position module.

10. **Navigation bar**

    The navigation bar allows the driver to switch between the dashboard window and the GPS window (Figure 6.20). The first contains all the information modules and the second is equipped with a map downloaded from a kivy-garden Github repository [44]. The main purpose of this feature is for future works to embed a GPS on this window.

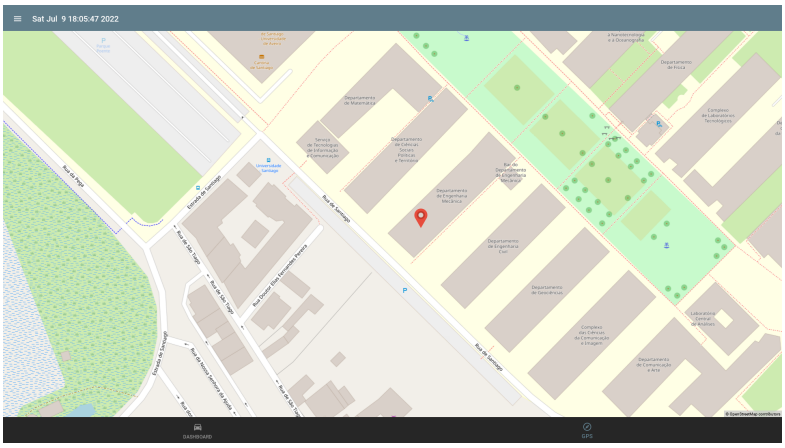


Figure 6.20: GPS Screen.

6.2.3 Pop-ups

A pop-up is a window designed to grab the user's attention to communicate certain events with him. Usually, they are followed by a sound notification and a small text explaining the occurrence (Figure 6.21). Unlike warnings, pop-ups can be ignored since they only notify minor occurrences. In this work, two were used. The first notifies the driver when the autonomy drops below 15% (Figure 6.22). The second advises the driver to turn off the A.C. when the range is less than 30 km to extend the range of the vehicle (Figure 6.23). To close them, the driver must press the "close" button on the right corner of the pop-up.

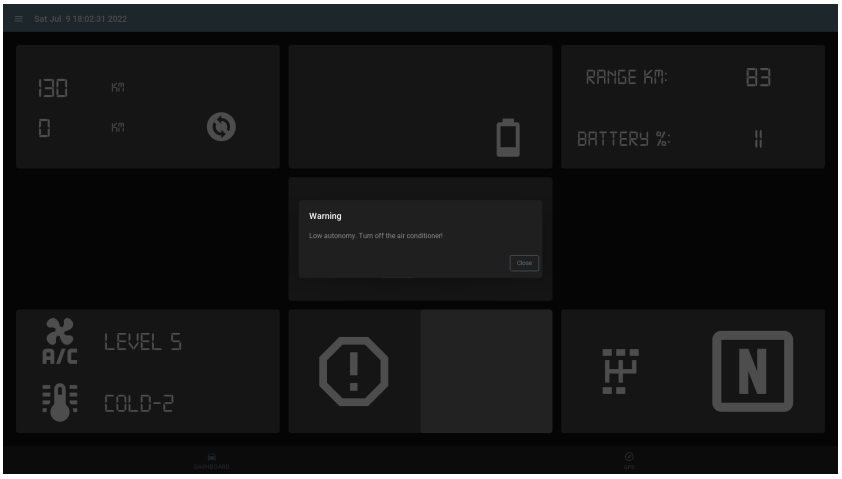


Figure 6.21: Pop-up.

Figure 6.22: Low autonomy pop-up.

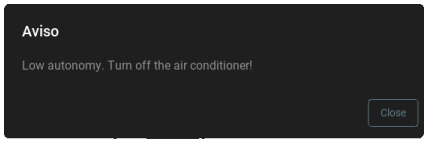


Figure 6.23: A.C. pop-up.

6.3 Dashboad field test

In order to properly analyze the dashboard functionality, the solution had to be tested on the road. Therefore, a research test was conducted. It consisted of a lap around Aveiro's University (Figure 6.24), where five participants tested the impact of the device on the ATLASCAR2 driving experience.

Figure 6.24: Test circuit.

The dashboard was displayed on a monitor placed on the driver's seat (Figure 6.25). First, each participant would drive from point A to point B. Then he would perform a set of manoeuvres, like parking the car or driving without the safety belt, and finally return to the starting point. At the end of the circuit, the participants had to answer an inquiry to give their opinion about the dashboard's performance.

(a) (b)

Figure 6.25: Dashboard field test.

The inquiry was composed of six questions. The first five were yes or no questions. On the last, the participants had to choose a number from one to six, in which one was the lowest grade and six the highest. The questions and their results are presented on Table 6.3 and Table 6.4. The inquiry can be accessed on the following link:

`https://docs.google.com/forms/d/e/1FAIpQLScARdpvv7KBnHNx9Ue-PGjtfCbD_Bn6JK_ySAMG9Hezxv6EPA/viewform?usp=sf_link`

Table 6.3: First five questions and their respective results.

Questions	Yes	No
1. Was your driving experience improved?	100%	0
2. Were the different widgets organized and well presented?	100%	0
3. Did you find the alarms clear and useful?	100%	0
4. Did you find the pop-ups clear and useful?	100%	0
5. Did you consider the dashboard a distractive agent during your test?	0	100%

Table 6.4: Last question and its respective results.

Question	1	2	3	4	5	6
6. How do you rate the dashboard's overall performance?	0	0	0	20%	40%	40%

Chapter 7

Conclusions and future work

The following chapter presents a summary of all the work developed in this dissertation regarding the solution to extend the computational system autonomy and the dashboard infrastructure. A few suggestions and proposals for future work were presented based on the conclusions from the field test.

7.1 Conclusions

The work presented in this document had three main focuses. The first was the study of a new setup for the power circuit of the ATLASCAR2. The second was related to the development of a software with the global state of the car. The last aimed to develop a fully functional dashboard that kept the vehicle's driver updated on its global condition by displaying all related data on a dynamic screen.

In what concerns the first objective, extending the ATLASCAR2 computational system autonomy, it can be concluded that the new configuration of the power board enables the computer placed on the vehicle's trunk to be operational in a broader time span than before. Originally, this equipment was only operational for 15 minutes. The lead battery provides 25 minutes of autonomy when the ignition is off. When the engine is on, the total autonomy can be extended to 3 hours, depending on the car's driving conditions. In addition, compared with the old one, the board is now much more organized and has plenty of space for new expansions. It is also equipped with extra plugs that can help future researchers on the car to have easy access to the vehicle's power source.

The second objective aimed to create a software infrastructure that enabled raw CAN packets to be received and processed. The conclusion is that this objective was accomplished since, in total, twenty parameters of the vehicle ECU's are now available and ready to be read.

Regarding the creation of the dashboard, the developed work created a network that can access the information received from the ATLASCAR2 CAN bus and then display it on a dynamic display. The results of the field test were also conclusive. In general, the five participants that took part in the study stated that the dashboard improves the driving experience of the vehicle. They found the widgets, alarms and pop-ups were clear and well presented. The display was organized, allowed an easy comprehension of

the information and was not distracting while the car was on the road. The final average score of the solution's performance was 5 on a scale from 1 to 6. Therefore, it can be concluded that the developed dashboard was a success.

7.2 Future works

The present dissertation aims to be the cornerstone for future projects since it is the first work on the ATLASCAR2 dashboard. The next expansion should equip the car with a small device, such as NUC or a Raspberry Pi, responsible for storing the dashboard software. A touchscreen has to be connected to it for the dashboard to be installed entirely on the vehicle. Since the ATLASCAR2 central panel has no space to add these devices, a solution can also be developed to support them.

Regarding the display, there is room for some improvements as well. To incorporate past ATLASCAR2 projects, the dashboard infrastructure must first be connected to the vehicle's ROS network. The car can also be equipped with a rear sensor and camera, which are enabled when the reverse layout is activated. Future works can also create virtual representations to add to the close proximity layout. The dashboard can be connected to web servers to acess live data regarding traffic and road conditions. This feature will enable the velocity warning and make it possible to add an intelligent GPS feature to the display. Another feature that could be added is the possibility of enabling some functionalities using voice commands.

Lastly, the current work only provides one-way communication since the driver can only check information from the vehicle's ECU's. Therefore, future studies could evaluate the possibility of the user controlling some car systems by accessing the dashboard screen. For instance, the driver could adjust the temperature of the air conditioner or the radio's volume by accessing the instruments on the display.

Appendix A

Hardware and software instructions

A.1 Power the switch board and inverter

First, go to the power distribution board and mount the fuse in the fuse holder. Then turn on the 120 A circuit breaker. Next, the 16 A circuit breaker must be switched to feed the sensors (Figure A.1). To power the UPS, hit the button on the front side of the inverter and press the power button (Figure A.2). The UPS and sensors are now ready to be used.

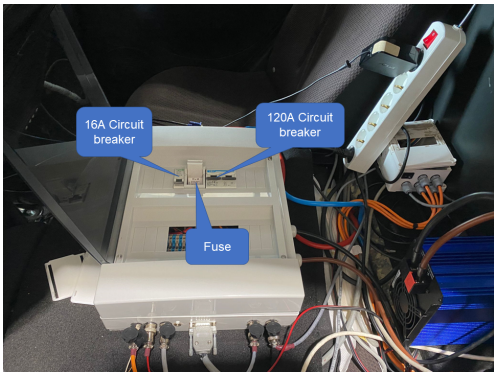


Figure A.1: Power distribution board on ATLASCAR2.

Figure A.2: Inverter and UPS on ATLASCAR2.

A.2 Monitor the vehicle status

1. Connect the CANalyze with the OBD-II port of the vehicle;
2. Set up the device acording to the instructions in section 3.2.2 ;
3. Upload the `can_publisher` script avaible on `https://github.com/Miguel-Pinto99/Thesis2022`;
4. Run the ROS node responsable for publishing the vehicle status:

```
$ rosrun dashboard can_publisher
```

A.3 Dashboard

1. Plug the CANalyze with the OBD-II port of the vehicle;
2. Set up the device acording to the instructions in section 3.2.2 ;
3. Upload the dashboard code avaible on `https://github.com/Miguel-Pinto99/Thesis2022`;
4. Run the package using the launch file:

   ```
   $ roslaunch dashboard dashboard.launch
   ```

References

[1] Atlas project. Accessed: 2022-06-01. [Online]. Available: http://atlas.web.ua.pt/

[2] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. Mccoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile." [Online]. Available: http://www.autosec.org/

[3] Can in automation: History of the can technology. Accessed: 2022-06-30. [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/

[4] N. H. T. S. Administration and U. D. of Transportation, "Traffic safety facts crash - stats critical reasons for crashes investigated in the national motor vehicle crash causation survey," 2015. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

[5] Surjeet and P. Bhardwaj, "An overview of adas in internet of vehicles," pp. 77–92, 2021. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-46335-9_6

[6] Uber blog- steel city's new wheels. Accessed: 2022-06-30. [Online]. Available: https://www.uber.com/blog/pennsylvania/new-wheels/

[7] K. Hojjati-Emami, B. S. Dhillon, and K. Jenab, "Reliability prediction for the vehicles equipped with advanced driver assistance systems (adas) and passive safety systems (pss)," *International Journal of Industrial Engineering Computations*, vol. 3, pp. 731–742, 2012. [Online]. Available: www.GrowingScience.com/ijiec

[8] D. Figueiredo, "Remote control for operation and driving of atlascar2." Mestrado Integrado em Engenharia Mecânica. Universidade de Aveiro, 2020. [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/HardwareInterfaces/2020_DiogoFigueiredo/Dissertation_FinalVersion/Thesis_DiogoFigueiredo.pdf

[9] A. N. S. Institute, "American national standard criteria for safety symbols national electrical manufacturers association," 2011.

[10] R. Swette, K. R. May, T. M. Gable, and B. N. Walker, "Comparing three novel multimodal touch interfaces for infotainment menus," 2013, pp. 100–107. [Online]. Available: http://www.keenanmay.com/wp-content/uploads/2019/10/autoui2013-rickswette-multimodaltouch.pdf

[11] T. A. Ranney, J. L. Harbluk, and Y. I. Noy, "The effects of voice technology on test track driving performance: Implications for driver

distraction," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 46, pp. 1814–1818, 9 2002. [Online]. Available: https://www.researchgate.net/publication/310826831_The_Effects_of_Voice_Technology_on_Test_Track_Driving_Performance_Implications_for_Driver_Distraction

[12] T. Poitschke, F. Laquai, S. Stamboliev, and G. Rigoll, "Gaze-based interaction on multiple displays in an automotive environment," 2011, pp. 543–548. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6083740

[13] A. M. Moutinho and P. Pereira, "Soluções industriais para visualização de dados em tempo-real." Mestrado Integrado em Engenharia Mecânica. Faculdade de Engenharia da Universidade do Porto, 2018. [Online]. Available: https://repositorio-aberto.up.pt/bitstream/10216/111325/2/259282.pdf

[14] Tesla model 3 owner's manual. Accessed: 2022-06-30. [Online]. Available: https://www.tesla.com/ownersmanual/index-model-3.html

[15] The mercedes-benz eqs450 is a legit luxury car - cnet. Accessed: 2022-06-30. [Online]. Available: https://www.cnet.com/roadshow/pictures/2022-mercedes-benz-eqs450-plus/6/

[16] L. Cristovão, "Interface obd para o atlascar2 e monitorização do seu estado." Mestrado em Engenharia de Automação Industrial. Universidade de Aveiro, 2018. [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/HardwareInterfaces/2018_LuisCristovao/Relatorio_Apresentacao/PEA_80886.pdf

[17] J. Pereira, "Quadro elétrico atlascar2." Mestrado em Engenharia de Automação Industrial. Universidade de Aveiro, 2017. [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/SystemDevelopment/2017_JosePereira/PEA_Relatorio_QE_71985.pdf

[18] J. Diogo and M. Correia, "Visual and depth perception unit for atlascar2." Mestrado Integrado em Engenharia Mecânica. Universidade de Aveiro, 2017. [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/Perception/2017_DiogoCorreia/TeseDiogoCorreia.pdf

[19] A. Reuschenbach, M. Wang, T. Ganjineh, and D. Göhring, "Idriver - human machine interface for autonomous cars." IEEE Computer Society, 2011, pp. 435–440. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5945275

[20] S. S. Awasthi, S. Arrigoni, P. Awasthi, and F. Braghin, "An interactive human-machine control interface for an autonomous shuttle." Institute of Electrical and Electronics Engineers Inc., 2021. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9662728

[21] M. Motors, "Mitsubishi i-miev 2015 owner's manual," 2015. [Online]. Available: https://mitsubishi-motors.co.uk/wp-content/uploads/2021/06/12MY-i-MiEV-Owners-Manual.pdf

[22] "i-miev warrant and maintenace manual," 2016. [Online]. Available: https://www.mitsubishicars.com/content/dam/mitsubishi-motors-us/images/siteimages/pdf/what-drives-us/warranty/2016MY%20Warranty%20Manual%20-%20IMIEV%20Complete.pdf

[23] Apc smart-ups, line interactive, 1500va, tower, 230v, lcd-apc macedonia. Accessed: 2022-06-30. [Online]. Available: https://www.apc.com/shop/mk/en/products/APC-Smart-UPS-Line-Interactive-1500VA-Tower-230V-8x-IEC-C13-outlets-LCD/P-SMT1500I

[24] Mitsubishi mz690402 battery. Accessed: 2022-06-30. [Online]. Available: https://www.allcarpartsfast.co.uk/genuine-parts/mitsubishi/mitsubishi-mz690402-battery/

[25] Socketcan - controller area network — the linux kernel documentation. Accessed: 2022-06-30. [Online]. Available: https://docs.kernel.org/networking/can.html#overview-what-is-socketcan

[26] Can-utils - elinux.org. Accessed:2022-06-30. [Online]. Available: https://elinux.org/Can-utils

[27] Ros/introduction - ros wiki. Accessed: 2022-06-01. [Online]. Available: http://wiki.ros.org/ROS/Introduction

[28] Kivy: Cross-platform python framework for nui development. Accessed: 2022-06-30. [Online]. Available: https://kivy.org/#home

[29] Kivymd · github. Accessed: 2022-06-30. [Online]. Available: https://github.com/kivymd

[30] Kivy garden · github. Accessed: 2022-06-30. [Online]. Available: https://github.com/kivy-garden/

[31] Nexus p-2308h4/hr4 - global it solutions. Accessed: 2022-07-04. [Online]. Available: http://nexus-solutions.pt/sistemas/servidores-3/p-2308h4-hr4/#!

[32] Inversor de corriente, 1000w - rs components. Accessed: 2022-07-04. [Online]. Available: https://pt.rs-online.com/web/p/inversores-de-potencia/1793330

[33] Customized cylindrical fuse manufacturers, suppliers, factory - jasco. Accessed: 2022-07-04. [Online]. Available: https://www.jasco.cn/fuse/cylindrical-fuse.html

[34] Fuse carrier - sp 58 - for hrc cylindrical fuses type 22 x 58 - 3 p - 0 216 04 - legrand. Accessed: 2022-07-04. [Online]. Available: https://www.legrand.com/ecatalogue/021604-fuse-carrier-sp-58-for.html

[35] Rs pro cable gland, m12 max. cable dia. 6.5mm, nylon, grey, 3mm min. cable dia., ip68, with locknut — rs components. Accessed: 2022-07-04. [Online]. Available: https://uk.rs-online.com/web/p/cable-glands/8229739

[36] Probots gx16 male to female aviation plug 3 pin connector. Accessed: 2022-07-04. [Online]. Available: https://probots.co.in/gx16-male-to-female-aviation-plug-3-pin-connector.html

[37] Solfight catalog. Accessed: 2022-07-04. [Online]. Available: https://www.globlec.pt/images/uploaded/Catalogos/soflight-catalogo-2022.pdf

[38] Experienced supplier of non-polarity dc circuit breaker 3p 12-750v 80-125a. Accessed: 2022-07-04. [Online]. Available: https://www.yroenergy.com/shop/non-polarity-dc-circuit-breaker-3p-12-750v-80-125a

[39] Elettrocanali catalog. Accessed: 2022-07-04. [Online]. Available: https://www.elettrocanali.com/prodotti/contenitori-centralini

[40] Mitsubishi i-miev (2015-2018) price and specifications - ev database. Accessed: 2022-07-14. [Online]. Available: https://ev-database.org/car/1096/Mitsubishi-i-MiEV

[41] H. M. Boland, M. I. Burgett, A. J. Etienne, and R. M. S. III, "An overview of can-bus development, utilization, and future potential in serial network messaging for off-road mobile equipment," in *An Overview of CAN-BUS Development, Utilization, and Future Potential in Serial Network Messaging for Off-Road Mobile Equipment*, 2015. [Online]. Available: www.intechopen.com

[42] C. Smith, *The Car Hacker's Handbook, A Guide for the Penetration Tester.* San Francisco: No Starch Press,Inc, 2016.

[43] Decyphering imiev and ion car-can message data - mitsubishi i-miev forum. Accessed: 2022-07-06. [Online]. Available: http://myimiev.com/forum/viewtopic.php?t=763

[44] Github - kivy-garden/garden.mapview: Migrated to https://github.com/kivy-garden/mapview. Accessed: 2022-07-11. [Online]. Available: https://github.com/kivy-garden/garden.mapview