Universidade de Aveiro Departamento de Engenharia Mecnica
2020

Diogo Augusto Rodrigues de Figueiredo

Remote Control for Operation and Driving of ATLASCAR2

Controlo Remoto para a Operação e Condução do ATLASCAR2


Universidade de Aveiro Departamento de Engenharia Mecnica
2020

Diogo Augusto Rodrigues de Figueiredo

Remote Control for Operation and Driving of ATLASCAR2

Controlo Remoto para a Operação e Condução do 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.

o júri / the jury

presidente / president

Prof. Doutor Miguel Armando Riem de Oliveira
Professor Auxiliar da Universidade de Aveiro

vogais / committee

Prof. Doutor Carlos Fernando Couceiro de Sousa Neves
Professor Coordenador do Instituto Politécnico de Leiria - Escola Superior de Tecnologia e Gestão

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado C/ Agregação da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

A realização desta dissertação de mestrado contou com importantes apoios e incentivos. A todas as pessoas que me acompanharam neste projeto quero deixar aqui o meu sincero agradecimento.

Ao Professor Vítor Santos, pela sua orientação, apoio e disponibilidade durante todo o semestre.

Aos meus amigos, em especial ao David, que, dentro e fora do contexto curricular, me ajudaram a creser e aprender ao longo destes 5 anos.

Aos meus companheiros do projeto ATLAS, Rúben e Rui, pela amizade e boa disposição transmitida ao longo do semestre.

Aos meus pais, pelo apoio e confiança que depositaram em mim durante todo o percurso académico.

À Raquel, por toda a compreensão, motivação e companhia demonstrada em todos os momentos.

keywords

Controller Area Network; SocketCAN; ROS Architecture; Remote Control; Electronic Control Units; Steering System; Propulsion System

abstract

Obtaining a Self Driving Vehicle requires a well-planned collaboration between different areas and technologies, namely, it is crucial to effectively combine the sensory systems with the ability of remotely control the vehicle. This dissertation, integrated in the University of Aveiro Autonomous Driving project ATLASCAR2, links the vehicle's actuators to the perception, navigation and decision making algorithms by creating a control and monitoring infrastructure and presents solutions to remotely control vehicle's direction and speed.

The control and monitoring infrastructure developed in this work uses the Controller Area Network of the vehicle for communication purposes and to access the current status of the vehicle by processing the messages exchanged between control units. In order to obtain solutions to remotely act on the direction and speed of the vehicle, the operation logic of the steering and propulsion systems of the Mitsubishi i-MiEV was studied and solutions that replicate the behaviour of the sensors used on these system are implemented.

The results show the appropriateness of the developed solutions to control and monitor the vehicle and prove the feasibility of the method used to remotely control the steering wheel and accelerator pedal of the vehicle. However, for the braking system, an external mechanical actuator acting on the brake pedal is required to effectively control this system.

Based on the results, the presented solutions show the great potential of utilizing the ATLASCAR2 to be used as a test platform of Autonomous Driving applications in a near future.

palavras-chave

Controller Area Network; SocketCAN; Arquitetura ROS; Controlo Remoto; Electronic Control Units; Sistema de Direção; Sistema de Propulsão

resumo

A obtenção de um Veículo Autónomo requer colaboração entre várias áreas e tecnologias, nomeadamente, é fundamental conseguir combinar os sistemas sensoriais com a capacidade de controlar o veículo remotamente de forma eficaz. Esta dissertação, integrada no projeto de Condução Autónoma da Universidade de Aveiro ATLASCAR2, conecta os atuadores do carro aos algoritmos de perceção, navegação e tomada de decisão criando uma infraestrutura de controlo e tomada de decisão e apresenta soluções para controlar remotamente a direção e velocidade do carro.

A infraestrutura de controlo e monitorização desenvolvida neste projeto utiliza a Controller Area Network do veículo para efetuar a comunicação e aceder ao atual estado do veículo através do processamento das mensagens trocadas entre as unidades de controlo do mesmo. Para obter soluções para a atuação remota da direção e velocidade do veículo, foi estudado o modo de funcionamento dos sistemas de direção e propulsão do Mistubishi i-MiEV e a solução implementada replica o comportamento dos sensores utilizados nestes sistemas.

Os resultados obtidos demonstram a adequação das soluções desenvolvidas para o controlo e monitorização do veículo e comprovam a viabilidade dos métodos utilizados para controlar remotamente o volante e acelerador do carro. No entanto, para o sistema de travagem seria necessário utilizar o atuador mecânico externo a atuar no pedal do travão para controlar este sistema remotamente.

Os resultados apresentados evidenciam que o veículo ATLASCAR2 possa ser utilizado como plataforma de testes de Condução Autónoma num futuro próximo.

Contents

1 Introduction **1**
- 1.1 Project Context and Motivation . . . 1
  - 1.1.1 The ATLAS Project . . . 1
  - 1.1.2 Self Driving Vehicles and Technology . . . 3
- 1.2 Problem Description and Objectives . . . 4
- 1.3 Document Structure . . . 5

2 State of the Art **7**
- 2.1 Background - Automotive Electric Systems . . . 7
- 2.2 Related Work . . . 8
  - 2.2.1 Mitsubishi i-MiEV . . . 8
  - 2.2.2 Other Vehicles . . . 10
- 2.3 Related Work in Other Contexts . . . 11
- 2.4 Current Implementations on Automobile Industry . . . 12
- 2.5 Summary . . . 13

3 Experimental Infrastructure **15**
- 3.1 Hardware . . . 15
  - 3.1.1 CANalyze . . . 15
  - 3.1.2 ATLASCAR2 . . . 16
- 3.2 Software . . . 18
  - 3.2.1 ROS - Robot Operating System . . . 18
  - 3.2.2 SocketCAN . . . 19
- 3.3 Summary . . . 22

4 Car Status Monitoring **25**
- 4.1 Understanding CAN Protocol . . . 25
- 4.2 Identification of the Mitsubishi i-MiEV CAN Frames . . . 27
  - 4.2.1 Mitsubishi i-MiEV CAN Bus . . . 27
  - 4.2.2 Locating and Reading the CAN Bus . . . 28
  - 4.2.3 Perform Message Identification . . . 29
- 4.3 Summary . . . 31

5 Remote Control Solutions **33**
5.1 Steering Wheel Remote Control . . . 33
5.1.1 Mitsubishi i-MiEV Power Steering Logic . . . 33
5.1.2 Proposed Solution . . . 35
5.2 Vehicle Speed Remote Control . . . 40
5.2.1 Mitsubishi i-MiEV Propulsion Logic . . . 40
5.2.2 Mitsubishi i-MiEV Braking Logic . . . 43
5.2.3 Proposed Solution . . . 43
5.3 Summary . . . 46

6 Actuators Control Management **49**
6.1 Remote Control Using the CAN Bus . . . 49
6.1.1 External ECU . . . 49
6.1.2 Creating CAN Messages . . . 51
6.2 Driving Modes Compatibility . . . 51

7 Tests and Results **55**
7.1 Direction Remote Control . . . 55
7.2 Speed Remote Control . . . 64

8 Conclusions and Future Work **69**
8.1 Conclusions . . . 69
8.2 Future Work . . . 70

A Arduino IDE Code **71**

B Steering wheel response to different types of surfaces **77**

C Instruction Manual **79**
C.1 Monitor the vehicle status . . . 79
C.2 Control the Steering Wheel and Accelerator Pedal . . . 79

List of Tables

4.1 OBD-II port pinout. 28
4.2 Relation between the shift position and the value of the first byte of the message 0x418. 30
4.3 Relation of message `0x424` with some of the instrument panel variables. . 31

5.1 Signals of the B-114-2 connector terminals [24]. 35
5.2 Arduino UNO specifications [32]. 39
5.3 EV-ECU terminals regarding accelerator pedal position [24]. 45

6.1 Description of the CAN messages created to send commands to the controllers. 51

7.1 Analysis of the signals sent to the EPS-ECU and the respective steering wheel state with the vehicle stationary and in motion. 61
7.2 Response of the steering system to the same electrical signals at different positions. 63

Intentionally blank page.

List of Figures

1.1 ATLAS robots at the 2010 Robotics Festival - Atlas2010 and AtlasMV3 [2]. 2
1.2 ATLASCAR1 test platform [2]. 2
1.3 ATLASCAR2 test platform [3]. 3
1.4 Levels of Driving Automation [5]. 4
1.5 Dissertation function in the ATLASCAR2 project. 4

2.1 World's first self driving taxi [13]. 8
2.2 Mitsubishi i-MiEV used in the AUTO C-ITS project in Madrid [14]. . . . 9
2.3 Remote control solution for the braking system in the AUTO C-ITS project in Madrid. 9
2.4 ISEAUTO autonomous minibus [16]. 10
2.5 Attack surfaces existing in a modern vehicle [22]. 11

3.1 CANalyze - open source hardware to receive and transmit CAN messages [23]. 15
3.2 Control unit responsible for the steering operation in the ATLASCAR2. . 16
3.3 Control unit responsible for the propulsion operation in the ATLASCAR2. 16
3.4 General operation logic of Mitsubishi i-MiEV. 16
3.5 Mitsubishi i-MiEV steering components. [24] 17
3.6 General operation of the Mitsubishi i-MiEV propulsion system. 18
3.7 Basic operation architecture of ROS. [25] 18
3.8 SocketCAN communication layers [11]. 19

4.1 Voltage levels of the CAN bus to data transmission [24] 26
4.2 Standard CAN frame structure, with the number of bits of each field [28]. 26
4.3 Mitsubishi i-MiEV CAN Bus [24]. 27
4.4 OBD-II port pinout [29]. 28

5.1 Electric Power Steering-ECU System Construction Diagram. 34
5.2 Operation of the Mitsubishi i-MiEV power system. 34
5.3 EPS-ECU connectors [24]. 35
5.4 Voltage values generated by the torque sensor while turning the steering wheel completely to the left with the vehicle stationary. 36
5.5 Voltage values generated by the torque sensor while turning the steering wheel completely to the right with the vehicle stationary. 37
5.6 Voltage values generated by the torque sensor while turning the wheel completely to the left with the vehicle at the speed of 6 km/h. 38
5.7 Voltage values generated by the torque sensor while turning the wheel to completely the right with the vehicle at the speed of 6 km/h. 38

5.8 Arduino UNO [32]. 39
5.9 Low-pass filter [33]. 40
5.10 Electric motor control unit [24]. 41
5.11 EV-ECU operation logic [24]. 42
5.12 Torque command logic of the EV-ECU using basic control [24]. 42
5.13 Electric motor used in the brake electric vacuum pump [24]. 43
5.14 Mitsubishi i-MiEV braking system [24]. 44
5.15 Regenerative braking system [34]. 44
5.16 Relation between the voltage of the accelerator pedal position sensor main and sub line and the opening of the accelerator pedal [24]. 46
5.17 Voltage values of the accelerator pedal position sensor main and sub lines when the accelerator pedal is pressed completely. 47

6.1 External ECU. 50
6.2 Using an OBD-II port splitter in the ATLASCAR2 to connect all the devices to the vehicle's network. 50
6.3 Circuit used in the transition between driving modes of the steering system. 52
6.4 Transition between autonomous and manual driving modes. 53
6.5 Circuit used in the transition between driving modes of the steering system. 54

7.1 Steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU. 56
7.2 Detail view of the steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU. 56
7.3 Steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU. 57
7.4 Detail view of the steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU. 57
7.5 Steering wheel position of the car in stationary state over a surface of road tar with the corresponding voltage values sent to the EPS-ECU . . . 58
7.6 Steering wheel position of the car in stationary state over a surface of sand with the corresponding voltage values sent to the EPS-ECU. 59
7.7 Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU. 59
7.8 Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU. 60
7.9 Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU. 60
7.10 Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU. 61
7.11 Differences observed in the steering wheel with the vehicle in stationary state and at different speeds. 62
7.12 Controlling the steering wheel angle with CAN communication. 65
7.13 Speed of the vehicle with the respective voltage values sent to the EV-ECU. 66
7.14 Speed of the vehicle with the respective voltage values sent to the EV-ECU. 66
7.15 Speed of the vehicle with the respective voltage values sent to the EV-ECU. 67

B.1 Steering wheel position of the car in stationary state over a surface of road tar with the corresponding voltage values sent to the EPS-ECU. . . 77
B.2 Steering wheel position of the car in stationary state over a surface of sand with the corresponding voltage values sent to the EPS-ECU. 78

Intentionally blank page.

Chapter 1

Introduction

Autonomous Driving (AD) has become one of the focus of the automobile industry over the past years, as such, many technological advances have been developed at industrial and academic levels. As the systems evolve, it becomes clear that AD is the future of the automobile industry, since, in its complete development, it will drastically improve the safety and efficiency of the vehicle's mobility.

To obtain a level of Full Automation (the vehicle performs all driving tasks under all conditions) different technologies must be combined. Road and obstacle detection, trajectory planning and decision making algorithms are responsible for defining the speed and direction of the vehicle. To move the vehicle based on this information, a full control of its main actuators is required, which can be achieved by taking advantage of the high-computerised operation methods on today's automobiles.

Modern vehicles are no longer composed only by mechanical components. Instead, they use several Electronic Control Units (ECU) communicating via an internal vehicular network to monitor and control the car status [1]. Based on the understanding of these concepts and the Mitsubishi i-MiEV operating logic, this dissertation aims to find solutions for the remote control of the i-MiEV steering wheel and speed and integrate these controllers in the AD global system in a robust and efficient way.

1.1 Project Context and Motivation

1.1.1 The ATLAS Project

This dissertation is part of the ATLAS project, created by the Group of Automation and Robotics at the Department of Mechanical Engineering of the University of Aveiro, Portugal. This project is focused on the development of a sensory architecture in order to create autonomous navigation solutions in cars and other platforms [2].

The first step on the ATLAS project was the development of mobile autonomous robots (Fig. 1.1), which took part at the Portuguese National Robotics Festival autonomous driving competitions. The participation and success in this AD competition allowed the project to proceed to the next step, the ATLASCAR1.

ATLASCAR1 (Fig. 1.2) is a Ford Escort model used as prototype for research on Advanced Driver's Assistance Systems. The vehicle was equipped with several sensors to identify the surroundings and act on the vehicle mechanical components using external

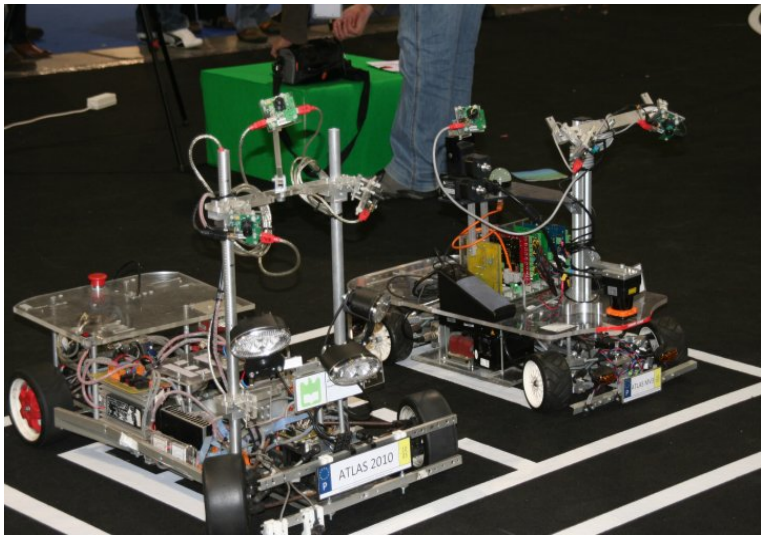


Figure 1.1: ATLAS robots at the 2010 Robotics Festival - Atlas2010 and AtlasMV3 [2].

actuators for that propose. After some interesting and successful results, the ATLAS project evolved to a new full-sized platform, the ATLASCAR2.

Figure 1.2: ATLASCAR1 test platform [2].

The ATLASCAR2 (Fig. 1.3) is an electric Mitsubishi i-MiEV vehicle from 2015 in which the research in this dissertation will occur on. This car is equipped with LIDAR sensors, GPS and cameras used in navigation and perception algorithms. The fact that it is a more modern car than the ATLASCAR1, brings new and easier possibilities to test and control the algorithms, as this dissertation will describe.

Figure 1.3: ATLASCAR2 test platform [3].

1.1.2 Self Driving Vehicles and Technology

As described above, the main objective of the ATLAS project is to create a Self Driving Vehicle (SDV) – ATLASCAR2. The interest in this technology is one of the major motivations for the development of this dissertation.

The International Society of Automotive Engineers (SAE International) has divided the level of automation of a vehicle in six different levels (Fig. 1.4), based on four principal parameters: execution of steering and acceleration/deceleration, monitoring of driving environment, fallback performance of dynamic driving tasks and system capability [4].

In the first three levels (0-2), the human monitors the driving environment. These are the most common degrees of automation on today's automobile industry. In the levels 3-5 the automated driving system monitors the driving environment. These are the levels in which this dissertation will work on, since the steering and speed control works with an automatic digital system. At the third level (Conditional Automation) the vehicle performs all the driving tasks with the expectation that the driver will respond appropriately to a request to intervene. In the level 4, called High Automation, the driving tasks are performed by the vehicle even if the human does not respond to an override request. In the last degree – Full Automation – the vehicle performs all driving tasks under all conditions. At this level, no human attention is required.

The recent interest in the development of AD technologies is mainly driven by two reasons: safety and time. In 2019 there were more than 135 thousand road accidents in Portugal [6]. Most of these accidents were caused by human error, such as drink-driving, over-speed situations or infringement of traffic laws. These situations would not have happened in an autonomous driving environment where the human does not intervene directly in driving task. Also, in a level of Full Automation, humans can be completely free from driving tasks, allowing them to dedicate the travelling time to other purposes. Statistics are that, in the U.S., the average time spend inside the vehicle is almost an hour per day [7]; that time could be used in other activities if travelling in a last-level

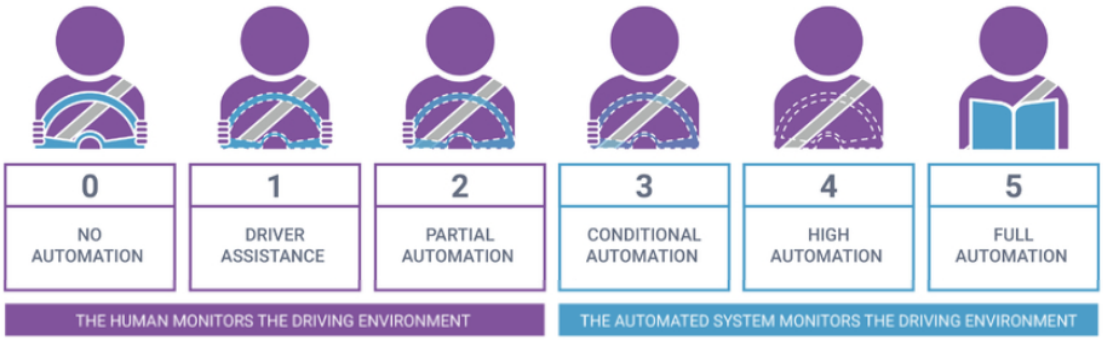


Figure 1.4: Levels of Driving Automation [5].

SDV.

What seems to be against the insertion of SDV on the roads are mainly ethical issues, since no technology is 100% error free and the fact that a system mistake can cost a human life can be a real issue. In Portugal, AD tests are allowed as long as a human driver is ready to intervene in the vehicle at any time.

1.2 Problem Description and Objectives

The ATLASCAR2 project is in a phase where the perception and navigation algorithms are ready to be incorporated within the Mitsubishi i-MiEV hardware. That said, the next step in this project is to be able to remotely control the vehicle's main actuators: steering wheel, brake and accelerator pedal. Thus, the main focus of this dissertation is to develop and manually test solutions to act on the ATLASCAR2 speed and direction, in order to allow their further integration with the perception and navigation algorithms, in case of success. A schematic representation of how this dissertation suits in the ATLASCAR2 project is presented in Figure 1.5.

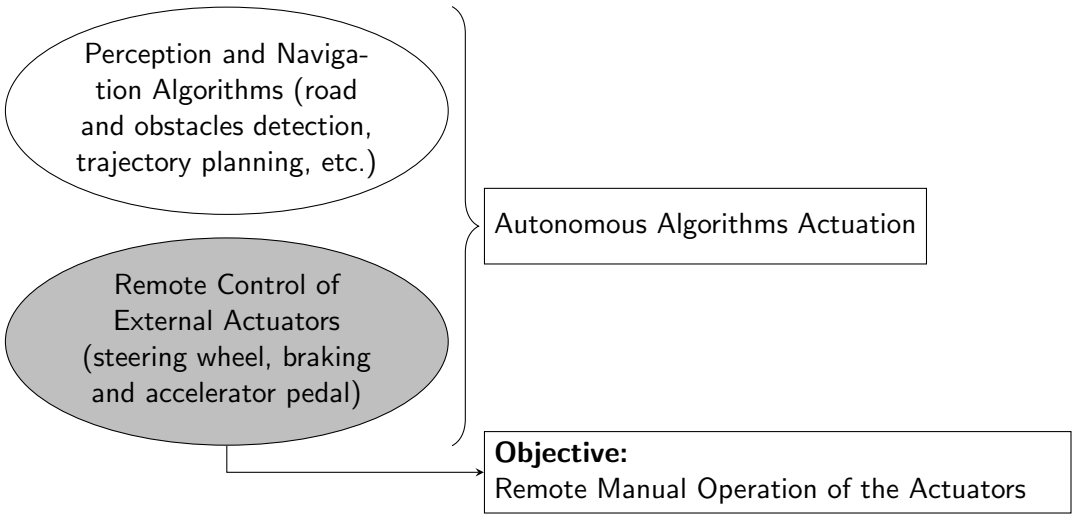


Figure 1.5: Dissertation function in the ATLASCAR2 project.

To properly develop this type of remote control solutions it is necessary to have a real-time update of the car status, which is also one of the tasks of this work.

In addiction, the integration and testing of the developed solutions in the ATLASCAR2 global system are part of this work. The following topics summarise the main objectives of the dissertation:

- installation of a unit to access the vehicular internal network;
- real-time monitoring and publishing of the status of the ATLASCAR2;
- development of a control unit for the vehicle's actuators dedicated to the direction and propulsion systems;
- test and integrate the designed systems.

1.3 Document Structure

This document is organised in eight chapters. In the first two chapters, the dissertation is presented by introducing the problem and objectives and describing related works from different authors. Chapters 3 explains the infrastructure of the work, divided in two categories: hardware and software. In Chapter 4, the proposed solution to publish the vehicle's state to the global system is described. Chapters 5 and 6 refer the proposed solutions to remotely control the vehicle's actuators and explain the control infrastructure created for that purpose. The tests made to evaluate the proposed solutions and the results obtained are presented in Chapter 7. Finally, in Chapter 8, the conclusions of this work and future directions are explained.

Intentionally blank page.

Chapter 2

State of the Art

As mentioned before, this dissertation will take advantage of the high degree of computerization of modern vehicles to remotely control the actuators of the ATLASCAR2. This chapter presents some works that use similar techniques to control external actuators of a Mitsubishi i-MiEV and other vehicles.

2.1 Background - Automotive Electric Systems

In order to understand some of the concepts mentioned in this dissertation and the presented solutions, it is important to provide a basic background concerning the automotive embedded systems architecture.

In the late 1970s, due to requirements of California Clean Air Act, the vehicle production in U.S. implemented the first digital control embedded system [8]. This device, called *Engine Control Unit*, was able to improve efficiency and reduce pollutants by adjusting the fuel/oxygen mixture before combustion. The success of this control technique led to the integration of similar systems controlling and monitoring other functions of the vehicle. Currently, these digital systems cover, literally, all the vehicles features, including the throttle, transmission, brakes,passenger climate and lighting controls, external lights,entertainment, and so on [1].

Indeed, premium cars can have up to 100 ECUs [9] that must communicate with each other to monitor and control the vehicle status. To fill the need for an efficient and reliable communication between ECUs, in 1985, Bosh developed the Controller Area Network (CAN) standard protocol. The internal network of a vehicle can be accessed via the On-Board Diagnostics II (OBD-II) port, which is mandatory in all cars in U.S. after 1996 and Europe after 2004 [10] and can be normally found under the steering column on modern cars. The CAN bus protocol and the characteristics of the OBD-II port are explained in detail in Chapter 4.

Several research links and documentation that explain the evolution of computerization in modern vehicles are available online. However, very few of these documents look at this topic in a logic of penetrating the car network and ECUs. As an introduction on how to penetrate cars and its potentials, the online tutorial "How to hack a car — a quick crash-course", by Kenny Kuchera [10], is a good first article to understand the principals of cars networks and perform the first tests on reading and writing messages on the car network. A deeper and more complex explanation on this theme can be found on the book "The Car hacker's handbook: a guide for the penetration tester", by Craig

Smith [11].

2.2 Related Work

2.2.1 Mitsubishi i-MiEV

As mentioned on the ATLASCAR2 project presentation, the Mitsubishi i-MiEV is a full electric vehicle that brings a lot of possibilities for developing and testing AD applications. This fact, associated with the low market price of the car, makes it a good option when trying to turn a regular car into a SDV. Thus, there are several AD projects which, directly or indirectly, use this model as a platform.

In 2013, an autonomous vehicle startup - nuTonomy [12] - was founded and proposed thousands of self-driving taxis in Singapore by the year of 2019. One of the models used was the Mitsubishi i-MiEV (Fig 2.1). They were able to remotely control all the car actuators, but, unfortunately, the solutions used are not available.

Figure 2.1: World's first self driving taxi [13].

Another work that involves self-driving applications in a Mitsubishi i-MiEV is the demonstration project AUTO C-ITS [14], that ATLASCAR2 project is also part of. This project is co-financed by the European Union and have been creating AD implementations in Portugal, Spain and France. In Spain, the project has been developed in partnership with the Polytechnic University of Madrid and the vehicle used as the project platform is the Mitsubishi i-MiEV (Fig. 2.2).

Again, there is no information concerning the implemented solutions, however, in a demonstration video [14], it is possible to see that the braking system is controlled by an external mechanical actuator on the brake pedal, as can be seen in the Figure 2.3. The solution obtained for the steering wheel remote control is not clear, but, in the same video, it is possible to see the steering wheel moving without the help of a mechanical external actuator.

Figure 2.2: Mitsubishi i-MiEV used in the AUTO C-ITS project in Madrid [14].

Figure 2.3: Remote control solution for the braking system in the AUTO C-ITS project in Madrid.

Finally, there is the ISEAUTO project [15]. This is a cooperation project between the Tallinn University of Technology (TalTech) and Silberauto Estonia that focuses on the development of an autonomous minibus (Fig. 2.4) to operate mainly on the campus of TalTech. In this project the Mitsubishi i-MiEV was used as a test platform and the minibus was built on the i-MiEV trolley, so the final solution uses the propulsion system, shift mechanism and steering architecture of the Mitsubishi i-MiEV [16].

Figure 2.4: ISEAUTO autonomous minibus [16].

2.2.2 Other Vehicles

There are countless projects that aim to develop AD applications. Some of these are being developed by major companies, such as the Waymo from Google [17] and the Autopilot from Tesla [18]. However, understandably, these big companies have no available information regarding the implemented solutions for the autonomous mobility of their vehicles. So, in this section, the work of two security researchers that developed remote control applications on two different cars and published all their tools, data, research notes and papers on [19] will be highlighted.

Charlie Miller and Chris Valsek were able to control the main actuators of a Toyota Prius and Ford Escape by reverse engineering the vehicles' internal communication and ECU firmware. This was the initial approach of this dissertation - remotely control the ATLASCAR2 actuators by sending the correct CAN frames into the CAN bus, however, as will be later explained, the Mitsubishi i-MiEV ECUs are not ready to control all its actuators based on CAN communication. Also, this method is associated with some complications that Charlie Miller and Chris Valsek describe in their work. First of all, the CAN messages used by the ECUs to perform actions in the vehicle are kept confidential by the manufacturer, due to safety reasons. Additionally, the critical actions are protected by a password, also unknown. Finally, there is a conflict problem, since the target ECU will not only receive the injected messages but also those of the original ECU [20].

In their publications, Charlie Miller and Chris Valsek discussed techniques to deal with this issues that will not be described in this section, although it is important to notice this method as a possible solution to solve the problem of autonomous mobility

in modern vehicles.

2.3 Related Work in Other Contexts

The utilization of several control units in today's vehicles has improved efficiency and safety in automotive industry, but has also introduced new potential risks. The possibility of remotely act on the actuators of a car can compromise the security of the vehicle occupants when used with malicious objectives.

In 2011, researchers from the University of Washington and the University of California San Diego [21] showed that malicious code can be injected in the vehicle's bus by an attacker directly or indirectly (wireless) to control critical systems [22]. In fact, all the ways the vehicle communicates with the exterior represents a vulnerability that an attacker can take advantage of to control the vehicle (Fig. 2.5).

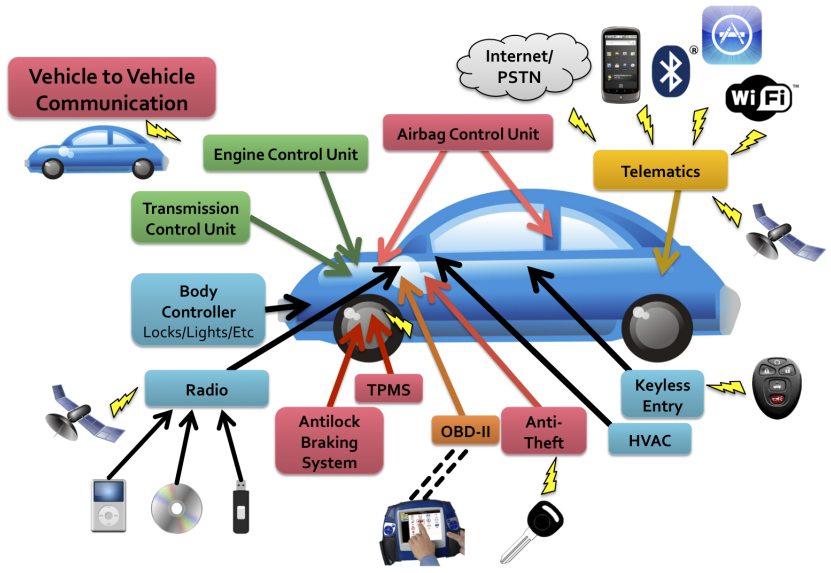


Figure 2.5: Attack surfaces existing in a modern vehicle [22].

In their research, the investigators were able to take full control of all the ECUs connected to the vehicle's CAN bus using different types of attack surfaces, divided by the authors in the following way:

- Direct physical access: plug the attack hardware directly in the OBD-II port of the car;
- Indirect physical access: this vulnerability is mainly composed by entertainment systems, such as CD player, USB port and iPod port;
- Short range wireless access: in this case, the Bluetooth channel was used. This category also includes Remote Keyless Entry, RFIDs, Tire Pressure Monitoring Systems, WiFi, and Dedicated Short-Range Communications.
- Long range wireless access: include broadcast receivers for long-range signals, such as the Global Positioning System (GPS).

Unfortunately, the tools and techniques used in the work are not available and not even the model of the vehicle used as platform is revealed, due to safety concerns. The main goal of the research was only to show the existence of the referenced threats in modern vehicles.

In the case of this dissertation, the main objective is to take control of the vehicle actuators to perform an AD solution, however, it is important to note that similar techniques can be used to jeopardize the safety of the vehicle and its occupants.

2.4 Current Implementations on Automobile Industry

Last level AD vehicles are still not a reality in today's mobility implementations. So, in this section, some vehicle functionalities that use the control units and communication between ECUs will be presented.

Indeed, most of recent innovations in automobile industry were implemented using software alone. The creation of these functionalities would not be feasible by applying the traditional mechanical and electrical solutions of the automobile industry.

The systems presented below act on the vehicles external actuators without human intervention, so, reverse engineering them can, likely, allow to remotely control the main actuators, under certain conditions.

Cruise Control

The majority of today's cars have Cruise Control. This system allows the speed to be controlled automatically by an ECU without actions being performed by the driver on the pedals.

Automatic Parking System

The Automatic Parking System is achieved by controlling, coordinately, the steering angle and speed of the vehicle, taking into consideration the available space to park. A vehicle with this feature is able to steer the steering wheel, in a precise way, without human intervention (at least at low speed driving).

Advanced Emergency Braking System

This system automatically detects a potential collision and activates the braking system to decelerate/stop the vehicle. In this case, the vehicle is able to use the brakes with no pressure being applied in the brake pedal.

The topics above include systems capable of control, without human intervention, the three main components of cars: accelerator pedal, brakes and steering wheel. These systems alone are proof that modern cars are prepared to be transformed in self-driving vehicles. In addiction, there are well-known functionalities that are only possible by using control units and the CAN bus:

- Auto Start/Stop: in order to improve emissions and fuel consumption, the Auto Start/Stop system uses various sensor inputs that run in the CAN bus to determine if is possible to shut down the vehicle;
- Parking Assist systems: when the reverse gear is engaged a signal is send via the CAN bus and is used to activate various systems, such as the parking sensor system;
- Collision Avoidance System: the vehicle speed, the speed of the vehicle in front of it and the distance between the vehicles is monitored in the CAN bus, to provide a warning to the driver in case of imminent collision;

2.5 Summary

Several works and projects focused on the remote actuation of vehicles were presented in this chapter, including the Mitsubishi i-MiEV, which is the ATLASCAR2 platform. Moreover, emphasis was also given to a different approach based on the control of vehicle actuators by penetrating the CAN bus and replicating the CAN messages that the control units use to perform the specific actions.

Intentionally blank page.

Chapter 3

Experimental Infrastructure

This chapter presents the hardware (Section 3.1) and software (Section 3.2) used in this dissertation to solve the proposed problems.

3.1 Hardware

3.1.1 CANalyze

The ability to receive and send CAN packets to the CAN bus of the vehicle in a fast and reliable way is crucial for this dissertation and the ATLASCAR2 project in general. For that, CANalyze (Fig. 3.1) is used. This device is an open source, native CAN interface for Linux (Operating System in which the project is based on). This hardware enables the monitoring and transmission of CAN frames using SocketCAN (Section 3.2.2), facilitating the communication process.

In a simple way, the CANalyze device performs on-board processing of the CAN packets to make them readable to the user, allowing the computer to become a node of the vehicle CAN bus.

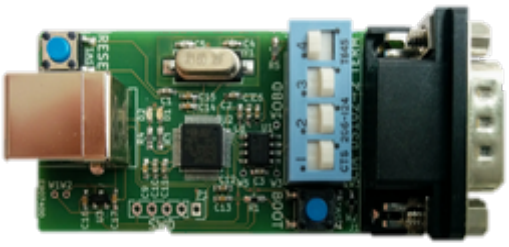


Figure 3.1: CANalyze - open source hardware to receive and transmit CAN messages [23].

To connect CANalyze to the OBD-II port of the vehicle, a OBD-II to DB9 serial cable is required. Also, after processing the CAN packets, CANalyze sends them through a USB-B port. In order to transfer the data to the computer, a USB-B to USB-A cable is required.

3.1.2 ATLASCAR2

As mentioned in the first chapter, the ATLASCAR2 is a Mitsubishi i-MiEV vehicle in which the work will be carried out. In the context of this dissertation, the components of the vehicle used are the ones related to the driving solutions, particularly the ECUs that control the vehicle's steering (Fig. 3.2) and propulsion (Fig. 3.3) operations.

Figure 3.2: Control unit responsible for the steering operation in the ATLASCAR2.

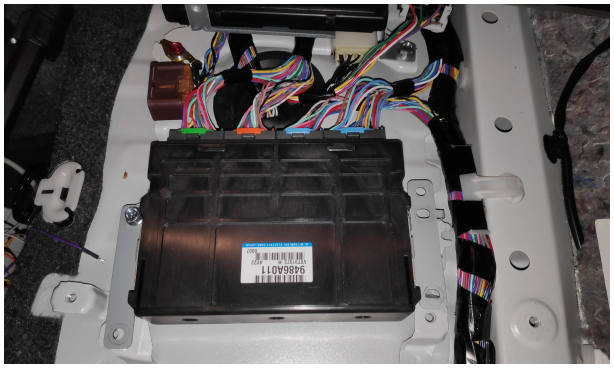


Figure 3.3: Control unit responsible for the propulsion operation in the ATLASCAR2.

The operation system of the Mitsubishi i-MiEV is based on ECUs processing electrical signals from sensors and outputting a value to an actuator, according to a previously programmed software. Figure 3.4 is a schematic representation of this operation logic.

Figure 3.4: General operation logic of Mitsubishi i-MiEV.

This dissertation will be focused on the ATLASCAR2 sensors, ECUs and actuators regarding the steering and propulsion systems.

Steering System

In the Mitsubishi i-MiEV, the Electric Power System-ECU (EPS-ECU) is responsible to manage the steering operation. In a simple explanation, this controller uses the speed of the vehicle and signals from the torque sensor, according to the steering force, and controls the electric motor that acts on the steering column. The Mitsubishi i-MiEV steering system consists of the components shown in Figure 3.5.

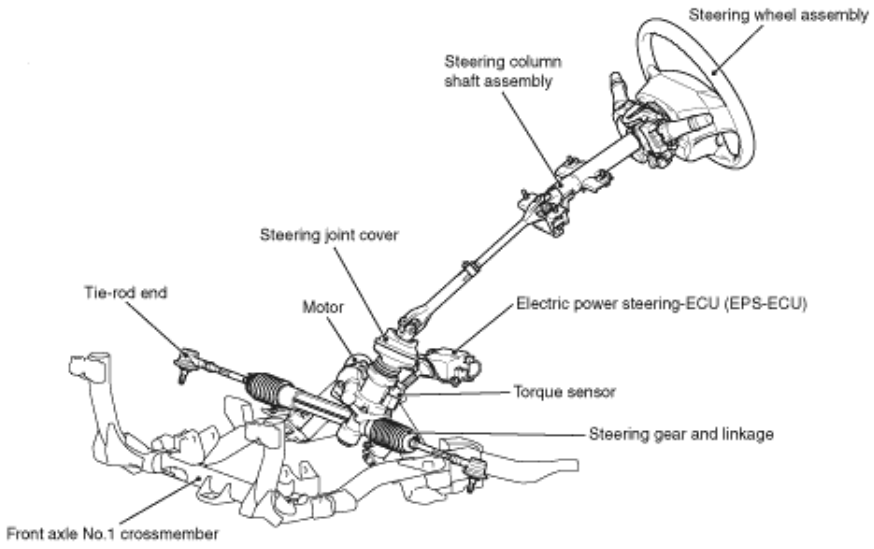


Figure 3.5: Mitsubishi i-MiEV steering components. [24]

Propulsion System

When compared to the steering system, the propulsion system of the Mitsubishi i-MiEV is more complex, due to safety, energy consumption and comfort reasons.

As represented in Figure 3.6, the software of the Electric Vehicle-ECU (EV-ECU) is programmed to output the torque command, in the form of a CAN message, to the Electric Motor Control Unit (EMCU), based on driver, vehicle and battery information. Among the EV-ECU inputs, the Accelerator Pedal Position (APS) sensor and the Brake Pedal Stroke (BPS) sensor represent, respectively, the accelerator pedal opening and the amount of brake pedal stroke.

The torque command signal calculated by the EV-ECU is then processed by the EMCU, which determines the final torque and supplies the electric motor with the corresponding current. The EMCU also controls the inverter circuit, which allows the electric motor to perform regenerative braking and decelerate the vehicle.

In Chapter 5, the steering and propulsion systems will be explained in detail in order to understand how these systems were used to remotely control the vehicle direction and speed.

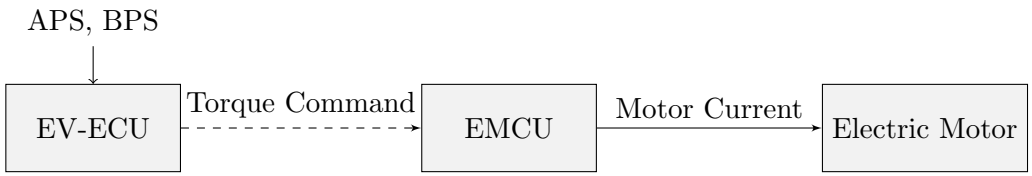


Figure 3.6: General operation of the Mitsubishi i-MiEV propulsion system.

3.2 Software

3.2.1 ROS - Robot Operating System

An AD project must have a well-planned communication support among the independent algorithms. In the case of the ATLASCAR2, the Robotic Operating System (ROS) architecture is used for that purpose. This framework operates on top of Linux and has an organised and efficient communication method that is able to handle large volumes of data, which makes it ideal to deal with a project of this complexity.

A ROS project is based on four main components: master, nodes, topics and services. The nodes are executables that communicate with each other using topics or services. In the context of this work, the topics are used in order to perform the communication between nodes in a publisher-subscriber logic. These topics contains messages that can either be standard ROS messages, or personalized messages. The control of the communication process is done by the master, which makes possible for nodes to find each other and exchange data [25] (Fig 3.7).

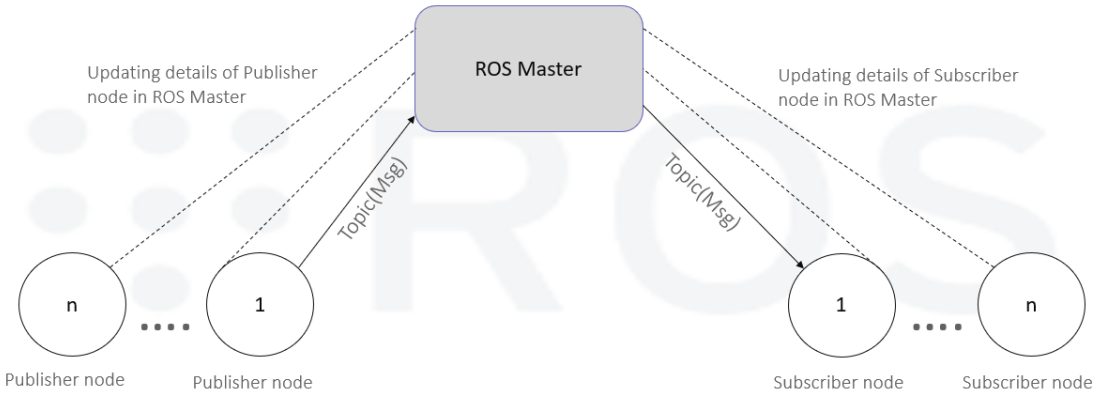


Figure 3.7: Basic operation architecture of ROS. [25]

This dissertation is inserted in the ROS architecture of the ATLASCAR2 project in a two-way communication with the other independent algorithms: it is part of this work to publish the vehicle status to the global system (these parameters are published in a topic named `NominalData`), also, the messages received from the car CAN bus are published in their raw form in the `RawFrames` topic, which contains ROS standard messages created to support CAN communication (`can_msgs`); on the other hand, the decision making algorithms define and publish the desired state of the car on the global

system. In order to obtain this information and pass it to the controllers, the respective nodes should subscribe the topic that contains this information.

3.2.2 SocketCAN

Exchanging messages with the vehicle CAN bus using CANalyze requires a software application capable of it. Linux supports CAN communication since 2008, when the subsystem SocketCAN was created. SocketCAN is a set of open source CAN drivers and a networking stack (Fig. 3.8) developed by Volkswagen AG with the main goal of develop a similar interactions with the CAN bus as with other networks (e.g. TCP/IP) in order to facilitate the development of a program to communicate with CAN devices.

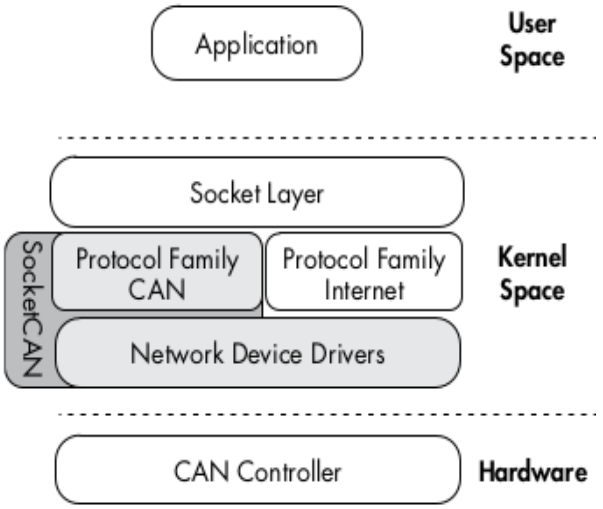


Figure 3.8: SocketCAN communication layers [11].

Indeed, in Linux, connecting to a CAN socket is the same as connecting to any other socket. The following C code is enough to set up the CAN communication and bind to the socket, allowing the exchange of CAN messages with the network.

```
int s;
struct sockaddr_can addr;
struct ifreq ifr;

s = socket(PF_CAN, SOCK_RAW, CAN_RAW);

strcpy(ifr.ifr_name, "can0" );
ioctl(s, SIOCGIFINDEX, &ifr);

addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

bind(s, (struct sockaddr *)&addr, sizeof(addr));
```

The bytes being exchanged in the CAN network can be read to a CAN frame structure in the following way:

```
struct can_frame frame;
nbytes = read(s, &frame, sizeof(struct can_frame));
```

Writing CAN frames to the network can be done in a similar way:

```
nbytes = write(s, &frame, sizeof(struct can_frame));
```

The basic CAN frame structure is defined in *include/linux/can.h* as:

```
struct can_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR */
        __u8    can_dlc; /* frame payload length in byte  */
        __u8    __pad;   /* padding */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[8] __attribute__((aligned(8)));
};
```

The C code presented above is enough to start to perform specific actions and interact with a CAN bus in a Linux OS. In addiction to the CAN device drivers, SocketCAN provides several user-space utilities and applications to interact with the CAN network devices: `can-utils`.

Can-utils

`Can-utils` is a SocketCAN package with several tools that makes it easy to send, receive and analyse CAN messages. This package can be installed with the following command:

```
$ sudo apt-get install can-utils
```

In the context of this work, the following tools of the `can-utils` package were used [26]:

- `candump`: dumps all the received CAN packets to the console (Listing 3.1). As there are countless messages being exchanged between ECUs, the output of this tool is very unorganised and it is hard to discover any patterns to identify a particularly message. Luckily, the `candump` output can be filtered using, for instance, the tool `grep`, that allows to print only the CAN messages of one particularly identifier (Listing 3.2).

```
$ candump can0
6D5  [8]  00 00 00 44 40 40 40 15
101  [1]  04
38D  [8]  02 00 00 00 00 00 00 00
325  [2]  01 00
346  [8]  27 10 22 00 20 00 00 1F
236  [8]  10 01 10 00 90 00 00 C3
308  [8]  00 00 00 00 10 00 00 00
285  [8]  07 D0 02 00 75 00 08 10
385  [8]  43 00 00 00 00 00 00 00
373  [8]  B1 B1 7F B8 0D 49 80 06
200  [8]  00 20 60 04 C0 00 C0 00
215  [8]  00 00 00 00 00 00 00 00
288  [8]  07 D0 27 10 00 09 11 10
696  [8]  03 E4 01 F2 41 00 27 10
29A  [8]  01 34 57 46 55 30 30 30
6FA  [8]  02 34 37 31 00 00 00 00
```

Listing 3.1: Dumpimp all the received CAN packets.

```
$ candump can0 | grep "236"
236  [8]  10 01 10 00 10 00 00 03
236  [8]  10 01 10 00 20 00 00 2B
236  [8]  10 01 10 00 30 00 00 33
236  [8]  10 01 10 00 40 00 00 7B
236  [8]  10 01 10 00 50 00 00 63
236  [8]  10 01 10 00 60 00 00 4B
236  [8]  10 01 10 00 70 00 00 53
236  [8]  10 01 10 00 80 00 00 DB
236  [8]  10 01 10 00 90 00 00 C3
236  [8]  10 01 10 00 A0 00 00 EB
236  [8]  10 01 10 00 B0 00 00 F3
236  [8]  10 01 10 00 C0 00 00 BB
236  [8]  10 01 10 00 D0 00 00 A3
236  [8]  10 01 10 00 E0 00 00 8B
236  [8]  10 01 10 00 F0 00 00 93
236  [8]  10 01 10 00 00 00 00 1B
```

Listing 3.2: Printing the messages with identifier 0x236.

- `cansniffer:` organises the received CAN packets, by only showing the CAN messages that are changing. This is very useful when trying to identify what CAN packets correspond to particular actions. For example, in the Listing 3.3, there are no actions being performed in the Mitsubishi i-MiEV and in the Listing 3.4 the pedal brake is being pressed. With this method the messages with the identifiers `0x231`, `0x285` and `0x377` can be associated with parameters related with the brake pedal.

```
 ID  data  ...
119  00 00 00 00 00 01 01 5B
149  EA 7F 00 FF 14 80 01 B7
156  00 00 00 00 12 21 11 F8
208  00 20 60 03 C0 00 C0 00
210  00 00 00 00 80 00 00
212  00 00 00 00 A7 D0 00 00
236  10 01 10 00 20 00 00 2B
288  07 D0 27 10 00 09 11 10
29A  01 34 57 46 55 30 30 30
300  00 1B 1F FF 87 D0 FF FF
373  B1 B1 7F B7 0D 49 80 06
384  00 00 00 25 00 64 63 00
3A4  07 10 90 9E 8B 61 00 75
695  04 00 01 00 FB 09 85 00
696  03 E4 01 F2 41 00 27 10
6FA  02 34 37 31 00 00 00 00
```

Listing 3.3: Cansniffer output with no external actions on the vehicle.

```
 ID  data  ...
119  00 00 00 00 00 01 07 15
149  E1 7F 00 FF 14 80 07 C6
156  00 00 00 00 12 21 17 B6
208  00 20 60 04 C0 00 C0 00
210  00 00 00 00 00 00 00
212  00 00 00 00 27 D0 00 00
231  00 00 00 00 02 00 00 00
236  10 01 10 00 E0 00 00 8B
285  07 D0 02 00 74 00 08 10
288  07 D0 27 10 00 0A 11 10
29A  01 34 57 46 55 30 30 30
300  00 1B 1F FF 07 D0 FF FF
373  B1 B1 7F B8 0D 49 80 06
377  04 9A FF FF 41 3A 04 00
384  00 00 00 25 00 63 63 00
695  04 02 01 03 EF 0B 15 00
696  03 E4 01 F2 41 00 27 10
6FA  02 34 37 31 00 00 00 00
```

Listing 3.4: Cansniffer output when the pedal brake is pressed.

- `cansend:` sends a single CAN frame to the network. This tool is very helpful to evaluate the result on sending specific messages to the CAN bus:

  ```
  $ cansend can0 123#445566778899AABB
  ```

  The command above sends a message with the identifier 0x123 and the data `0x44`, `0x55`, `0x66`, `0x77`, `0x88`, `0x99`, `0xAA` and `0xBB` to the network. This can also be used to send the same message at a specific rate, doing the follow command:

  ```
  $ while true;
  do cansend can0 123#445566778899AABB;
  sleep 0.002;
  done;
  ```

- `canplayer:` replays CAN packets previously saved.

- `cangen:` generates random CAN frames and sends them to the network.

The last two presented tools (`canplayer` and `cangen`) are particularly interesting when working in a virtual CAN bus, as described below.

Virtual CAN

It is possible, using the `can-utils` tools, to create a virtual CAN bus, which is very useful when trying to simulate activity of a real CAN bus and for testing CAN software without having to be directly connected with the vehicle network [27]. The simulated bus can be created with the following commands:

```
$ sudo ip link add dev vcan0 type vcan
$ sudo ip link set up vcan0
```

The virtual CAN bus can be used in two main ways: generate random CAN packets (using the `cangen` utility) or play a recorded data file with CAN messages. This allows the use of previously saved messages from the car in motion, facilitating the test of programs.

3.3 Summary

The hardware components and software tools described in this chapter constitute the basis of this project. The ATLASCAR2 parts related to the steering and propulsion systems and the device utilized to communicate with the vehicle CAN bus through the OBD-II port were used as hardware. Regarding the software, it can be divided in two categories:

- ATLASCAR2 software: in order to insert this work in the global system, it was developed in the same Operating System (Linux OS) and framework (ROS) of the remaining project;

- specific software of this work: tools used to develop and test the solutions of this dissertation, which are the Linux support for CAN communication - SocketCAN and `can-utils`.

Intentionally blank page.

Chapter 4

Car Status Monitoring

One of the objectives of this dissertation is to develop a solution to update the vehicle status in the global system in real-time. As explained in later chapters, the ECUs of the car communicate with each other using the CAN protocol. The exchanged CAN messages contain information regarding the critical parameters of the vehicle, such as its velocity and direction, and non-critical parameters, such as information of the lights, doors and other less critical components.

This chapter explains how to access the CAN bus and relate the messages in this network to the state of the car in order to publish them in the global system.

4.1 Understanding CAN Protocol

Since the identification of the car status is performed using the vehicle network, it is important to have a deeper understanding of the CAN protocol and CAN messages.

In Chapter 2, CAN was introduced as the network used in vehicles to perform communication between ECUs. There are four main reasons that make this network so popular in the automobile industry [28]:

1. transmission rates are must faster in CAN communication - 500 kb/s - than in conventional communication (p.e. Local Interconnect Network (LIN), which provides speed up to 20 kb/s), allowing much more data to be sent [24];
2. ECUs communicating via the CAN interface extremely reduces the cost and complexity of the wiring structure of the vehicle than through analog signals;
3. the CAN bus allows a central diagnosis system available through the OBD-II port;
4. CAN communication is robust to failure of subsystems and electromagnetic interference, which makes it ideal for vehicles;

Indeed, the main purpose of CAN is to allow any ECU to communicate with the entire system without complex dedicated wiring: using the CAN bus, an ECU can broadcast information through only two wires - CAN high (CANH) and CAN low (CANL). Using these wires, CAN uses differential signalling, that is, to transmit a logic 1 the two lines can have the same level of 2.5V (Recessive State) and to send a logic 0 the voltage of the CANH line raises to 3.5V and in the CANL line drops to 1.5V, creating a differential voltage of 2.5V. Figure 4.1 explains the voltage levels of the CAN bus and their relation with the data transmitted.

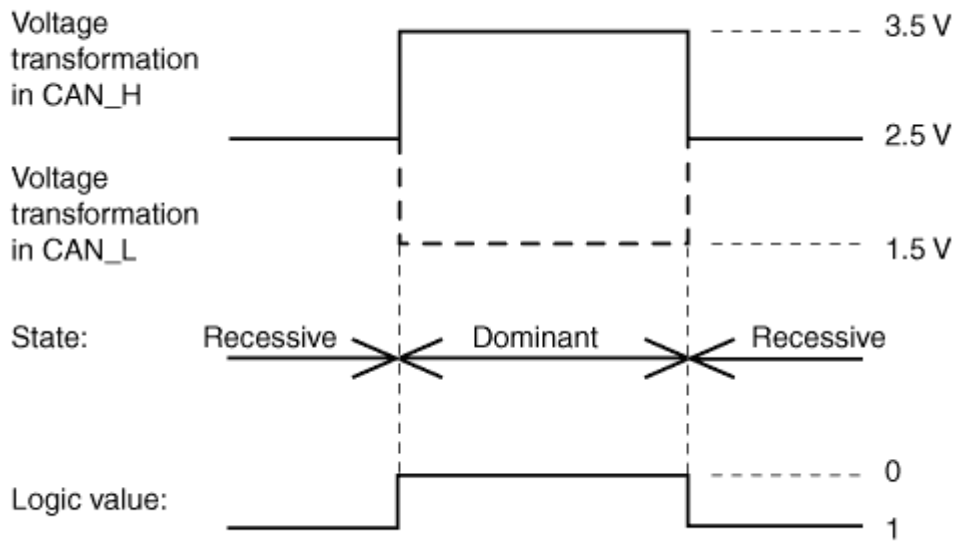


Figure 4.1: Voltage levels of the CAN bus to data transmission [24]

CAN Frames

The CAN communication is done via CAN frames. To understand the messages exchange between ECUs, it is important to know the structure of a CAN frame (Fig. 4.2):

Figure 4.2: Standard CAN frame structure, with the number of bits of each field [28].

- SOF (Start of Frame): logic 0 to indicate the other nodes (ECUs) the beginning of a CAN frame;
- ID (Identifier): identifies the data content;
- RTR (Remote Transmission Request): indicates whether the frames sends data or requests data from another node;
- Control: specifies the frame type and data length;
- Data: contains up to 8 bytes of data;
- CRC (Cyclic Redundancy Check): check for data errors. The nodes that send and receive the frame apply prescribed operations to the data field and the receiver node compares the values to detect errors in the transmission;

- ACK (Acknowledge): field used to indicate if the node has received the data correctly;
- EOF (End of Frame): indicates the end of the CAN frame.

Note that, in the applications used in this dissertation, the CAN ID and Data fields are the ones that require greater attention. The identifier of a message is used by the ECU to process or ignore the received frame, since it represents its priority - lower ID values have higher priority. In a vehicle, this is used to prioritize information, for example, a frame that contains information about the vehicle speed should have a lower ID than a message related with the radio system.

4.2 Identification of the Mitsubishi i-MiEV CAN Frames

4.2.1 Mitsubishi i-MiEV CAN Bus

As the standard CAN buses, the Mitsubishi i-MiEV CAN bus consists of the two lines, CAN high and CAN low, and two terminal resistors connected by the main bus line. The bus also has a sub-line that connects each ECU to the main bus (Fig. 4.3) [24].

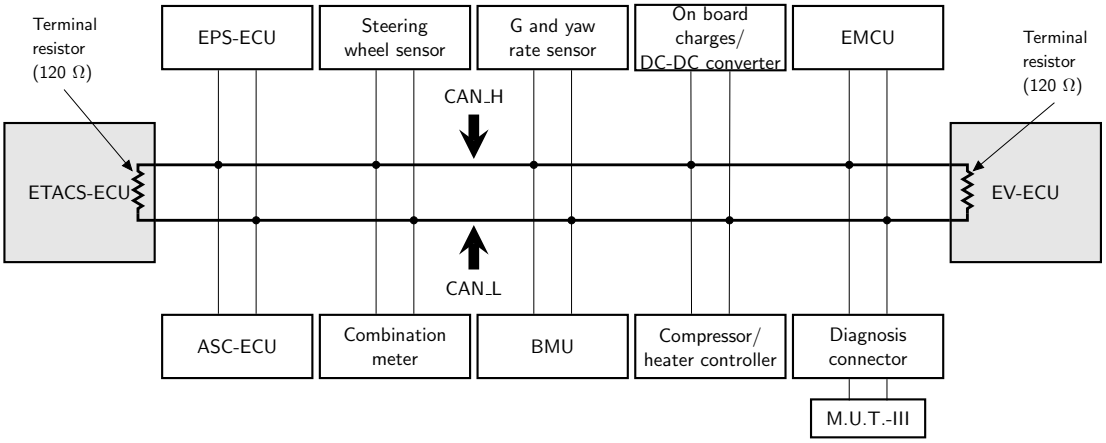


Figure 4.3: Mitsubishi i-MiEV CAN Bus [24].

As can be seen in the Figure 4.3, the components connected in the CAN bus are:

- ETACS-ECU: monitors doors, lights and other non-critical components;
- ASC-ECU: the Activate Stability Control system is responsible to reduce the vehicle speed when it detects that the vehicle is in a dangerous condition;
- EPS-ECU: controls the power steering system;
- Combination Meter: instrumental panel. All the warning lights and information in the panel are transmitted through the CAN bus;
- Steering Wheel Sensor: measures the steering force;
- BMU: Battery Management Unit;

- G and yaw rate sensor: detects the yaw rate and lateral acceleration of the vehicle;
- Compressor/heater controller: controls the temperature management of the interior of the vehicle;
- On board charger/DC-DC converter;
- Diagnosis connector: OBD-II port;
- EMCU: electric motor control unit;
- EV-ECU: vehicle main controller. Integrates information from the driver actions, vehicle and battery.

4.2.2 Locating and Reading the CAN Bus

Since the CAN bus packets are broadcast, all ECUs in the network see every packet, so, if a device is connected to the bus, all the communication will be available. The easier and more efficient way to connect to the CAN bus is using the diagnosis connector. The OBD-II port can usually be found under the vehicle dashboard and must be accessible without the need for tools.

In Figure 4.4 and Table 4.1,the pins of the OBD-II port and the respective assignments are shown.

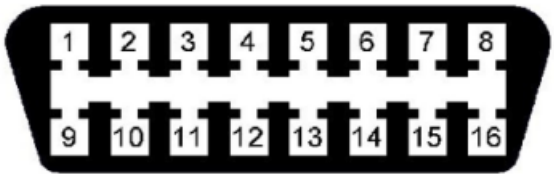


Figure 4.4: OBD-II port pinout [29].

Table 4.1: OBD-II port pinout.

Pin	Description	Pin	Descripton
1	Manufacture Option	9	Manufacture Option
2	J1850 Bus (+)	10	J1850 Bus (-)
3	Manufacture Option	11	Manufacture Option
4	Chassis Ground	12	Manufacture Option
5	Signal Ground	13	Manufacture Option
6	CAN (J-2234) High	14	CAN (J-2234) Low
7	K-Line (ISO 9141-2)	15	L-Line (ISO 9141-2)
8	Manufacture Option	16	12V Battery Power

The pins of the OBD-II port dedicated to the CAN bus are the pins 6 and 14, respectively, the CAN high and CAN low lines. So, these pins alone enable the connection to the CAN bus, however, pins 4 and 5 (ground pins) and pin 16 (provides a constant supply of 12-volt power from the vehicle's battery) are also important. Indeed, the pin 16 will be crucial in the proposed solutions of this work, since it is used to provide power

to devices plugged into the OBD-II port without the need of an external power supplier. In the context of this dissertation, the manufacture dedicated pins, pins 2 and 10 (J1850 bus) and pins 7 and 15 (ISO 9141-2 bus) are not used.

To connect and communicate to the vehicle's CAN bus CANalyze is used with the techniques described in Chapter 3.

4.2.3 Perform Message Identification

The CAN packets received by the device connected to the OBD-II port have no information concerning which ECU sends them and the manufacturer does not provide information regarding the function of each message. So, the messages received using the tools described above are, at first sight, completely unknown. Thus, in order to relate the CAN frames with specific information, a reverse engineering of the CAN bus data must be done, that is, to evaluate the received CAN messages using the sniffer tool of `can-utils` while changing the car parameters.

The reverse engineer process can take quiet some time. Luckily, there is already a considerable amount of information available online about the meaning of CAN messages exchanged in the Mitsubishi i-MiEV. The meaning of some CAN messages fields and their relation with the car status can be obtained in [30], [31] and [29].

The conventional way to refer to the CAN data field is by corresponding each byte with the letter B and the number of the byte in the respective message: B0, B1, B2, B3, B4, B5, B6 and B7, since a CAN message has up to 8 bytes of data.

To obtain an AD solution, the most important parameters of the car are the ones related with its direction and velocity, because this information is crucial for the navigation and perception algorithms. Next, the identifier and fields of the CAN bus messages that have information related with the velocity and direction of the vehicle are presented.

Vehicle Speed

The value of the vehicle speed is directly obtain in the byte B1 of the message with the identifier `0x412`. This message also contains information regarding the total kilometeres of the vehicle in bytes B2, B3 and B4, and is given by:

$$Total\ km = (B2 \times 65536) + (B3 \times 256) + B4\ [\text{km}] \tag{4.1}$$

Also, the revolution per minute of the electric motor can be obtain in the CAN bus, using the message with ID `0x298` and byte B6:

$$Electric\ Motor\ Revolutions = B6 \times 256 - 1000\ [\text{rpm}] \tag{4.2}$$

Steering Angle

The angle of the steering wheel is transmitted in the ID `0x236` in the B0 and B1 bytes:

$$Steering\ Angle = \frac{(B0 \times 256 + B1) - 4096}{2}\ [^{\circ}] \tag{4.3}$$

Position of the Accelerator Pedal

The message with ID `0x210`, in byte B2, contains information about the current position of the accelerator pedal in percentage, being 0% the pedal released and 100% fully pressed:

$$Accelerator\ Pedal\ Position = \frac{B2}{250} \times 100 \tag{4.4}$$

Position of the Brake Pedal

Similarly to the accelerator pedal, the position of the brake pedal circulates in the CAN bus, in the message with ID `0x208` and bytes B2 and B3:

$$Brake\ Pedal\ Position = \frac{(B2 \times 256 + B3 - 24576)}{640} \times 100 \tag{4.5}$$

There is also the information if the pedal brake is pressed or not, in the ID `0x231` and byte B4, being `0x00` the pedal released and `0x02` pressed.

There are also other information elements of the vehicle that, not being as important as the parameters presented above, can be used in the continuation of this work, to control non-critical actuators of the vehicle, such as lights, door locks and blinkers.

The following list contains other car parameters and its decoding in the Mitsubishi i-MiEV CAN messages:

- **Shift Position:** message with identifier `0x418` and byte B0 (Table 4.2).

Table 4.2: Relation between the shift position and the value of the first byte of the message 0x418.

B0	Shift Position
0x44	Parking (P)
0x4E	Reverse (R)
0x50	Neutral (N)
0x44	Drive (D)

- **Autonomy of the vehicle**: message with identifier `0x424` and byte B1:

$$Autonomy = \frac{B1 - 10}{10}\ [\%] \tag{4.6}$$

- **Lights, Blinkers, Door Lock and Seat Belt**: All the information available in the instrumental panel is transmitted in the message `0x424`. The relation between this variable and the message data is presented in Table 4.3.

Table 4.3: Relation of message `0x424` with some of the instrument panel variables.

Byte	Bit	Variable
B0	b0	Seat Belt
B1	b6	Brake Lights
	b5	Main Beam
	b2	Dipped Beam
	b1	Left Blinker
	b0	Right Blinker
B2	b0	Door Lock

4.3 Summary

The monitoring of the vehicle status to the global system is performed by accessing the CAN bus of the vehicle, in which messages that contain information about its parameters are exchanged. By interpreting and processing these messages, a ROS topic with the following information is published in the global system:

- Speed of the vehicle;
- Steering wheel angle;
- Accelerator pedal position;
- State of the brake pedal;
- Brake pedal position;
- Shift position;
- State of the blinkers;
- State of the lights;
- State of the driver seat belt.

Intentionally blank page.

Chapter 5

Remote Control Solutions

As previously mentioned, the Mitsubishi i-MiEV uses digital units to control the steering and propulsion systems of the car. In this chapter, these systems are described, as well as the proposed solutions to remotely control the direction and speed of the vehicle.

In order to understand the logic behind the Mitsubishi i-MiEV operation, the vehicle's Workshop Manual and Technical Information Manual [24] were consulted.

5.1 Steering Wheel Remote Control

5.1.1 Mitsubishi i-MiEV Power Steering Logic

Today's cars have a power steering system that helps the driver steering the vehicle by reducing the effort required to move the steering wheel. In the Mitsubishi i-MiEV, an electric motor to add energy to the steering mechanism is used.

The electric motor is used in the steering system to control the EPS-ECU, which outputs current to the motor according to the vehicle speed and steering force. Indeed, it is crucial for this system to be sensitive to the speed of the vehicle, in order to allow a light steering force during stationary or low speed driving and a higher steering force during high speed driving. To pass the vehicle speed to the EPS-ECU, CAN communication is used, since, as mentioned in the previously chapter, this parameter runs on the CAN bus in the message with the identifier `0x418`. On the other hand, the steering force is measured using a torque sensor, which outputs two voltage signals (main and sub) to the EPS-ECU (Fig. 5.1).

In Figure 5.1, all the signals and information received by the EPS-ECU are represented; however, the steering operation can be simplified with the diagram of Figure 5.2 that is described in the following sequence:

1. torque sensor measures the steering force and outputs two voltage signals to the EPS-ECU according to that value;

2. based on the electric signals from the torque sensor and the vehicle speed, the EPS-ECU outputs the voltage for the electric motor;

3. the electric motor assists the steering operation in proportion to the voltage applied.

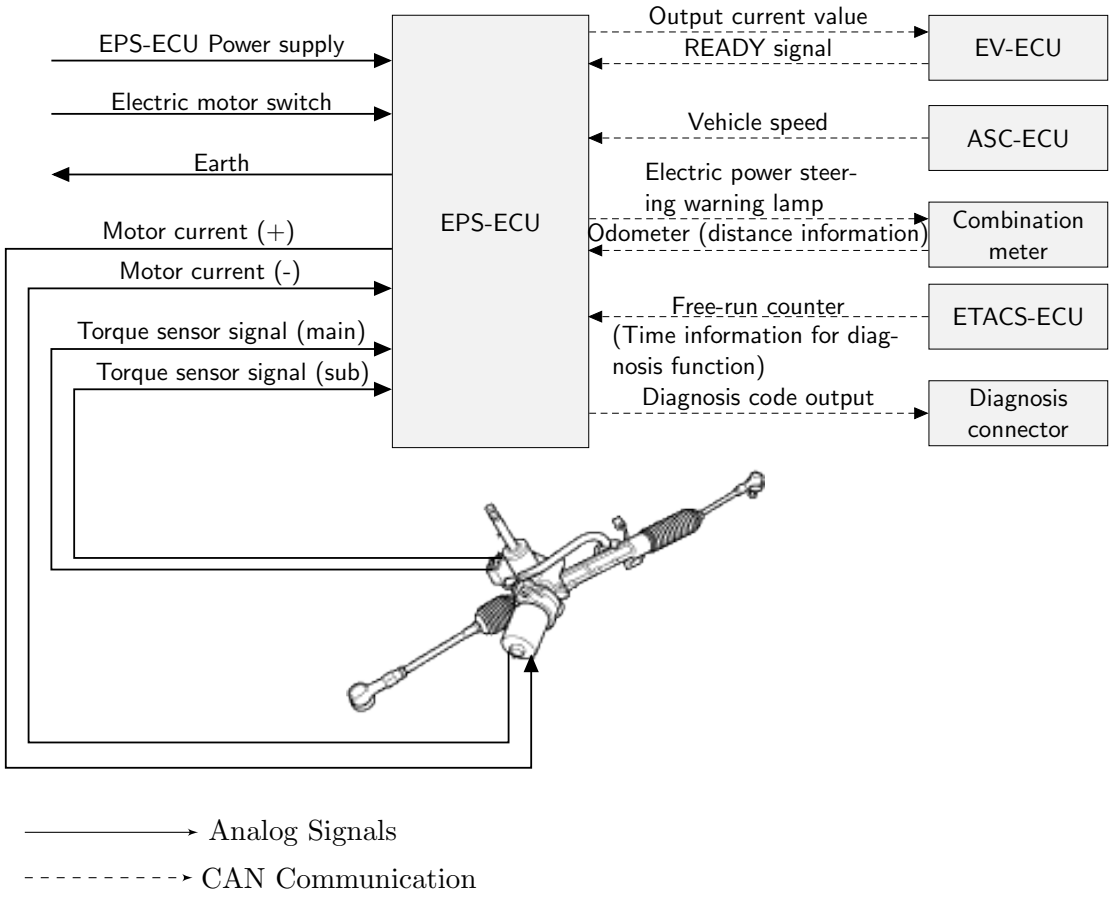


Figure 5.1: Electric Power Steering-ECU System Construction Diagram.

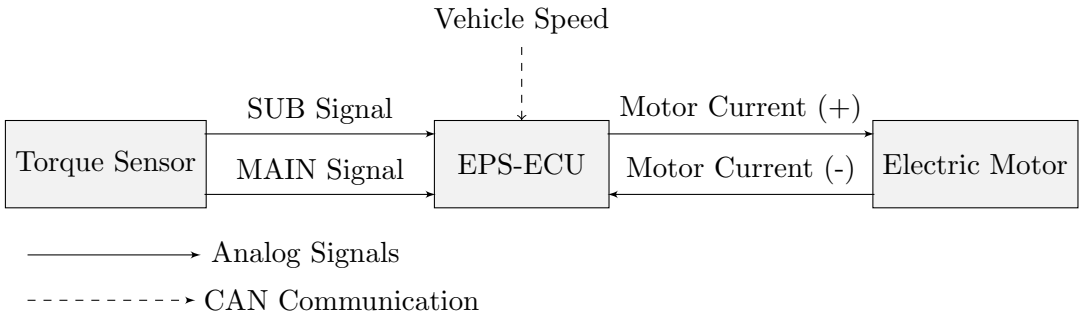


Figure 5.2: Operation of the Mitsubishi i-MiEV power system.

5.1.2 Proposed Solution

According to the operating logic of the Mitsubishi i-MiEV power steering system, there are two possible ways to remotely control the steering wheel: change the electric signals input of the torque sensor in the EPS-ECU and, therefore, change the current that the control unit outputs to the electric motor; or, directly input current to the electric motor. Due to greater accessibility to EPS-ECU than to the electric motor and simpler emulation of small voltage signals than power the current to an electric motor, the proposed solution is based on the first option.

In order to obtain the desired response of the steering wheel by replicating the signals from the torque sensor, it is important to understand the relationship between those signals and the steering angle.

Signal Analysis

The EPS-ECU has three dedicated connectors (Fig. 5.3):

- B-114: responsible to transmit the state of the electric motor unit (ON or OFF);
- B-114-1: outputs the current to the electric motor that assists the steering torque;
- B-114-2: connects the EPS-ECU to the torque sensor.

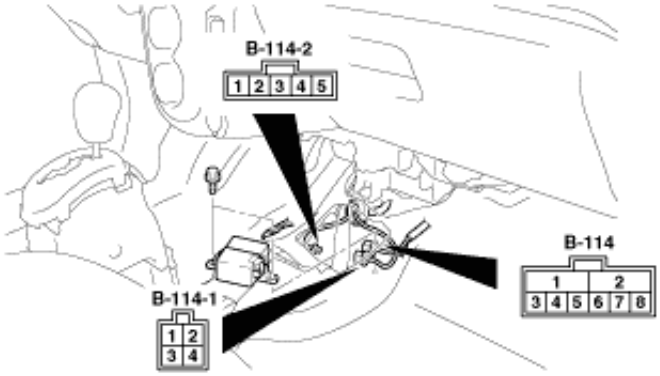


Figure 5.3: EPS-ECU connectors [24].

The proposed solution to control the steering wheel uses the signals from the B-114-2 connector. The Workshop Manual of the Mitsubishi i-MiEV has information about the function of each terminal of this connector, available in Table 5.1.

Table 5.1: Signals of the B-114-2 connector terminals [24].

Terminal No.	Check Item	Normal Conditions
1	Torque sensor main signal	0.5 to 4.5 V
2	Torque sensor sub signal	0.5 to 4.5 V
3	Torque sensor GND	0 V
4	Torque sensor power supply	4.5 to 5.5 V
5	Torque sensor shield GND	0 V

In Table 5.1, it can be seen that the first two terminals are the ones responsible for transmitting the signals from the torque sensor and the voltages values of these signals vary between 0.5 and 4.5 V. To understand how to use these signals to change the position of the steering wheel, the voltage values of terminals number 1 and 2 were analyzed while the steering wheel was manually turned.

The graphs of Figures 5.4, 5.5, 5.6 and 5.7 have the voltage measured in both signals of the torque sensor and the corresponding angle of the steering wheel. The angles were stipulated as positive when the steering wheel is turned to the left and negative when the steering wheel is turned to the right from a reference central position.

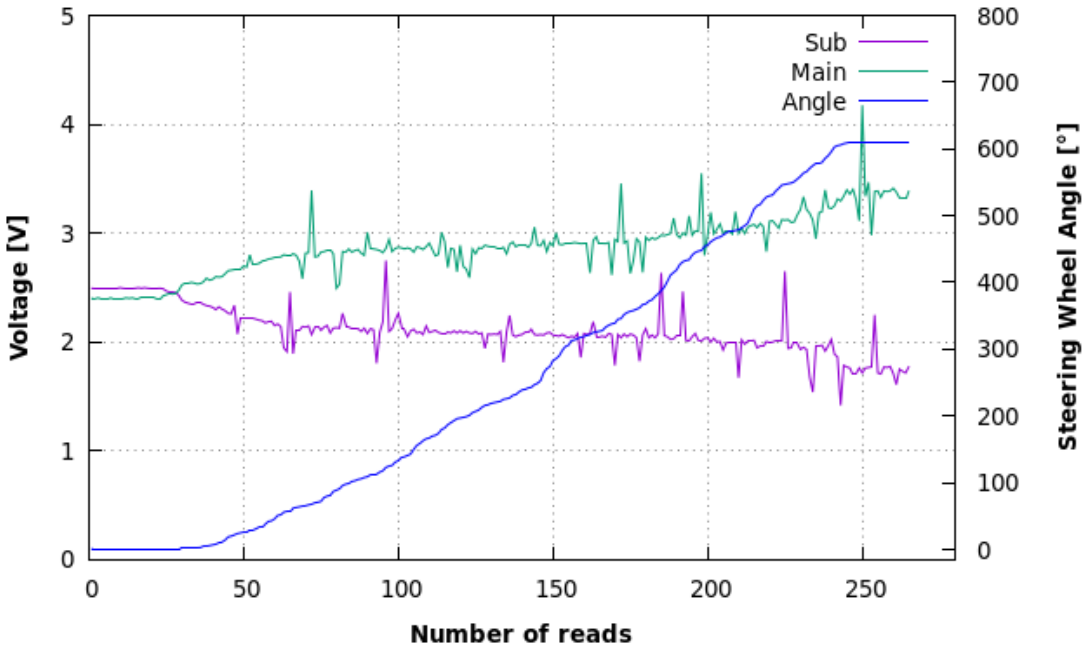


Figure 5.4: Voltage values generated by the torque sensor while turning the steering wheel completely to the left with the vehicle stationary.

In the graphs of Figures 5.4 and 5.5, the steering wheel is completely turned to the left and right at approximately constant rotational speed, as can be seen by the slope of the steering wheel angle line, with the vehicle in stationary state. Through the analysis of these graphical representations several conclusions can be drawn about the characteristics of the signals:

- the main and sub signals have, approximately, symmetrical voltages values relative to 2.5 V;
- when turning the steering wheel to the left, the main signal increases to the maximum of 3.2 V and the sub signal decreases to the minimum of 1.8 V. The opposite occurs when the steering wheel is turned to the right;
- the variation of the voltage values in both signals is approximately linear as the steering wheel is turned.

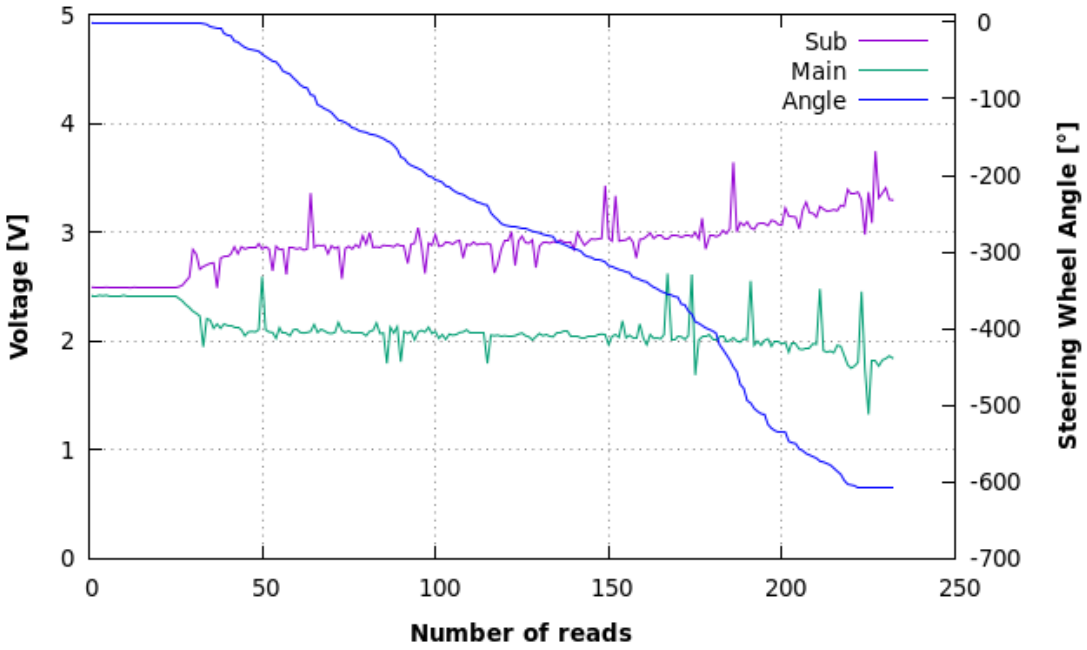


Figure 5.5: Voltage values generated by the torque sensor while turning the steering wheel completely to the right with the vehicle stationary.

In the graphs of Figures 5.6 and 5.7, the analyses represented in the graphs of Figures 5.4 and 5.5 were repeated with the car in a slow motion of 6 km/h. Comparing the two pair of graphs can be seen that with the vehicle in motion the voltage values are lower, ranging between 2 V and 3 V.

Though the presented graphs, it can be concluded that the torque sensor sends electrical signals between 1.5 V and 3.5 V based on the force applied by the driver in the steering wheel. This analysis is an indication on how to use these signals to remotely control the steering wheel, however, it is important to be aware that the power steering in the Mitsubishi i-MiEV is used to support the effort done by the driver and not to define the direction by itself, so the replication of these signals does not guarantee a similar positioning of the steering wheel.

Sending Signals to the EPS-ECU

The proposed solution emulates the main and sub signals from the torque sensor in the EPS-ECU and keeps the remaining terminals of the B-114-2 connector connected. In order to send the desire voltage to the B-114-2 connector of the EPS-ECU an Arduino UNO is used (Fig. 5.8). The specifications of the microcontroller can be found in Table 5.2.

Notice that, in the Arduino Uno, PWM (Pulse With Modulation) waves are used to create an output analog voltage. Briefly, PWM is a technique that uses digital control switching between on and off to generate a square wave, which can create voltages between 5 V and 0 V by changing the portion of time that the signal is on or off.

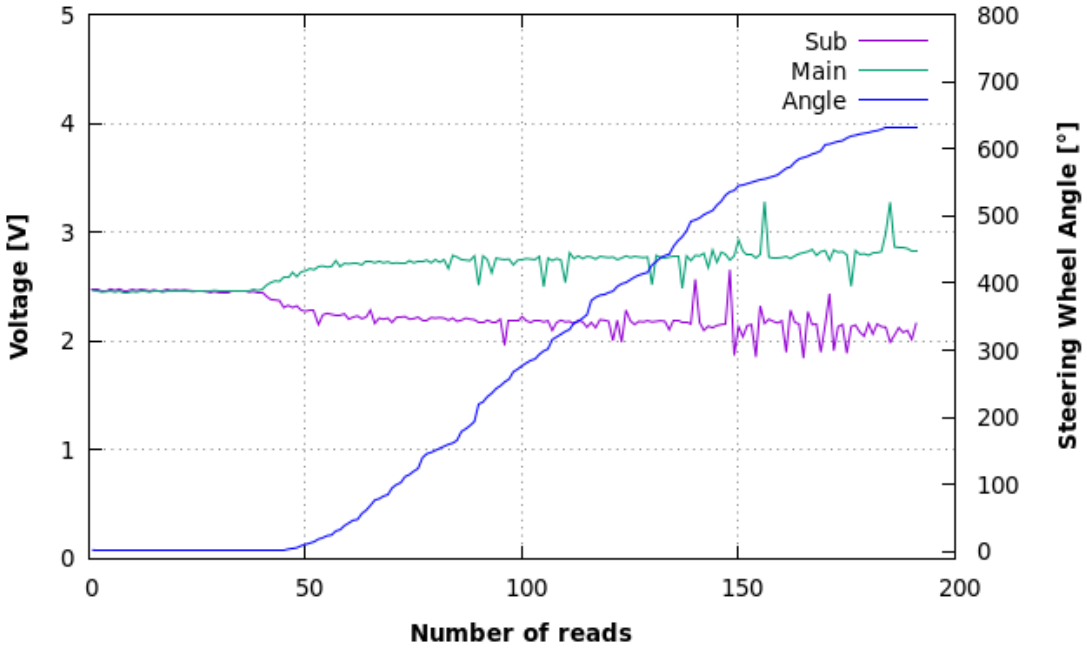


Figure 5.6: Voltage values generated by the torque sensor while turning the wheel completely to the left with the vehicle at the speed of 6 km/h.

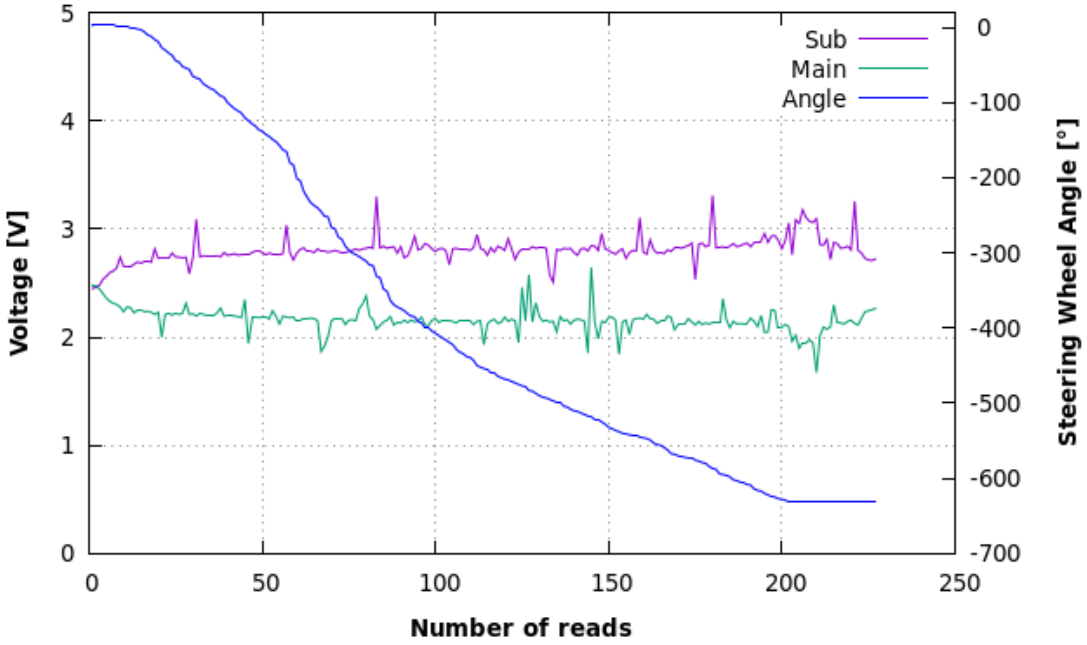


Figure 5.7: Voltage values generated by the torque sensor while turning the wheel to completely the right with the vehicle at the speed of 6 km/h.

Figure 5.8: Arduino UNO [32].

Table 5.2: Arduino UNO specifications [32].

Specifications	**Details**
Operating Voltage	5 V
Input Power Voltage	7-12 V
Digital I/O Pins	14
PWM Digital I/O Pins	6
PWM Pins	3, 5, 6, 9, 10, 11
PWM Frequency	490 Hz (pins 5 and 6: 980 Hz)

An application that controls the direction of a vehicle requires an accurate voltage, so, a steady voltage signal is needed. The PWM signal can be transformed into a smooth voltage by using a Low Pass Filter, which can be implemented with a resistor and a capacitor, as represented in Figure 5.9.

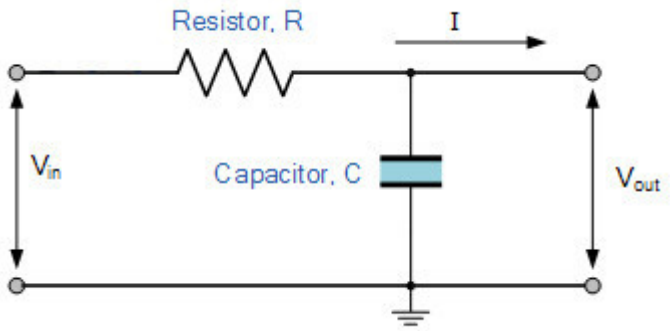


Figure 5.9: Low-pass filter [33].

The Resistor and Capacitor should be selected according to the desired cut-off frequency (f_c) of the filter, defined by Equation 5.1:

$$f_c = \frac{1}{2\pi RC} \text{ [Hz]} \tag{5.1}$$

The cut-off frequency determines the upper limit frequency where the input signal is allowed to pass through the filter. So, this value must be such that the filter is able to stabilize the PWM wave and, at the same time, does not cause a drop of the voltage. These considerations led to the use of a 220 Ω resistor and a 100 µF capacitor, resulting in a cut-off frequency of 7.2 Hz.

In Chapter 7 the tests performed using the method described in this section to control the steering wheel are presented.

5.2 Vehicle Speed Remote Control

In an electric vehicle, like the Mitsubishi i-MiEV, there are two actuators that are directly responsible for the speed of the vehicle: the electric motor and brakes. These actuators can be controlled by the driver using the accelerator and brake pedals. In the next sections, the logic behind the propulsion and braking systems of the Mitsubishi i-MiEV are explained.

5.2.1 Mitsubishi i-MiEV Propulsion Logic

As an electric vehicle, the Mitsubishi i-MiEV's propulsion is provided by an electric motor, which is controlled by the Electric Motor Control Unit (EMCU). This system is crucial for the proper operation of the vehicle, since it has direct impact in safety, performance, energy saving and comfort.

Although the EMCU is who directly controls the electric motor, the EV-ECU is the control unit that sends the torque command of the motor to the EMCU, using CAN

communication. The EMCU processes the torque command and determines the final torque. Based on this value, the EMCU outputs current to the electric motor unit (Fig 5.10).

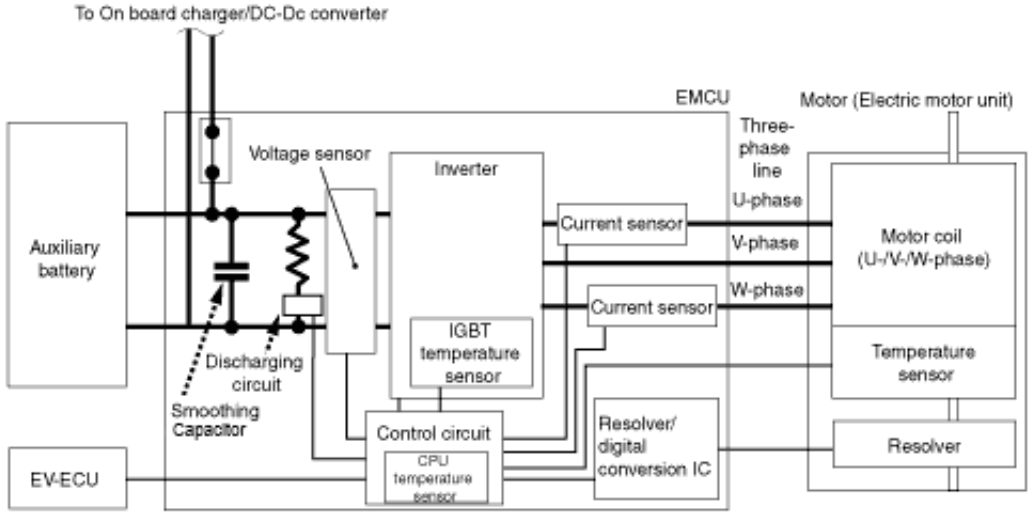


Figure 5.10: Electric motor control unit [24].

The EV-ECU calculates the torque command based on several instructions that are received via electric signals and CAN communication, as can be seen in Figure 5.11. In this work, the focus is on the variables that the driver controls to change the speed of the vehicle: the accelerator pedal opening (measured by the Accelerator Pedal Sensor - APS), the amount of pressure in the brake pedal (measured by the Brake Pedal Sensor - BPS) and the shift position switch. This dissertation does not include the shift control so, the focus will be on the two remaining variables.

The APS and the BPS use electric signals to transmit the position of respective pedals to the EV-ECU. In the case of the APS, the method is similar to the one used by the torque sensor, since it has a main and sub line, which sends electric signals to the EV-ECU.

In Figure 5.11, all the signals used by the EV-ECU to calculate the motor torque command to the EMCU can be seen. This pre-programmed software of the control unit has four main controls to determine the torque command [24]:

- Basic control: the torque command is calculated based on the vehicle speed and the accelerator opening, as in Figure 5.12;
- Power save control: when the main battery capacity becomes low, the power save control limits the motor (electric motor unit) output, turns off the A/C and heater, and illuminates the power down warning lamp on the combination meter.
- Shift position control: when the shift positions B or C of the vehicle are engaged. B (regenerative brake) represents the mode in which a stronger deceleration effect can be obtained without using the brake pedal, that is, by increasing regenerative brake effort. C (comfort) is a driving mode where the regenerative brake effort is decreased.

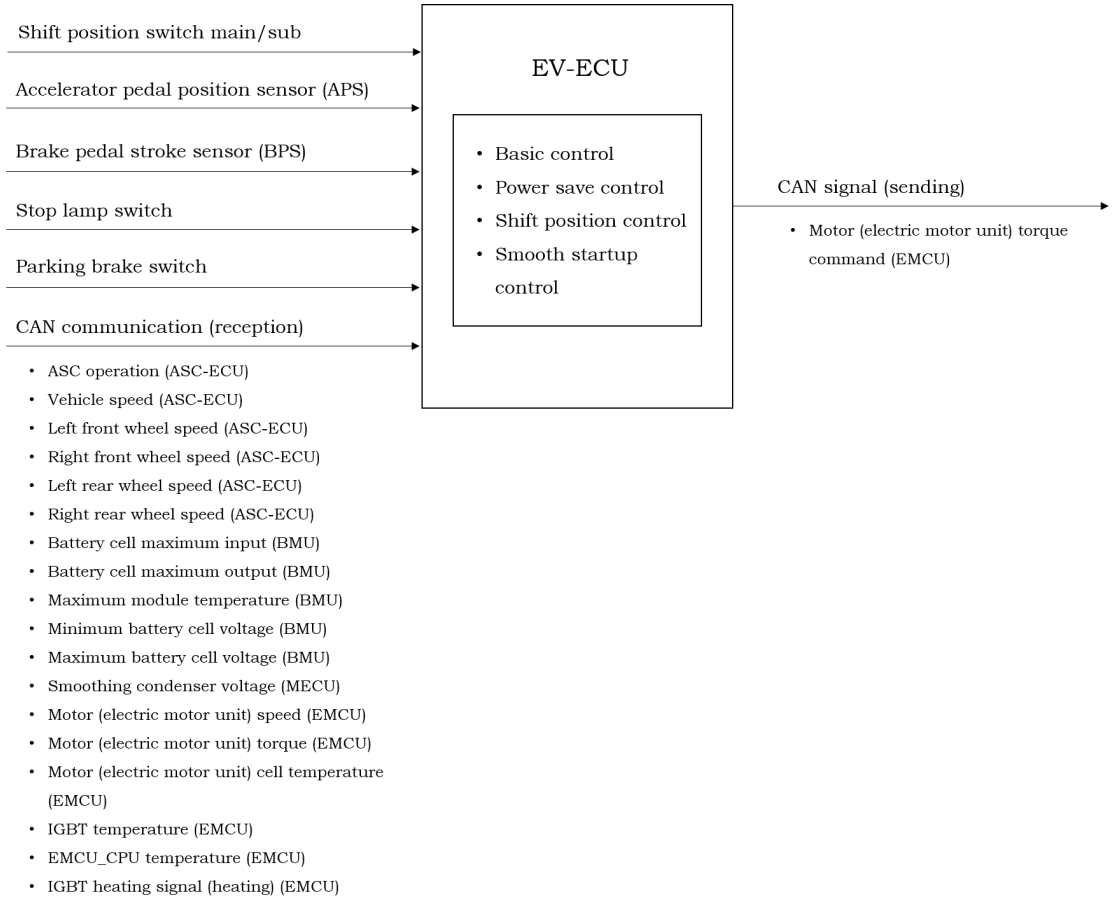


Figure 5.11: EV-ECU operation logic [24].

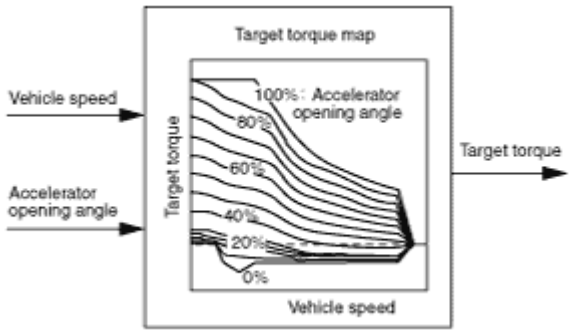


Figure 5.12: Torque command logic of the EV-ECU using basic control [24].

- Smooth startup control: suppresses the vibration generated in acceleration, ensuring a smooth driving.

5.2.2 Mitsubishi i-MiEV Braking Logic

The Mitsubishi i-MiEV braking system is a powered assisted conventional hydraulic system, thereby allowing an increased braking force with less brake pedal effort. As energy source of the brake booster, a brake electric vacuum pump is used (Fig. 5.13). In addition, the brake assist mechanism determines whether an emergency exists, based on the pedal depression speed and force, and provides the maximum brake force if needed.

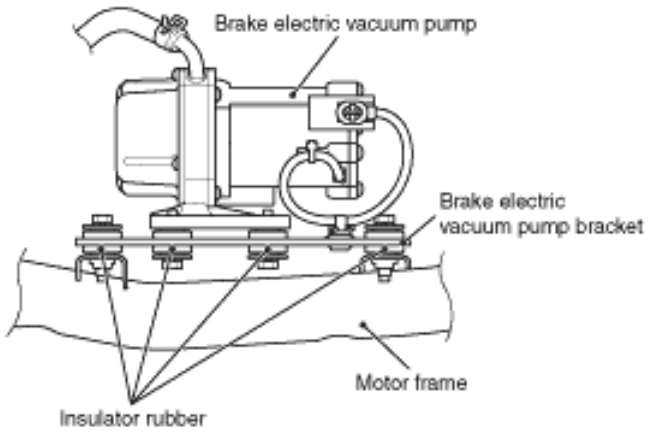


Figure 5.13: Electric motor used in the brake electric vacuum pump [24].

The brake vacuum pressure system, represented in Figure 5.14, uses the brake booster vacuum sensor to measure the vacuum pressure and sends an electric signal to the EV-ECU, which, based in this signal, controls the brake electric vacuum pump main relay and control relays, in order to activate/deactivate the brake electric vacuum pump.

Regenerative Braking

As most electric cars, besides the standard braking system, Mitsubishi i-MiEV also uses regenerative brake to reduce the speed of the vehicle. As represented in Figure 5.15, in regenerative brake, the motor revolution is converted to electric power to charge the main batteries of the car. Thus, when the accelerator pedal is free, the car slows down and the motor starts charging the batteries. As the car slows down the regenerative braking decreases until the car reaches its cruising speed of 6 km/h on a horizontal and flat road.

5.2.3 Proposed Solution

In order to properly control the speed of the ATLASCAR2, the propulsion and braking systems of the vehicle should be controlled.

According to the electric motor unit logic of the vehicle, there are two ways to remotely control the propulsion system: change the analog signals of the accelerator

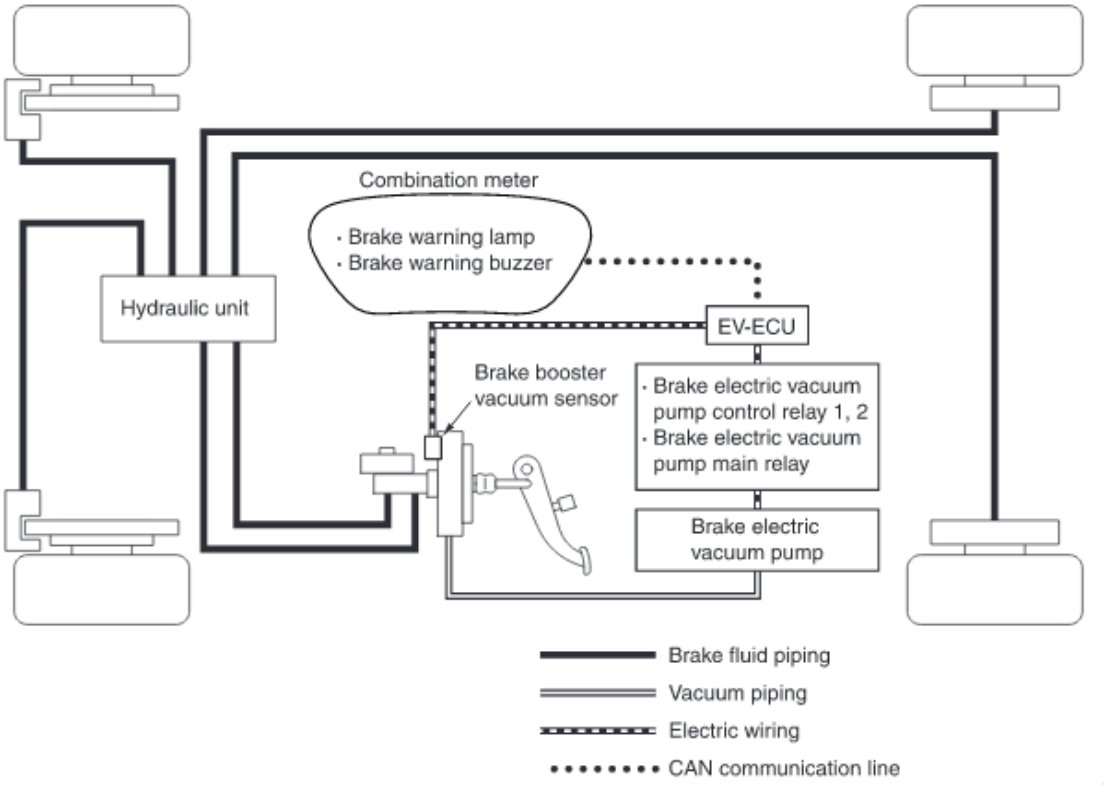


Figure 5.14: Mitsubishi i-MiEV braking system [24].

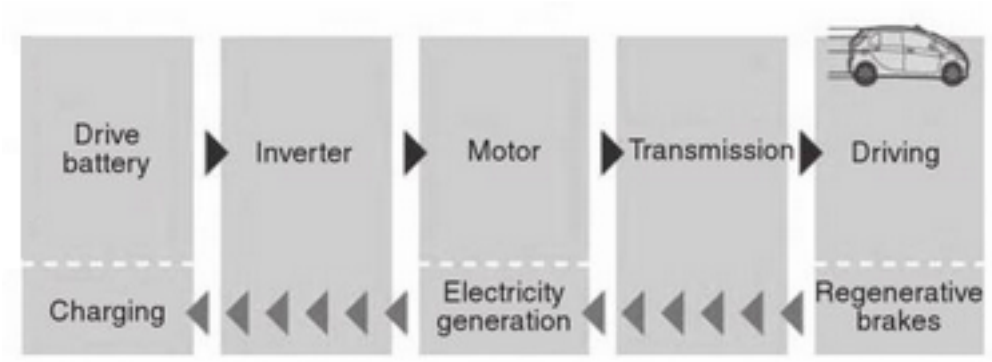


Figure 5.15: Regenerative braking system [34].

pedal that enters the EV-ECU, or change the digital CAN messages that the EV-ECU outputs to the EMCU with the electric motor torque command.

Concerning the braking system, since the hydraulic unit is directly actuated by the brake pedal stroke, the remote control possibility is based on a external mechanical actuator applying pressure on the brake pedal, similarly to one of the solutions presented in Section 2.2.1. Another possibility is to activate the brake vacuum generation system that assists the brake operation by changing the signals of the EV-ECU which controls the brake electric vacuum pump relays.

At the time of writing, the vehicle speed is controlled through acceleration and regenerative braking by changing the electric signals inputs of the APS in the EV-ECU. Obviously, a self-driving vehicle must have the possibility of acting on the brakes, however, with a proper use of the regenerative brake system, most of the brake actions can be done without using the brake pedal, especially if having the shift position corresponding to the regenerative braking (B) engaged.

Signal Analysis

As previously mentioned, the APS uses two electrical signals (main and sub) to transmit the information of the accelerator pedal opening to the EV-ECU.

The EV-ECU has four dedicated connectors with about twenty terminals each: C-106, C-108, C-110 and C-111. The C-106 and C-108 connectors are the ones responsible for transmitting information regarding the accelerator pedal position. Table 5.3 has information about the input signals of the EV-ECU related to the APS.

Table 5.3: EV-ECU terminals regarding accelerator pedal position [24].

Connector	Terminal No.	Check Item	Normal Condictions
C-106	8	Accelerator pedal position sensor (sub) power supply voltage	4.9 to 5.1 V
C-106	23	Accelerator pedal position sensor (sub) power supply earth	1 V or less
C-106	35	Accelerator pedal position sensor (sub) signal	0.3 to 2.5 V
C-108	42	Accelerator pedal position sensor (main) power supply	4.9 to 5.1 V
C-108	48	Accelerator pedal position sensor (main) power supply earth	1 V or less
C-108	59	Accelerator pedal position sensor (main) signal	0.8 to 4.8 V

According to Table 5.3, in order to emulate the signals of the APS, terminals 35 and 59 of the connectors C-106 and C-108, respectively, should be used. The vehicle's

Technical Information Manual provides information regarding the relationship between the voltage of both lines of the APS and the amount of accelerator pedal opening. Figure 5.16 shows that both signals increase proportionally with the position of the accelerator pedal and the main signal shows 1 V for the released pedal and 4 V in the full throttle point. The sub signal has always half the voltage of the main one.

In order to confirm the voltage values from the APS in the ATLASCAR2, measurements of these values were made pressing the accelerator pedal completely, which is represented in the graph of Figure 5.17. As can be seen, this graph points out a small difference from the information presented in the vehicle's Technical Information Manual, which is that the voltage value corresponding to the full throttle point is 4.5 V in the main line and not 4 V, as indicated in Figure 5.16.

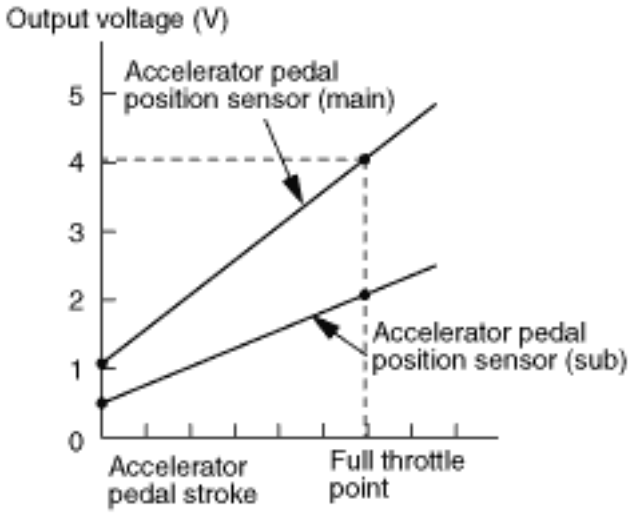


Figure 5.16: Relation between the voltage of the accelerator pedal position sensor main and sub line and the opening of the accelerator pedal [24].

Sending Signals to the EV-ECU

The method used to send electric signals to the EV-ECU is identical to the process described to send voltage to the EPS-ECU in the steering wheel control - an Arduino UNO associated with a Low Pass Filter.

In Chapter 7, the tests made to demonstrate the solution proposed in this section are presented.

5.3 Summary

In this chapter, the process used to obtain a solution to control the vehicle's steering and propulsion systems was explained and can be summarized as following:

1. understand the operating logic of the Mitsubishi i-MiEV;
2. recognize how to take advantage of the vehicle's operating logic to remotely control the respective systems;

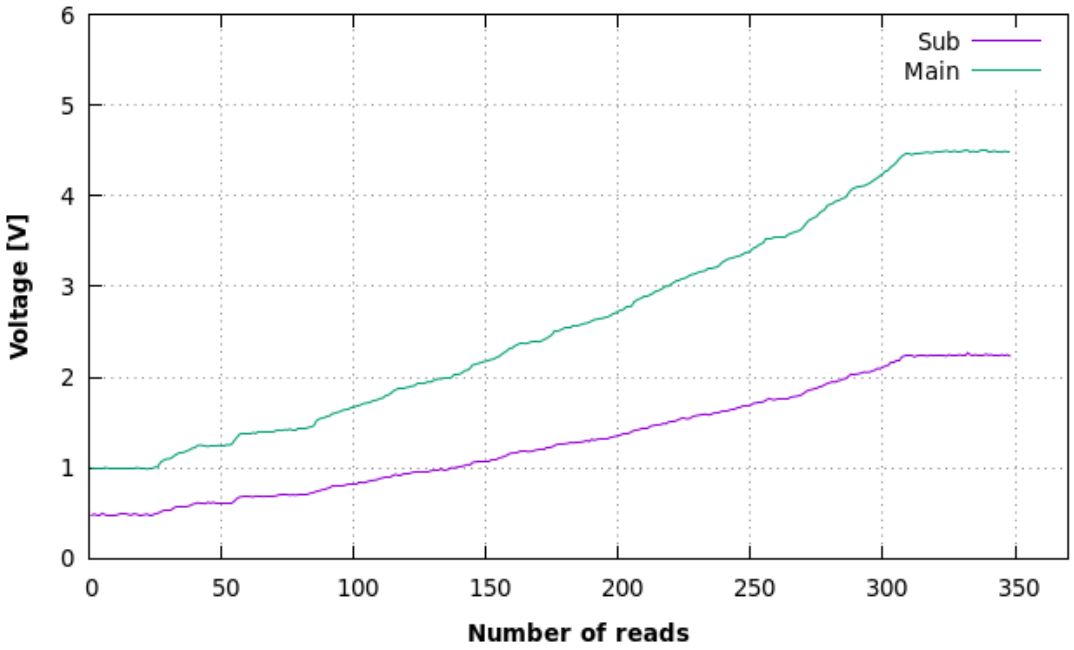


Figure 5.17: Voltage values of the accelerator pedal position sensor main and sub lines when the accelerator pedal is pressed completely.

3. analyze the existing signals when the systems are being operated manually;
4. realize how to replicate these signals to get similar results on the actuators.

Intentionally blank page.

Chapter 6

Actuators Control Management

In Chapter 5, the controllers used to define the speed and direction of the vehicle were described. This chapter explains how the desired values of the variables are passed to the controller. In addition, a solution is presented to make compatible the manual and remote/autonomous actuation of the vehicle's steering wheel and accelerator pedal.

6.1 Remote Control Using the CAN Bus

In a self-driving car, the values of the speed and direction of the vehicle, calculated by the navigation, perception and decision making algorithms, must be passed to the actuators' controllers in a reliable and efficient way. For that, it is possible to take advantage of the vehicle CAN bus and use the CANalyze device and the implementation presented in Chapter 4 to send CAN messages containing the values of the variables to control. These CAN messages will then be available to all the devices connected to the network so, the desired values can be passed to the controller by connecting it to the CAN bus network and enabling it to read and process CAN frames.

The implementation of the controller interpreting CAN messages and outputting voltage signals to perform specific actions is called External ECU.

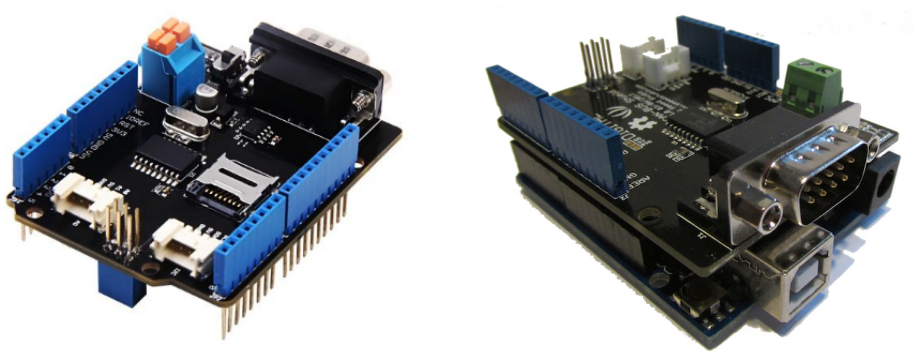
6.1.1 External ECU

In order to create an External ECU, the Arduino UNO, responsible to calculate and send the voltage signals, also needs to be able to receive and process the CAN messages. For that, a CAN-BUS Shield (Fig. 6.1a) is used incorporated with the Arduino UNO (Fig. 6.1b), enabling it to communicate with the CAN bus. The Listing A.1 on Appendix A is the Arduino IDE code used to receive CAN messages in the Arduino UNO.

This work uses two external ECUs, one for each controller. So, there are three devices that need to be connected to the vehicle's CAN bus through the OBD-II port of the car, so an OBD-II splitter is used, as shown in Figure 6.2.

The devices used to control the vehicle's direction and speed are summarized in the following topics:

- **CANalyze:** sends the instructions to the controllers via CAN bus;
- **External ECU 1:** reads and responds to the commands regarding the steering operation and generates the respective signals;



(a) CAN-BUS Shield [35].

(b) CAN-BUS Shield working with Arduino UNO [36].

Figure 6.1: External ECU.

Figure 6.2: Using an OBD-II port splitter in the ATLASCAR2 to connect all the devices to the vehicle's network.

- **External ECU 2:** reads and responds to the instructions concerning the speed of the vehicle and generates the respective signals;

6.1.2 Creating CAN Messages

In order to use the CAN bus to transmit the information to the controllers, specific CAN messages with their own identifiers were created. These identifiers must be different from the ones being exchanged in the vehicle to avoid the collision of messages. The created messages are described in Table 6.1.

Table 6.1: Description of the CAN messages created to send commands to the controllers.

Variable	ID	B0	B1	B2	B3	B4	B5	B6	B7
Steering Wheel Angle	`0x500`	Angle in Degrees	Left/Right Indication	-	-	-	-	-	-
Speed of the Vehicle	`0x501`	Speed in km/h	Emergency Level	-	-	-	-	-	-

In the message regarding the steering wheel control, the value of the angle is directly passed in the byte B0. Also, the byte B1 refers to the direction of the angle, being `0x00` left and `0x01` right.

The message used to command the speed of the vehicle has the value of the speed on byte B0 and, in byte B1, it contains information regarding emergency level of the specific actuation. This field is adopted to adjust the controller response speed to the urgency of the situation. An obvious example of the applicability of this field is when the vehicle must be immediately stopped due to an unpredictable event. The emergency levels are divided in three categories:

- `0x00`: normal conditions. The actuation is done considering a smooth driving operation;
- `0x01`: medium level. In this case the controller compromises the smoothness of the driving operation in order to reach the desired state faster;
- `0x02`: maximum emergency. The controller response must be as quick as possible, meaning that the brakes or accelerator are fully applied, depending on the situation.

6.2 Driving Modes Compatibility

Besides the possibility to remotely control the vehicle's actuators, a self-driving vehicle should be able to respond to direct actions on the steering wheel and accelerator pedal performed by the driver, as it would if the remote control systems were not installed on the car. Thus, the vehicle has two driving modes, autonomous and manual, that can be selected by the driver for the steering wheel and propulsion control systems.

As described in Chapter 5, the steering wheel and propulsion systems have a similar operation logic, that consists of a sensor sending two different electrical signals to the respective ECU. So, in both systems, the method used consists of receiving the electrical

signals from the vehicle sensor and the electrical signals from the controller (Arduino UNO) and select which signals to transmit to the respective ECU, depending on the desired driving mode.

The definition of the driving mode is performed by the driver using an on-off switch associated with the logic inputs of an analog multiplexer, which allows to change the output signals. Figure 6.3 shows the circuit used on the steering wheel. For the propulsion system the same electrical circuit is used, exchanging the terminals from the connector B114-2 and the terminals in the EPS-ECU to the connectors C-106 and C-108 and the respective terminals in the EV-ECU.

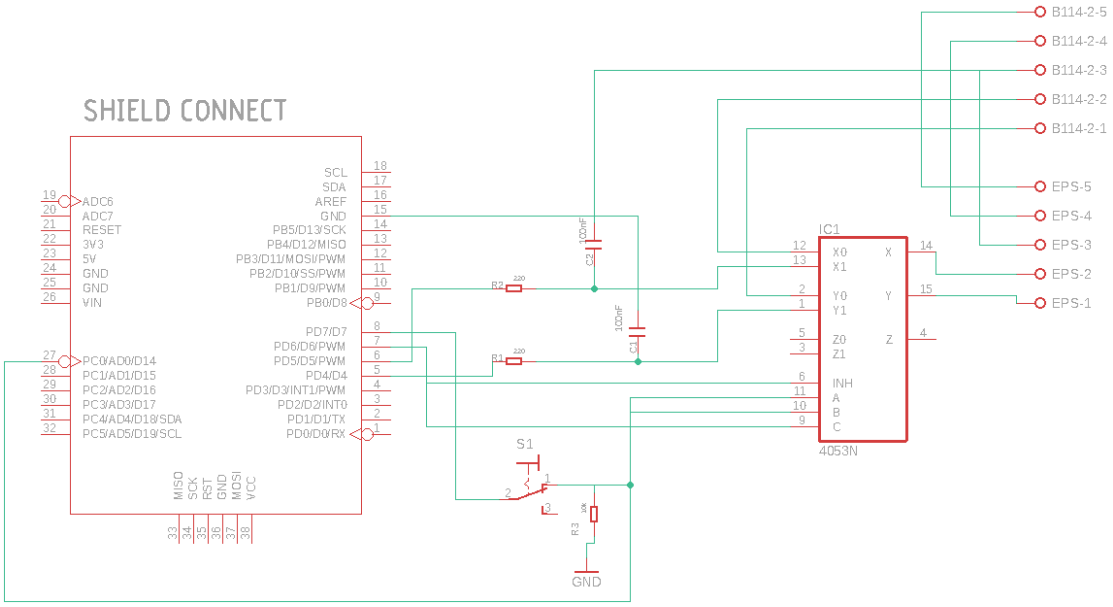


Figure 6.3: Circuit used in the transition between driving modes of the steering system.

In some situations, mainly due to safety reasons, the driver may need to change the direction and/or speed of the vehicle suddenly, not having the time needed to act on the switch to change to the manual driving mode. Thus, if the autonomous mode is on, and there is human intervention in the vehicle, the systems must give priority to the actions performed by the human operator. For that reason, it was implemented a feature in both systems that switches from autonomous to manual driving when the driver executes an action. In the steering wheel control system, this transition occurs when the driver turns the steering wheel, which is detected by the electrical signals from the torque sensor. For the propulsion system control, the driver can cancel the autonomous actuation by pressing the brake or accelerator pedal. The information about the state of the accelerator pedal is obtained by analyzing the electrical signals from the APS; in the case of the brake pedal, it is used the message with ID `0x231` to determine if the driver presses the pedal.

When the driver does one of the actions described above, a 5 second margin is then given to correct the vehicle status and decide whether the driver wants to continue in autonomous driving mode or change to manual driving mode permanently by using the switch. The diagram in Figure 6.4 summarizes the methods used to integrate the manual and autonomous driving modes.

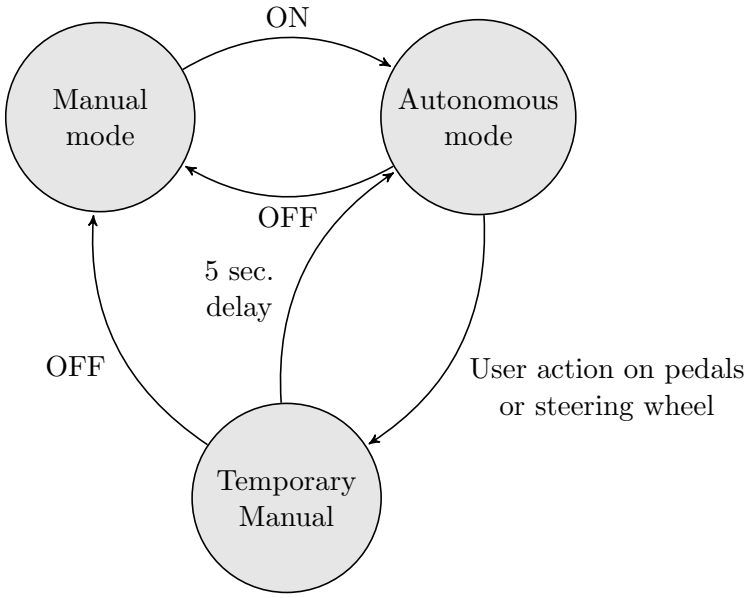


Figure 6.4: Transition between autonomous and manual driving modes.

In order to facilitate the assembly of the electrical circuit, a Printed Circuit Board (PCB) was developed to be integrated with the Arduino UNO and the CAN Bus Shield. This PCB is shown in Figure 6.5 and represents the circuit on Figure 6.3.

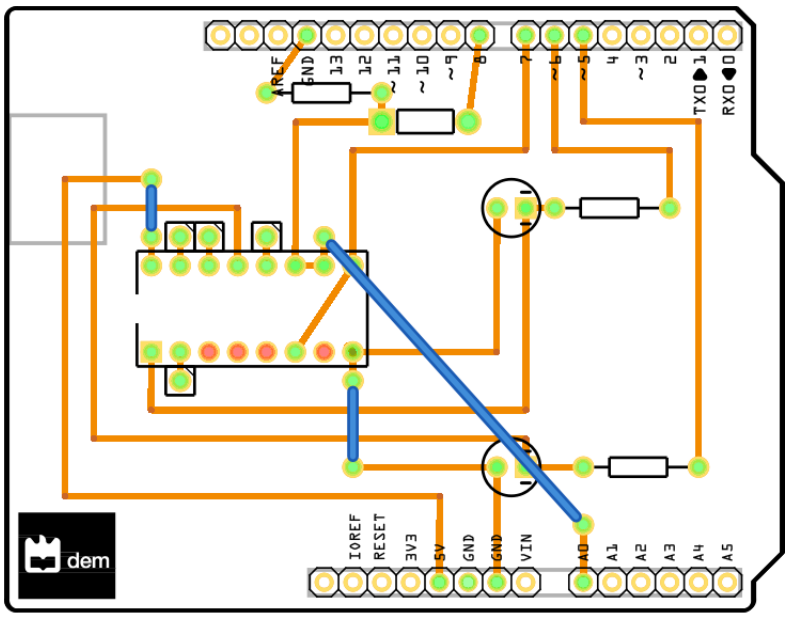


Figure 6.5: Circuit used in the transition between driving modes of the steering system.

Chapter 7

Tests and Results

In order to demonstrate the proposed solutions for the problems of this dissertation, tests were performed where the direction and speed of the vehicle are controlled without using the vehicle's steering wheel and pedals. In this chapter, those tests are described and the results are presented.

7.1 Direction Remote Control

As presented in Chapter 5, the approach used to control the steering wheel is to emulate the voltage sent by the torque sensor to the EPS-ECU. The analysis of the torque sensor signals done in that chapter revealed the main characteristics of these signals. Thus, in order to change the steering wheel position, similar types of signals were created using the Arduino UNO.

The graph in Figure 7.1 represents an experiment with the car at stationary state, where the torque sensor signals observed when turning the steering wheel to the left are replicated: the main signal increases linearly from 2.5 V to 3.2 V and sub signal decreases linearly from 2.5 V to 1.8 V. The graph demonstrates that the steering wheel rotates to the left, as expected. It can be seen that the steering wheel angle slope is not constant during the experiment – it increases rapidly when the main and sub signals reach 3 V and 2 V, respectively.

Also, can be noticed that the evolution of the steering wheel angle is not smooth, that is, it has an interrupted movement as the voltage values vary. The amplified representation of this experiment between the range of 0° and 70° of the steering angle in Figure 7.2 highlights this fact. The same experiment was performed rotating the steering wheel to the right and is represented in the graphs of Figures 7.3 and 7.4.

In order to understand the impact of different surface types in the response of the system, the same experiments were made using a higher and lower friction surface than the one used in the tests mentioned above, which was cement characteristic of a garage floor. The surfaces chosen to evaluate the effect of friction on the steering wheel position control model were the common tar on a normal road (higher friction) and sand (lower friction). The results of the experiments are presented in the graphs of Figures 7.5 and 7.6 for the rotation of the steering wheel to the left and, in for the rotation for the right, are presented in the graphs of Figures B.1 and B.2 in Appendix B. The results obtained show that the use of different surfaces with the same voltage values changes the results

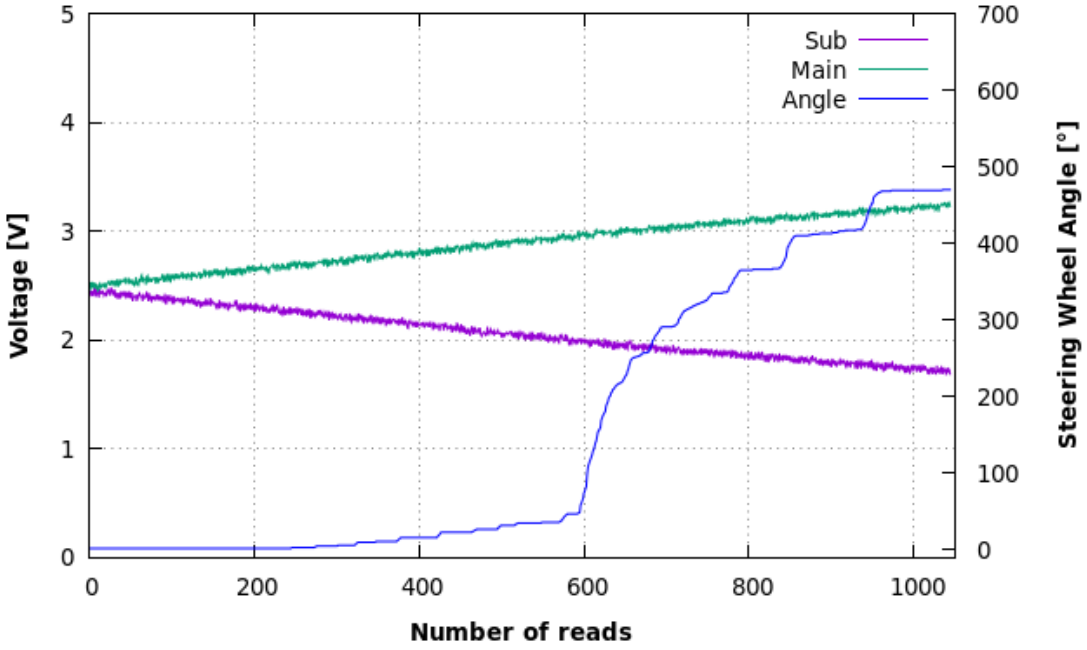


Figure 7.1: Steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU.

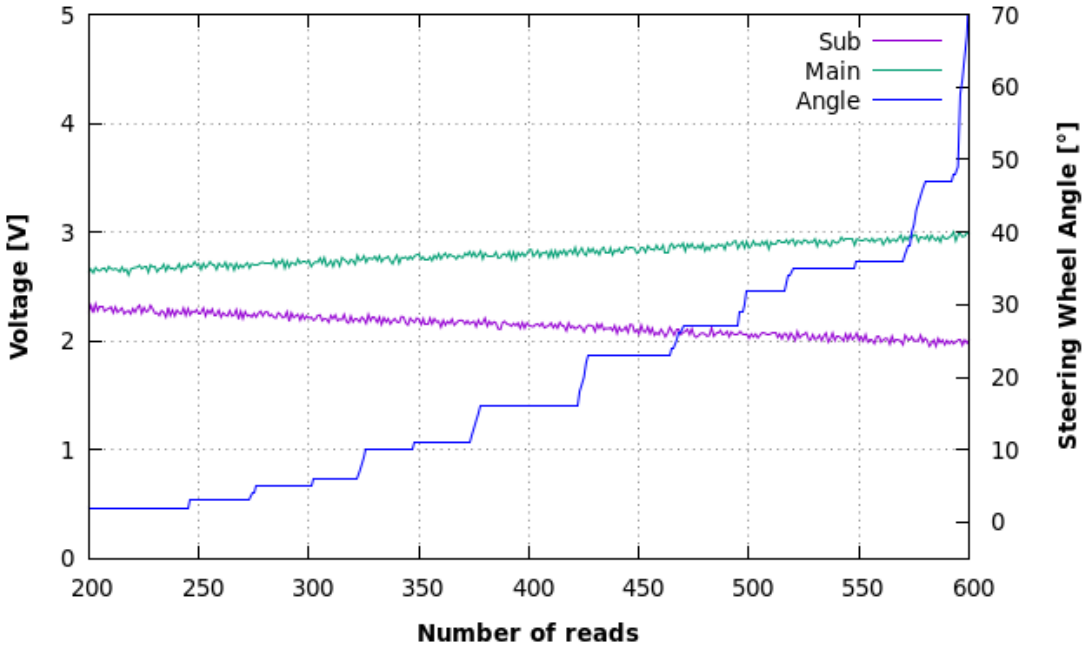


Figure 7.2: Detail view of the steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU.

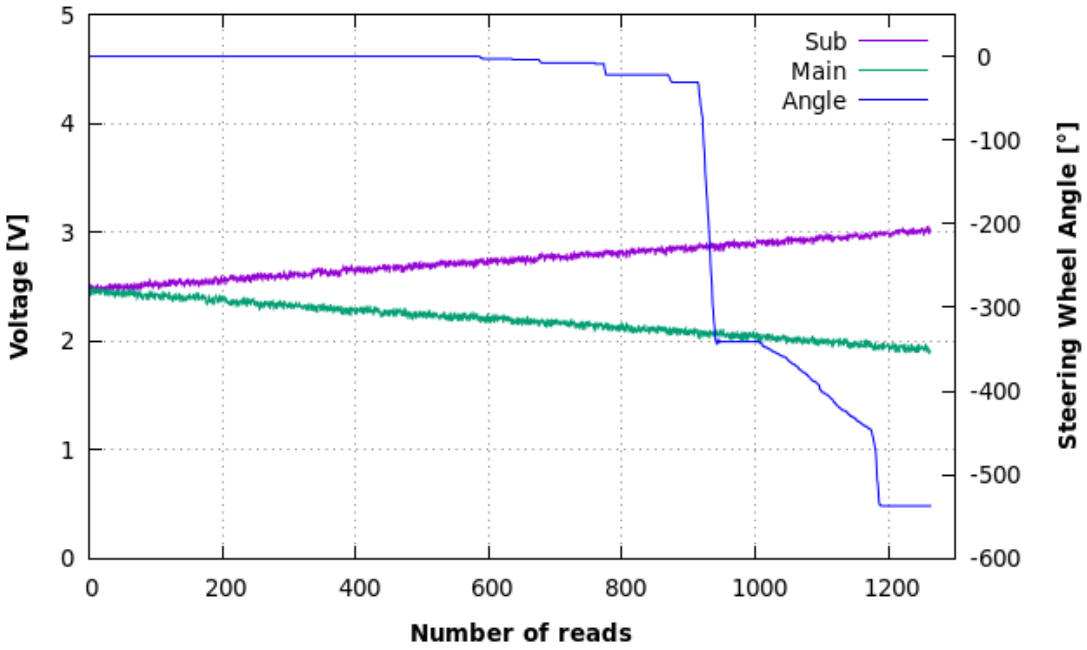


Figure 7.3: Steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU.

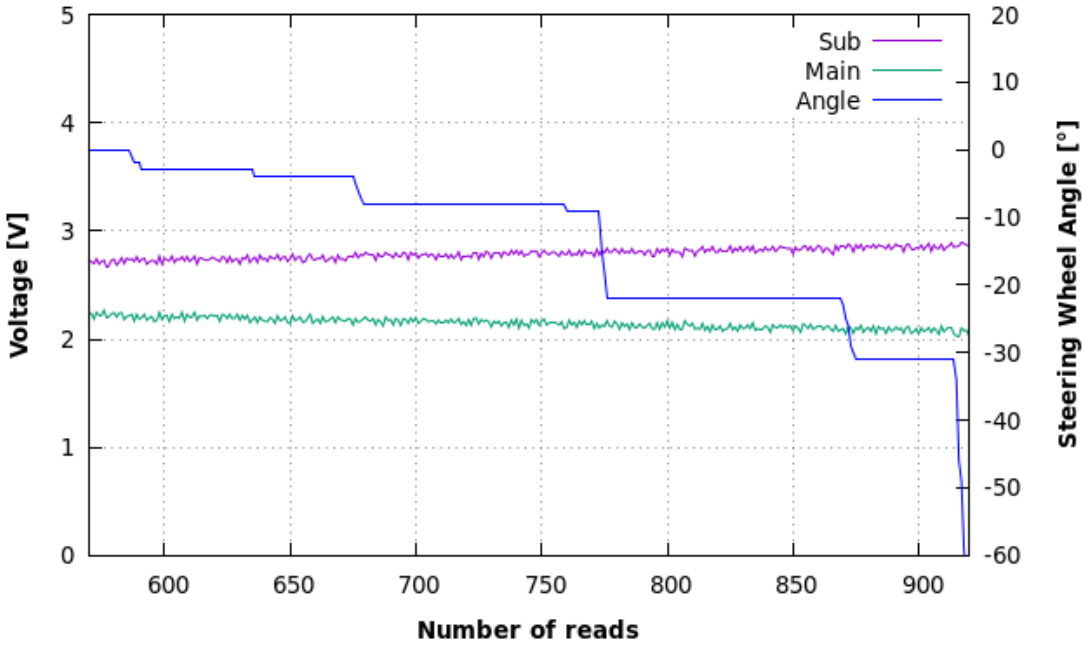


Figure 7.4: Detail view of the steering wheel position of the car in stationary state with the corresponding voltage values sent to the EPS-ECU.

obtained for the steering wheel position. For instance, the graphs of Figures 7.5 and B.1 indicate that using a higher friction surface, the maximum steering wheel angle obtain is 350°, which represents a decrease of more than 100° compared to the value obtained in previous experiments. On the other hand, the graph of Figure 7.6 and B.2 shows that in the lower friction case the maximum steering wheel angle reaches 550°, which is the maximum value obtained.

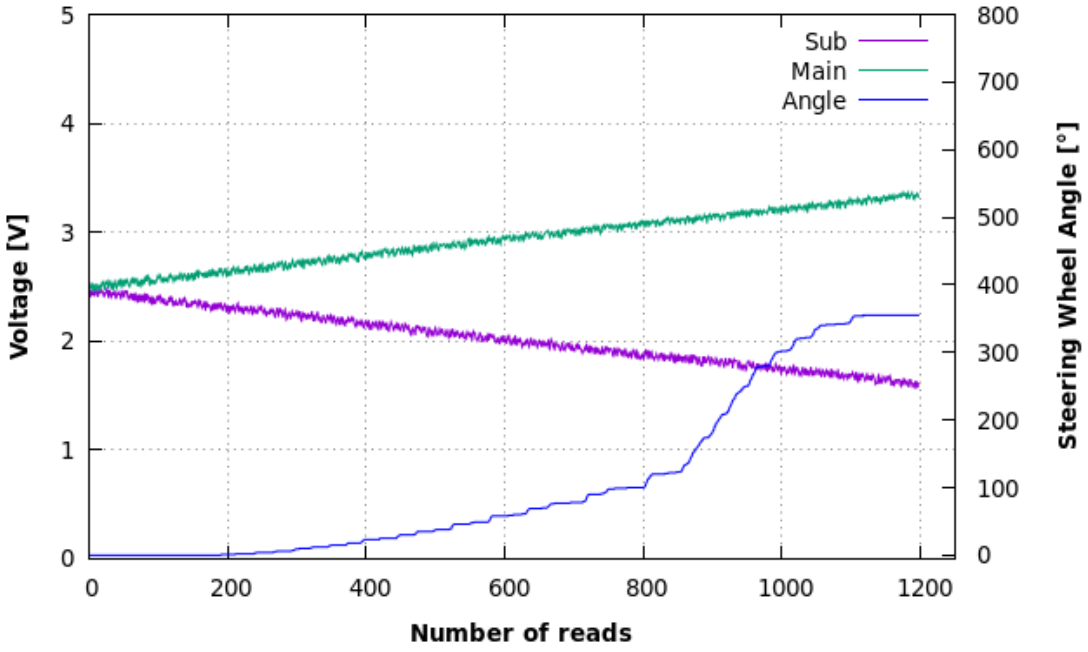


Figure 7.5: Steering wheel position of the car in stationary state over a surface of road tar with the corresponding voltage values sent to the EPS-ECU .

As previously mentioned, the EPS-ECU commands the electric motor based on the signals from the torque sensor and other variables, including the vehicle speed. Thus, in order to understand the impact of the speed of the vehicle in the response of the system, the same tests were made with the vehicle moving at cruise speed. The results are represented in the graphs of Figures 7.7 and 7.9, which represent the rotation of the steering wheel to the left and the right, respectively. Also, the results of these tests are represented in a smaller range of values of the steering wheel angle in Figures 7.8 and 7.10, in order to highlight the response of the system in the most common values of the steering wheel angle.

Comparing the results obtain in the experiments with the vehicle in stationary state with those where the vehicle is in motion, some differences in the response of the electric motor responsible to assist the steer can be noted. The following topics point out the distinctions observed in the graphs regarding the rotation of the steering wheel to the left (identical conclusions could be drawn in the case where the steering wheel is turned to the right):

- the graphs of Figures 7.1 and 7.7 show that the slope of the steering wheel angle line is approximately constant in the test with the vehicle in motion, whereas,

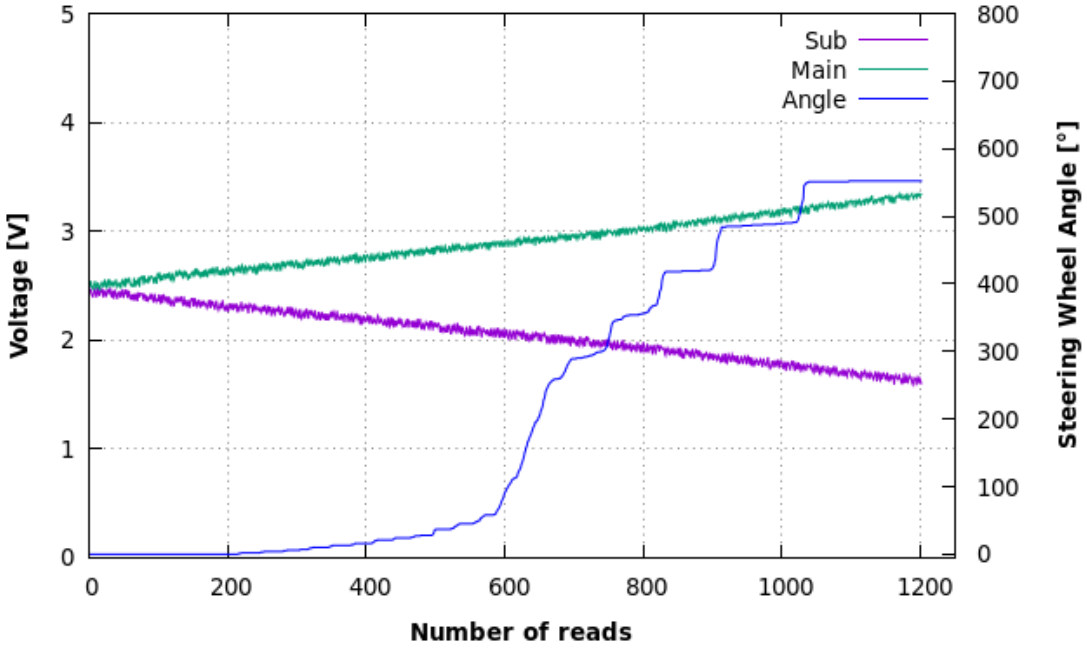


Figure 7.6: Steering wheel position of the car in stationary state over a surface of sand with the corresponding voltage values sent to the EPS-ECU.

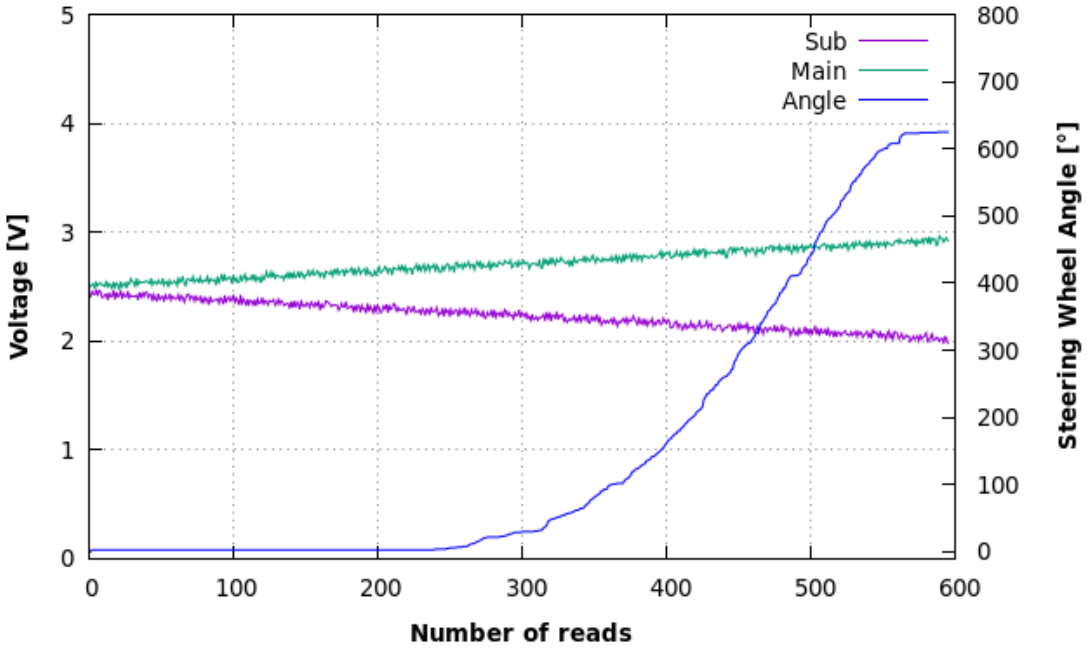


Figure 7.7: Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU.

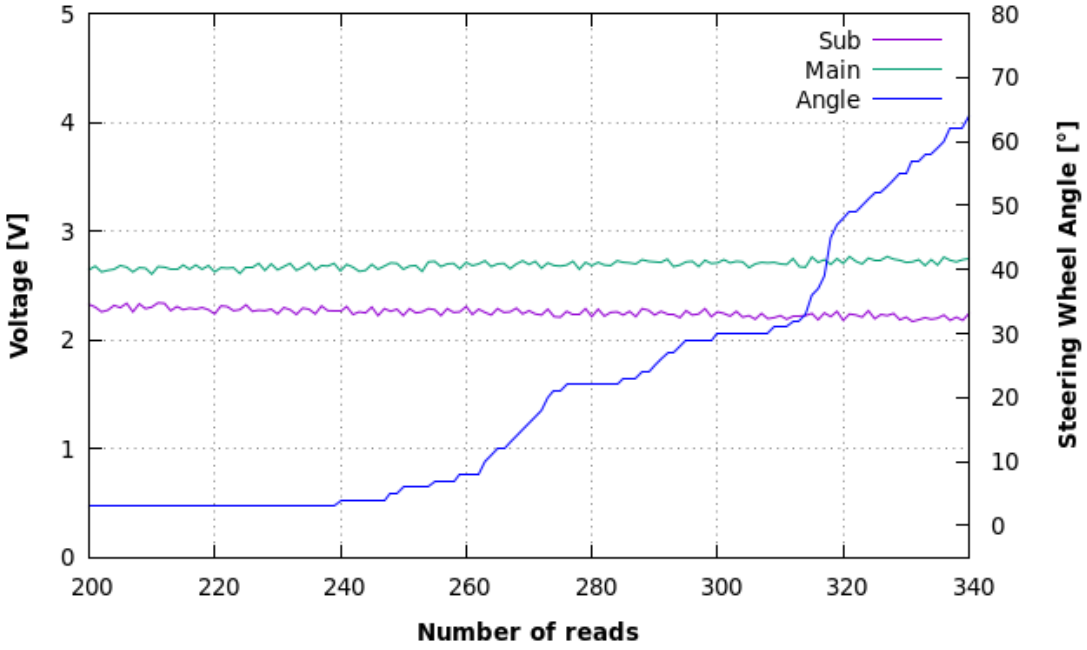


Figure 7.8: Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU.

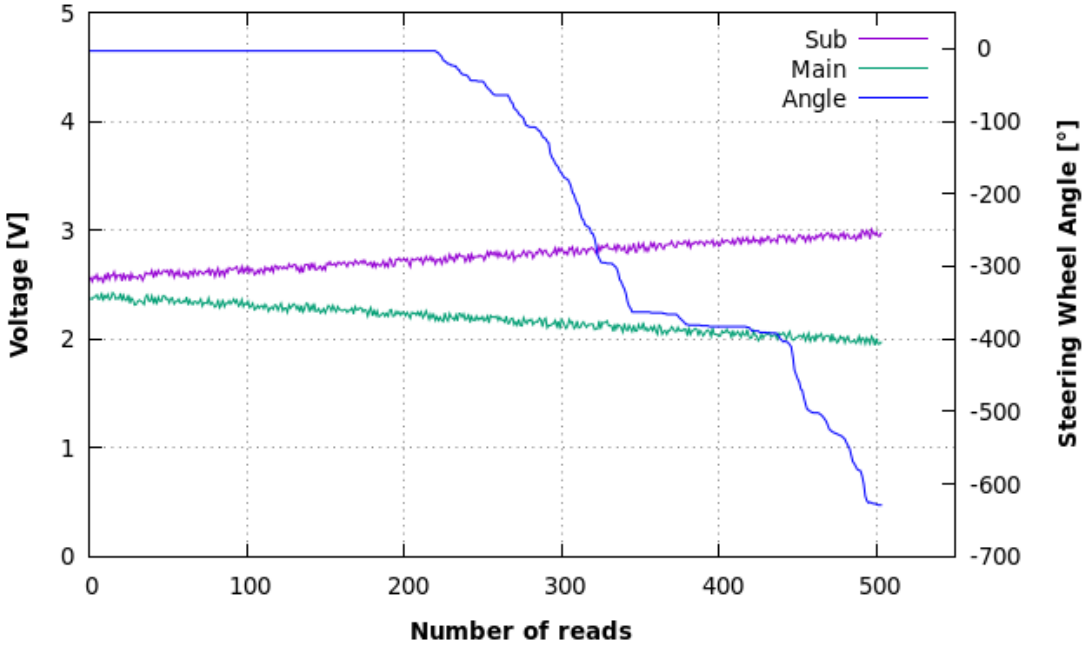


Figure 7.9: Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU.

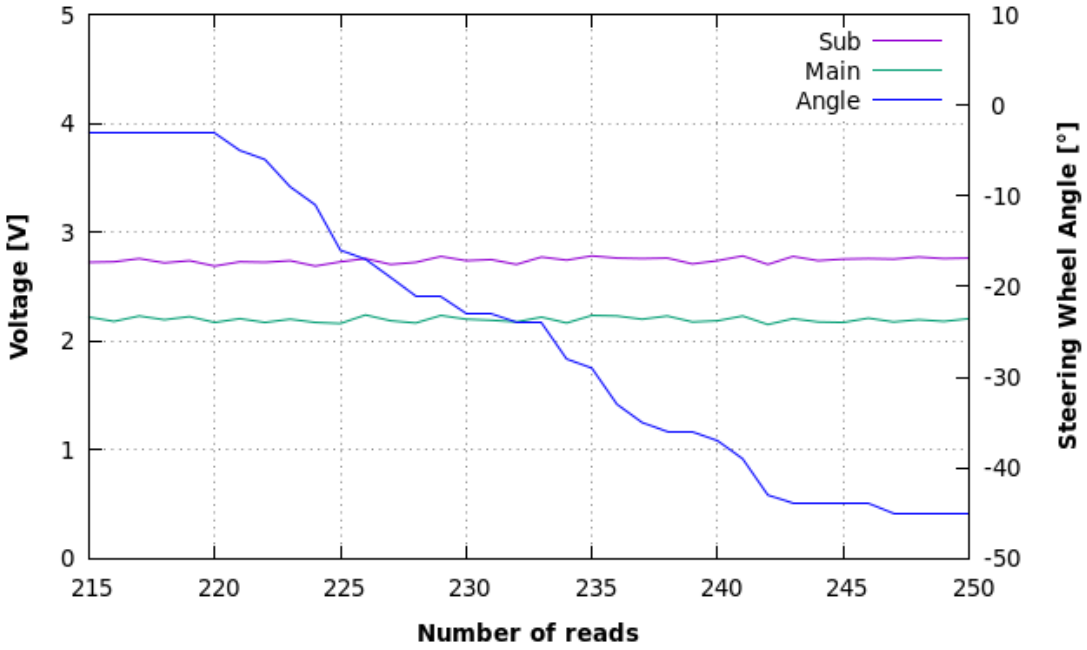


Figure 7.10: Steering wheel position of the car moving at cruise speed with the corresponding voltage values sent to the EPS-ECU.

when the vehicle is stationary, the steering wheel angle varies at different speeds in the range of 0° to 550°;

- in the graphs, where the range of angles from 0° to 70° (Figures 7.2 and 7.8) are highlighted, it can be seen that in the case of the experiment made in motion, the steering wheel rotation is smoother, which allows to control the direction of the vehicle with greater precision;
- with the vehicle in motion, the steering wheel controller is more sensitive to the variation of the electrical signals. Table 7.1 shows values taken from the same tests, which indicates that the EPS-ECU is programmed to act on the electric motor for lower voltage values when the vehicle is in motion.

Table 7.1: Analysis of the signals sent to the EPS-ECU and the respective steering wheel state with the vehicle stationary and in motion.

Steering Wheel State		**Electrical Signals at 0 km/h**	**Electrical Signals at 6 km/h**
Starts to rotate	Main Signal (V)	2.68	2.70
	Sub signal (V)	2.24	2.27
Reaches 60º	Main Signal (V)	2.96	2.76
	Sub Signal (V)	1.96	2.21
Reaches 400º	Main Signal (V)	3.12	2.84
	Sub Signal (V)	1.85	2.11

Table 7.1 also highlights a characteristic of the control system of the car in motion that is demonstrated in Figure 7.8. The variation of the electrical signals voltage between the point where the steering wheel starts to rotate and when it reaches 60° is minimal. For instance, as presented in the table, when the steering wheel starts to move the main and sub signals are, respectively, 2.70 V and 2.27 V and when the steering wheel angle is equal to 60° the voltage values are 2.76 V and 2.21 V, which represents a variation of 0.6 V. The fact that a wide steering wheel angle range corresponds to such a small variation of voltage values is a challenge in this solution, since it is difficult to reach such a high degree of accuracy in the voltages.

Through the observations made above, it can be concluded that the EPS-ECU changes the output to the electric motor taking into consideration the speed of the vehicle. In order to better understand this effect, an experiment was made where constant electrical signals are sent to the terminals of the EPS-ECU with the car in stationary state. The voltage values were selected to cause a slight rotation of the steering wheel to the left. Then, the speed of the vehicle is gradually increased, which causes an increase in the rotation angle of the steering wheel as soon as the speed becomes different from zero. The speed of the vehicle continued to be increased up to 13 km/h, however the state of the steering wheel did not change, which shows that the system reacts differently in situations of stationary state and with speed and does not take into consideration the speed value itself, at least at the speeds where the test was performed. Since the speed range from 0 to 15 km/h will be the most common in autonomous driving tests in the ATLASCAR2, for the time being, it is a valid conclusion to consider the steering wheel response the same at all speeds different from 0 km/h. This experiment is represented in the graph of Figure 7.11 and in a video available on: `https://youtu.be/J_h_DQJPj8w`.

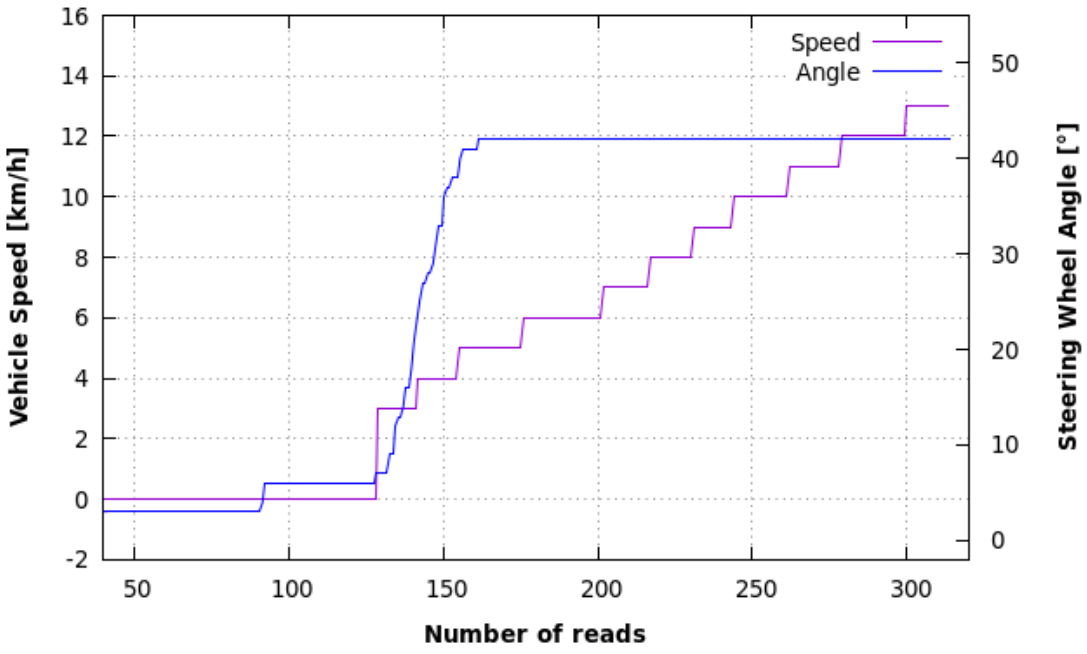


Figure 7.11: Differences observed in the steering wheel with the vehicle in stationary state and at different speeds.

Despite the conclusions already made, the tests presented above still do not clarify the full logic that the EPS-ECU uses to control the electric motor based on the main and sub signals. In order to clarify the control method used, an experiment was made in which constant voltage values are sent to the EPS-ECU and the steering wheel is manually positioned at different angles to observe the response of the steering wheel. A similar experiment is demonstrated in the video available on `https://youtu.be/sdePY5XrwRM`. Table 7.2 shows that, the EPS-ECU uses the electrical signals to define a displacement in the steering wheel angle. However, as expected in a controller of this nature, the displacement varies in an, apparently, non-logic way. For instance, the maximum and minimum displacements caused by the main signal 2.75 V and sub signal 2.25 V are, respectively, 57° and 25°. In the case of main signal equal to 2.76 V and sub signal 2.24 V the maximum and minimum displacements are 102° and 41°, respectively. This inconsistency of the displacement indicate that control of the steering wheel should be based on an increment/decrement logic, rather than directly position the steering wheel in the desired angle.

Table 7.2: Response of the steering system to the same electrical signals at different positions.

Main Signal [V]	Sub Signal [V]	Initial Steering Wheel Angle (°)	Final Steering Wheel Angle (°)	Steering Wheel Displacement (°)
2.75	2.25	0	31	31
		-16	37	53
		-33	2	35
		-71	-14	57
		-85	-36	49
		-176	-139	37
		-240	-199	41
		3	28	25
		18	48	30
		31	78	47
		63	110	47
		117	167	50
		209	249	40
2.76	2.24	0	73	73
		-21	51	72
		-60	19	79
		-132	-51	101
		-179	-92	87
		-281	-186	95
		-312	-210	102
		12	66	54
		23	90	67
		59	100	41
		101	155	54
		148	221	73
		214	284	70

Direction Control using a Potentiometer

After understanding the response of the steering system to certain voltage values of the electrical signals sent to the EPS-ECU terminals, it is possible to use a potentiometer to control the steering wheel of the vehicle. Since the control system requires two different voltage levels, the output from the potentiometer is processed, using the Arduino Uno, in two signals, according to the characteristics of the main and sub signals mentioned above. Thus, an Arduino IDE code was developed that uses the potentiometer signal to create two analog outputs that, after being interpreted by the EPS-ECU, produce the desired effect on the steering wheel of the vehicle. This code is in the Listing A.2 in Appendix A.

As seen above, the steering wheel angle range of 0° to 70° corresponds to an very low increase/decrease of the signals voltage values, so, it is difficult to have a proper control in that range using the potentiometer, because the lower maximum input on the potentiometer is 3V, in order to be able to rotate the steering wheel completely. However, as demonstrated in the video available on `https://www.youtube.com/watch?v=qr3qgnYNbDY`, this method can control the direction of the vehicle to perform approximate trajectories.

Direction Control through CAN

As presented on Chapter 6, the method implemented to send the instructions to the steering wheel controller uses the CAN bus of the vehicle. Thus, in this section, an experiment is described where the desired steering wheel angle is sent to the bus and then interpreted by the Arduino UNO, which is pre-programmed to compare the current and desired steering wheel angles and, based on this, change the analog output of the main and sub signals.

Since this experiment is for demonstration, a simple incremental controller is used, that is, after evaluating the desired steering wheel rotation, the analog signals are incremented/decremented, such as in the experiments of Figures 7.7 and 7.9, to rotate the steering wheel. When the desired steering wheel angle is reached (a margin of error of 5°is considered), the electrical signals are kept at 2.5 V in order to maintain the position. The Arduino IDE code used in this test is in Listing A.3 of the Appendix A.

The graph of Figure 7.12 is a representation of this experiment for the angle of 40° to the left. As can be seen, the position of the steering wheel established at 42°, which indicates that, in the future, this method can be used to effectively control the steering wheel.

7.2 Speed Remote Control

Similar to the control of the vehicle direction presented in last section, the control of the speed of the vehicle is done by emulating the sensor signals to the ECU that controls the system, as described in Chapter 5. At the time of writing, the solution available to change the speed of the vehicle just includes the possibility of accelerating the vehicle and uses regenerative braking to slow it down.

The tests performed replicate the previously analyzed APS signals, that is, a main signal between 1 V and 4.5 V and a sub signal with half the voltage of the main one.

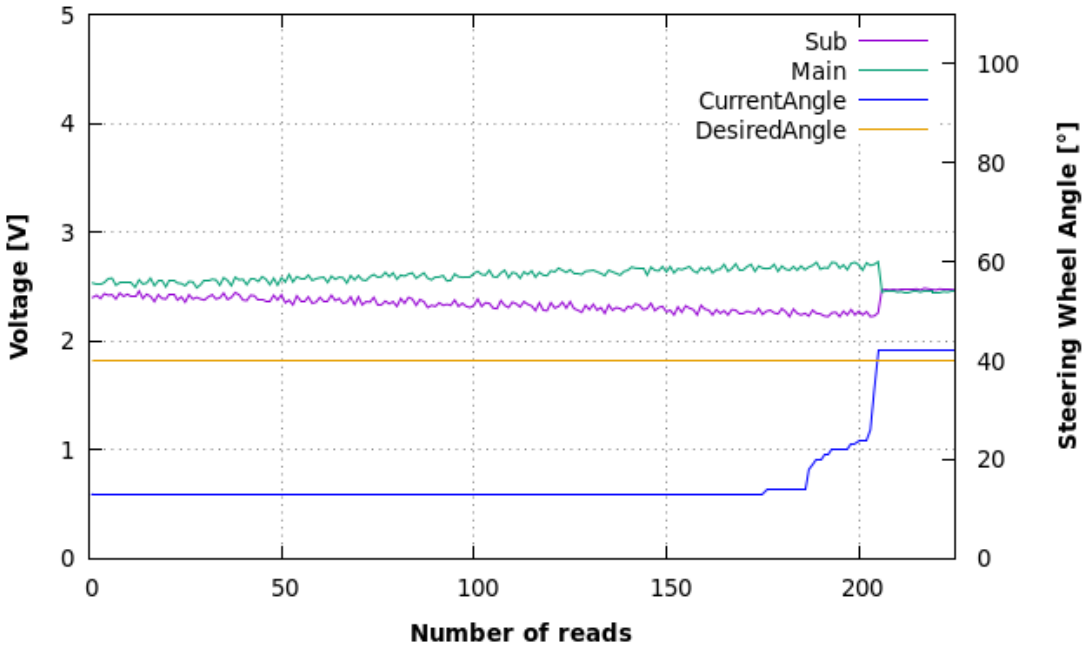


Figure 7.12: Controlling the steering wheel angle with CAN communication.

The graph of Figure 7.13 shows an experiment where the vehicle is moving at the cruise speed of 6 km/h and the main and sub signals are increased to 2 V and 1 V, respectively. As expected, that causes an increase in the speed of the vehicle. After the vehicle has reached a speed of 15 km/h, the voltage values were dropped to the normal conditions of 1 V and 0.5 V and, consequently, the vehicle starts to slow down. A similar experiment can be done with lower voltage values, which represents less acceleration and, obviously, a slower increase of the speed, as can be seen in the graph of Figure 7.14.

Also, it is possible to control the accelerator pedal in a similar way to the one used by the human operator when accelerating a vehicle, that is, gradually increase the acceleration. In the graph of Figure 7.15 can be seen that this method produces an exponential increase of the speed of the vehicle.

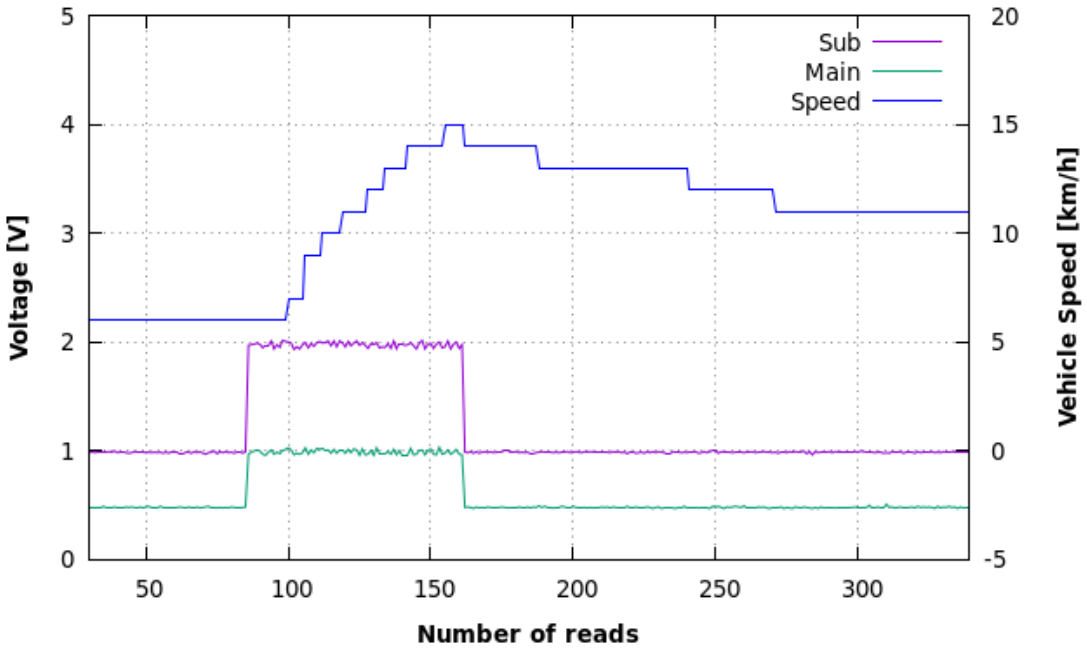


Figure 7.13: Speed of the vehicle with the respective voltage values sent to the EV-ECU.

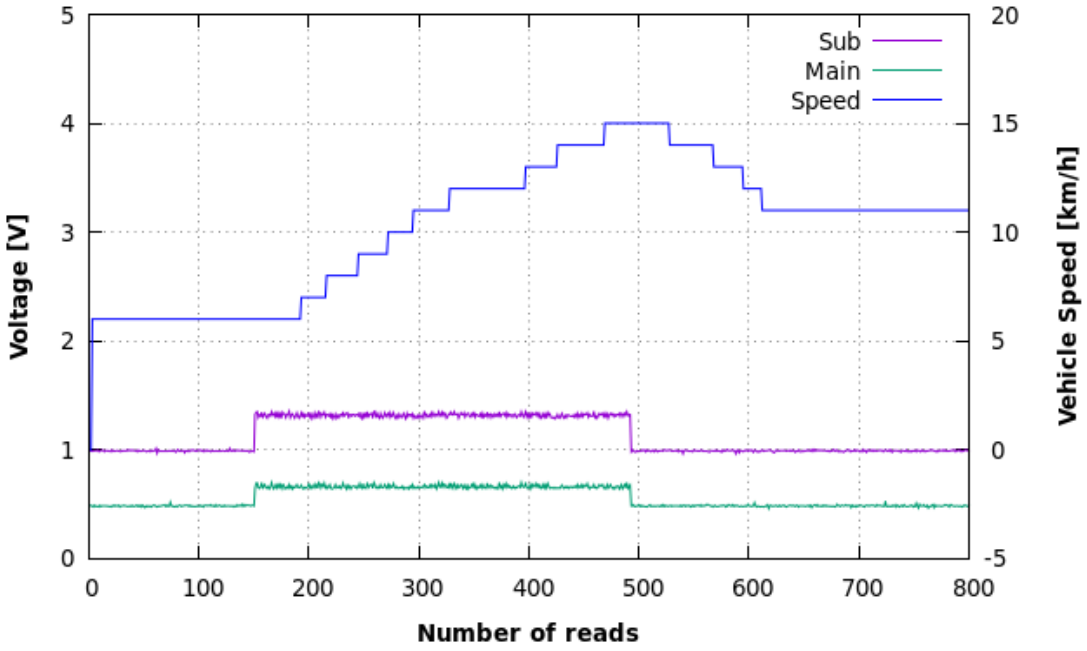


Figure 7.14: Speed of the vehicle with the respective voltage values sent to the EV-ECU.

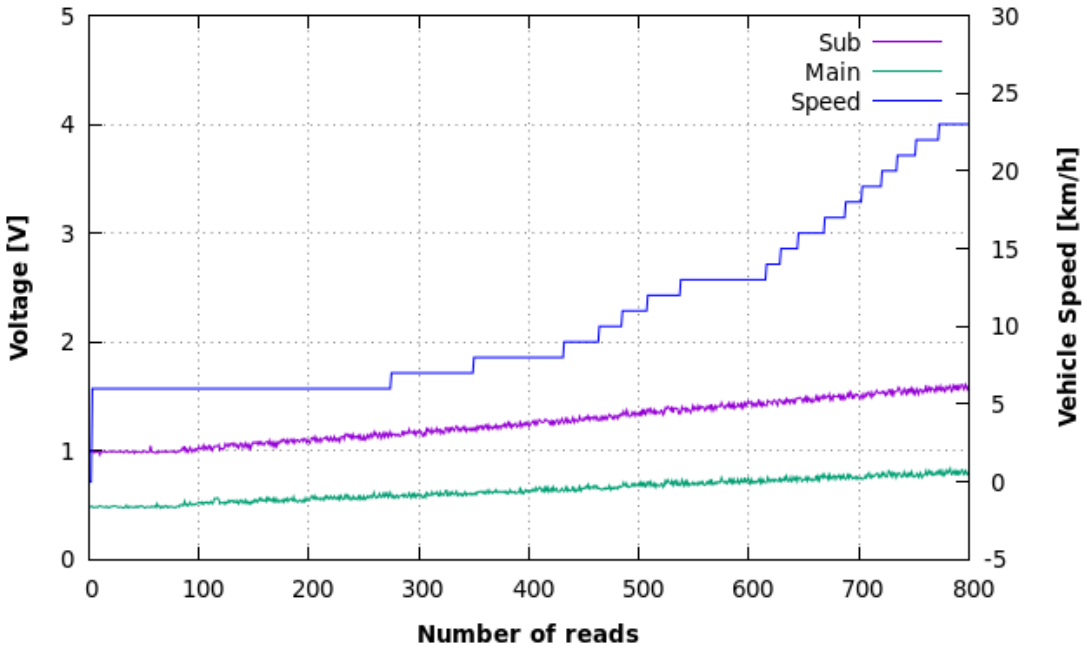


Figure 7.15: Speed of the vehicle with the respective voltage values sent to the EV-ECU.

Intentionally blank page.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

With the objective of enabling the ATLASCAR2 to be used as a platform in AD tests and applications, the work presented in this document aimed to develop solutions for the remote control of the Mitsubishi i-MiEV steering and propulsion systems and monitor the entire vehicle.

Regarding the monitoring part, this work presents a solution that publishes the vehicle status in the global system in a robust and effective way, since it uses the vehicle CAN bus, which is practically error free. The current status of the vehicle and the state of its main parameters is not only crucial for the perception and navigation algorithms, but also for the remote control solutions. In fact, the approach that directly connects the controllers to all CAN messages flowing on the vehicle bus provides varied possibilities in the ATLASCAR2 autonomous and control motion. An example of that is the solution created to send instructions to the controllers, that uses the CAN bus to send dedicated frames that can be processed and interpreted by the controllers.

In what concerns to the remote control of the direction of the vehicle, based on the tests and results presented in last chapter, it can be concluded that the steering system can be effectively controlled by sending signals to the EPS-ECU. The results indicate that the approach taken in this work can be used to develop an effective controller for the steering wheel, despite the non-linearity between the electrical signals received by the EPS-ECU and the position of the steering wheel.

In addition, this research aimed to identify a solution to act on the brake system of the car. Based on the explanation regarding the braking logic of the Mitsubishi i-MiEV, presented in Chapter 5, it can be concluded that this system is not prepared to be remotely controlled without having pressure being applied in the brake pedal. Thus, this dissertation proposes a solution that effectively increases the speed of the vehicle replicating the signals from the accelerator pedal sensor and uses regenerative braking to slow the vehicle, which, in some situations, can produce enough deceleration. While the inability of directly control the brakes limits the ability to effectively control the speed of the vehicle at the time being, this research clarifies the operation logic of the braking system, which provides a new insight into potential solutions - some suggestions are made in the next section.

8.2 Future Work

Being the first work on the ATLASCAR2 autonomous mobility, this dissertation intended to discover and demonstrate the existing possibilities to perform the remote control of the direction and speed of the vehicle. The findings documented in this thesis and the infrastructure created in this work facilitates the development and implementation of new techniques in future works. Appendix C describes the procedure to be performed to test some of the implemented solutions.

Based on the conclusions regarding the steering wheel control, future work can proceed with the development of a controller for the steering wheel. Nevertheless, future studies could address the possibility of controlling the steering wheel by acting directly on the electric motor that supports the steering operation, which offers a simpler way to control the direction of the vehicle.

Concerning the speed control, studies have shown that this control requires an external actuator applying pressure on the brake pedal, which is not the type of solution the ATLASCAR2 project is looking for, so, it should be considered to adapt the braking system of the vehicle to one that enables the creation of similar solution to the ones presented in this thesis. For instance, a possibility is to adapt the EV-ECU to open the vacuum valve that acts on the brake system when braking operations are required.

Also, in order to reduce the need to act on the braking system of the vehicle, future studies could consider change the vehicle's speed by manipulating the CAN message that the EV-ECU sends to the EMCU with the torque command, since, as mentioned in Chapter 5, the EV-ECU uses CAN communication to send the torque command to the EMCU. This method has other complications, such as CAN message confidentiality issues and conflict problems between messages, since the target ECU will receive the messages injected by the controller and the messages from the original ECU.

Appendix A

Arduino IDE Code

```
/*
  Receives and processes CAN messages to obtain the desired and current
    steering wheel angle from the global system.
 */

#include <SPI.h>
#include "mcp_can.h"

/*SAMD core*/
#ifdef ARDUINO_SAMD_VARIANT_COMPLIANCE
    #define SERIAL SerialUSB
#else
    #define SERIAL Serial
#endif

// CAN Communication
const int SPI_CS_PIN = 9;
MCP_CAN CAN(SPI_CS_PIN);

int dirD; //desired angle

void setup() {
  SERIAL.begin(115200);

  while (CAN_OK != CAN.begin(CAN_500KBPS)) {            // init can bus :
     baudrate = 500k
      SERIAL.println("CAN BUS Shield init fail");
      SERIAL.println(" Init CAN BUS Shield again");
      delay(100);
  }
  SERIAL.println("CAN BUS Shield init ok!");
}

void loop() {
  unsigned char len = 0;
  unsigned char buf[8];

  if (CAN_MSGAVAIL == CAN.checkReceive()) {         // check if data
    coming
    CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf:
    data buf
```

```
    unsigned long canId = CAN.getCanId();

    //message regarding the steering operation
    if (canId == 0x500)
        {
          //print the message
          SERIAL.println("-----------------------------");
          SERIAL.print("Get data from ID: 0x");
          SERIAL.println(canId, HEX);

          for (int i = 0; i < len; i++) { // print the data
            SERIAL.print(buf[i], HEX);
            SERIAL.print("\t");
            }
          //obtain the desired steering wheel angle
          dirD = buf[0];
          if (buf[1] == 0x01)
              {dirD = -dirD;}
        }
  }
}
```

Listing A.1: Arduino code used to receive the desired and current steering wheel angle through CAN communication.

```
/*
  Receives a signals from the potenciometer and, based on that value,
    creates two analog outputs that the produce the desired effect on the
    vehicle steering wheel
 */


int outPin=5;
int outPin2=6;
float pot1 = 255/2;
float pot2 = 255/2;

void setup() {
  pinMode(A0,INPUT);
  pinMode(outPin,OUTPUT);
  pinMode(outPin2,OUTPUT);
}

void loop() {
  poten1 = analogRead(A4)/14+102;
  poten2 = 255/2+(255/2-poten1);

  analogWrite(outPin,pot1);
  analogWrite(outPin2,pot2);

  delay(250);
}
```

Listing A.2: Arduino code used to control the steering wheel with a potenciometer.

```
/*
  Receives and processes CAN messages to obtain the desired and current
    steering wheel angle and, based on those values, change the analog
    output.
 */

#include <SPI.h>
#include "mcp_can.h"

/*SAMD core*/
#ifdef ARDUINO_SAMD_VARIANT_COMPLIANCE
    #define SERIAL SerialUSB
#else
    #define SERIAL Serial
#endif

// CAN Communication
const int SPI_CS_PIN = 9;
MCP_CAN CAN(SPI_CS_PIN);

int dirD; //desired angle
int dirA; //current angle
int delta;

int outPin=5;
int outPin2=6;
float pot1=255/2;
float pot2=255/2;

void setup() {
  pinMode(outPin,OUTPUT);
  pinMode(outPin2,OUTPUT);

  SERIAL.begin(115200);

  while (CAN_OK != CAN.begin(CAN_500KBPS)) {            // init can bus :
     baudrate = 500k
      SERIAL.println("CAN BUS Shield init fail");
      SERIAL.println(" Init CAN BUS Shield again");
      delay(100);
  }
  SERIAL.println("CAN BUS Shield init ok!");
}

void loop() {
  unsigned char len = 0;
  unsigned char buf[8];

  if (CAN_MSGAVAIL == CAN.checkReceive()) {         // check if data
    coming
    CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf:
    data buf

    unsigned long canId = CAN.getCanId();

    //desired steering wheel angle
    if (canId == 0x500)
```

```
    {
      dirD = buf[0];
      if (buf[1] == 0x01)
          {dirD = -dirD;}
    }

    //current steering wheel angle
    if (canId == 0x236)
    {
        dirA = (buf[0]*256+buf[1]-4096)/2;
    }
    delta = dirD-dirA;
  }

  //change output signals based on the angle values
  if (delta > 0) //rotate to the left
  {pot1=pot1+0.5;
  pot2=pot2-0.5;}
  elseif (delta < 0) //rotate to the right
  {pot1=pot1-0.5;
  pot2=pot2+0.5;}

  if (|delta| < 5)
  {pot1=255/2; //maintain position
  pot2=255/2;}

  analogWrite(outPin,pot1); //main signal
  analogWrite(outPin2,pot2); //sub signal

  delay(250);
}
```

Listing A.3: Arduino code used to control the steering wheel position using CAN communication.

Intentionally blank page.

Appendix B

Steering wheel response to different types of surfaces

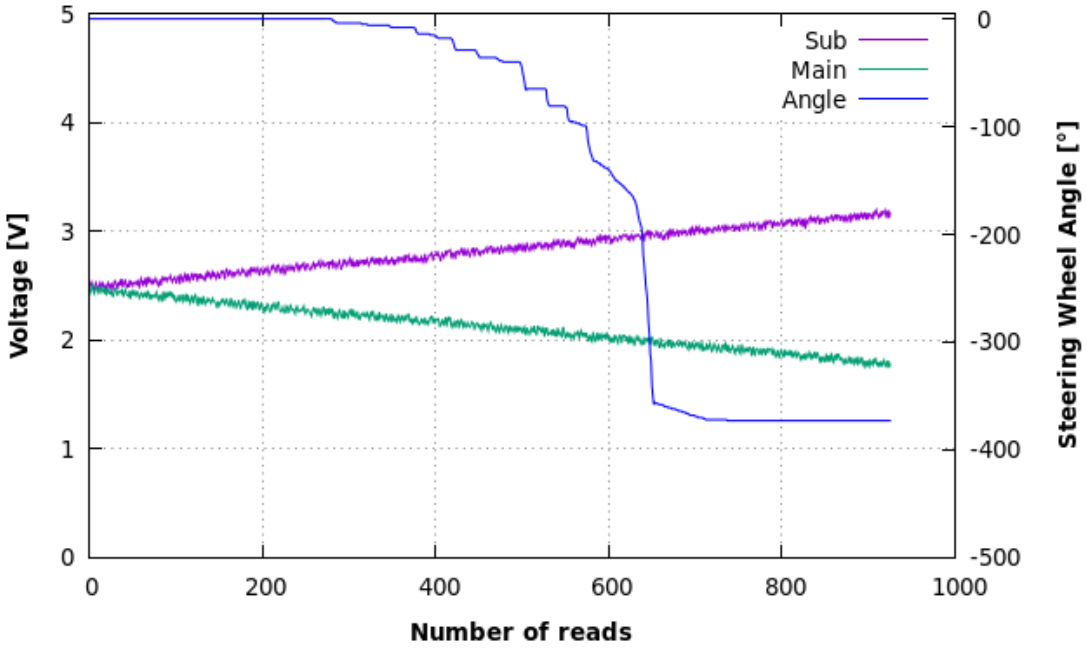


Figure B.1: Steering wheel position of the car in stationary state over a surface of road tar with the corresponding voltage values sent to the EPS-ECU.

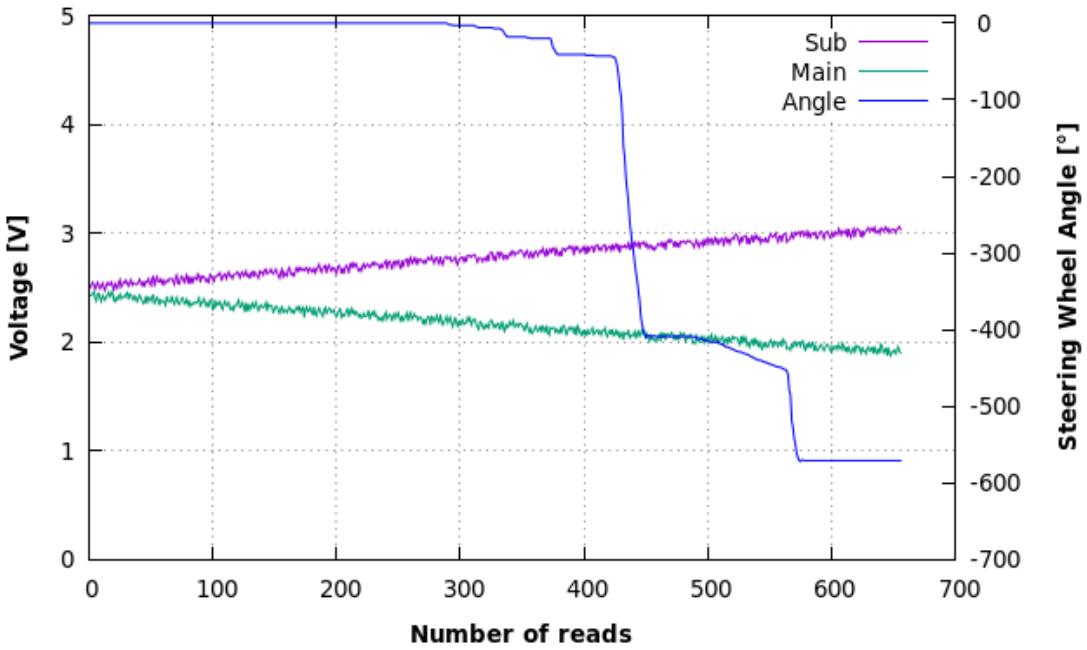


Figure B.2: Steering wheel position of the car in stationary state over a surface of sand with the corresponding voltage values sent to the EPS-ECU.

Appendix C

Instruction Manual

C.1 Monitor the vehicle status

1. Plug CANalyze to the OBD-II port of the vehicle;
2. Install the SocketCAN utilities:

   ```
   $ sudo apt-get install can-utils
   ```

3. Plug CAN to the PC and set up the device doing:

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

4. Run the ROS node responsible to the monitoring of the system:

   ```
   $ rosrun atlascar2 canReceiveAndUpdateStatus
   ```

5. Subscribe to the topic that contains the information about the vehicle: `/NominalData`.

C.2 Control the Steering Wheel and Accelerator Pedal

1. Plug the Arduino UNO with the CAN Bus Shield and the CANlyze to the OBD-II port of the vehicle;
2. Use the circuit presented in Chapter 6 to connect the Arduino UNO, torque sensor and EPS-ECU;
3. Upload the Arduino IDE code with `receiveCANsendV` available on `https://github.com/lardemua/ATLASCAR2RemoteControl` in the Arduino UNO;

Intentionally blank page.

References

[1] Karl Koscher et al. *Experimental Security Analysis of a Modern Automobile*. University of Washington, University of California San Diego, 2010. URL: http://www.autosec.org/pubs/cars-oakland2010.pdf.

[2] *ATLAS project*. URL: http://atlas.web.ua.pt/ (visited on 05/07/2020).

[3] José Pereira. *Quadro Elétrico ATLASCAR-2*. Projeto Engenharia de Automação Industrial. Universidade de Aveiro, 2017. URL: http://lars.mec.ua.pt/public/LAR%5C%20Projects/SystemDevelopment/2017_JosePereira/PEA_Relatorio_QE_71985.pdf.

[4] *SAE International Releases Updated Visual Chart for Its "Levels of Driving Automation" Standard for Self-Driving Vehicles*. URL: https://www.sae.org/news/press-room/2018/12/sae-international-releases-updated-visual-chart-for-its-%5C%E2%5C%80%5C%9Clevels-of-driving-automation%5C%E2%5C%80%5C%9D-standard-for-self-driving-vehicles (visited on 05/07/2020).

[5] *The 6 Levels of Vehicle Autonomy Explained — Synopsys Automotive*. URL: https://www.synopsys.com/automotive/autonomous-driving-levels.html (visited on 05/07/2020).

[6] *ANSR*. URL: http://www.ansr.pt/Estatisticas/RelatoriosDeSinistralidade/Pages/default.aspx (visited on 05/07/2020).

[7] *Think You're In Your Car More? You're Right. Americans Spend 70 Billion Hours Behind the Wheel*. AAA NewsRoom. Feb. 27, 2019. URL: https://newsroom.aaa.com/2019/02/think-youre-in-your-car-more-youre-right-americans-spend-70-billion-hours-behind-the-wheel/ (visited on 05/07/2020).

[8] *The Automobile and the Environment in American History by Martin V. Melosi*. URL: http://www.autolife.umd.umich.edu/Environment/E_Overview/E_Overview.htm (visited on 06/01/2020).

[9] Posted 01 Feb 2009 05:00 GMT. *This Car Runs on Code - IEEE Spectrum*. IEEE Spectrum: Technology, Engineering, and Science News. URL: https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code (visited on 06/01/2020).

[10] *How to hack a car — a quick crash-course*. freeCodeCamp.org. June 21, 2017. URL: https://www.freecodecamp.org/news/hacking-cars-a-guide-tutorial-on-how-to-hack-a-car-5eafcfbbb7ec/ (visited on 02/26/2020).

[11] Craig Smith. *The Car hacker's handbook : a guide for the penetration tester*. San Francisco: No Starch Press, Inc., 2016.

[12] *Aptiv — Autonomous Mobility*. Aptiv. URL: https://www.aptiv.com/solutions/autonomous-mobility (visited on 05/12/2020).

[13] Deutsche Welle (www.dw.com). *Singapore public to test self-driving taxis — DW — 23.09.2016*. DW.COM. URL: https://www.dw.com/en/singapore-public-to-test-self-driving-taxis/a-19569721 (visited on 06/01/2020).

[14] *AUTO C-ITS*. URL: https://www.autocits.eu/ (visited on 05/12/2020).

[15] *Iseauto*. URL: https://iseauto.taltech.ee/ (visited on 05/13/2020).

[16] Raivo Sell et al. "Autonomous Last Mile Shuttle ISEAUTO for Education and Research". In: 10 (Feb. 2020), p. 13. DOI: 10.4018/IJAIML.2020010102.

[17] *Home*. Waymo. URL: https://waymo.com/ (visited on 05/13/2020).

[18] *Autopilot*. URL: https://www.tesla.com/autopilot (visited on 05/13/2020).

[19] *Car Hacking: The definitive source*. URL: http://illmatics.com/carhacking.html (visited on 06/02/2020).

[20] Charlie Miller and Chris Valasek. *Adventures in Automotive Networks and Control Units*. 2014. URL: http://illmatics.com/carhacking.html.

[21] *CAESS - Publications*. URL: http://www.autosec.org/publications.html (visited on 06/01/2020).

[22] Karl Koscher et al. *Comprehensive Experimental Analyses of Automotive Attack Surfaces*. University of Washington, University of California San Diego, 2011. URL: http://www.autosec.org/pubs/cars-usenixsec2011.pdf.

[23] *CANalyze*. URL: https://kkuchera.github.io/canalyze/ (visited on 05/19/2020).

[24] *Mitsubishi i-MiEV Service Manual, Technical Information Manual & Body Repair Manual*. 2013.

[25] Parth Chhabra. *Getting Started With ROS(Robot Operation System)*. Medium. Mar. 6, 2019. URL: https://medium.com/@parthc21/getting-started-with-ros-robot-operation-system-96a6590ea683 (visited on 05/19/2020).

[26] *linux-can/can-utils*. original-date: 2015-03-04T18:34:07Z. May 19, 2020. URL: https://github.com/linux-can/can-utils (visited on 05/19/2020).

[27] *CAN communication tutorial, using simulated CAN bus — sgframework 0.2.3 documentation*. URL: https://sgframework.readthedocs.io/en/latest/cantutorial.html (visited on 05/19/2020).

[28] C. S. S. Electronics. *CAN Bus Explained - A Simple Intro (2020)*. CSS Electronics. URL: https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en (visited on 06/01/2020).

[29] Luís Cristóvão. *Interface OBD para o AtlasCar2 e Monitoriza c ao doseu Estado*. Projeto Engenharia de Automação Industrial. Universidade de Aveiro, 2018. URL: http://lars.mec.ua.pt/public/LAR%5C%20Projects/HardwareInterfaces/2018_LuisCristovao/Relatorio_Apresentacao/PEA_80886.pdf.

[30] Priit Laes. *plaes/i-miev-obd2*. Feb. 17, 2020. URL: https://github.com/plaes/i-miev-obd2 (visited on 02/26/2020).

[31] *Decyphering iMiEV and iON CAR-CAN message data - Mitsubishi I-Miev Forum*. URL: http://myimiev.com/forum/viewtopic.php?f=25&t=763&hilit=send+messages+to+can/ (visited on 05/31/2020).

[32] *Arduino Uno Rev3 — Arduino Official Store*. URL: https://store.arduino.cc/arduino-uno-rev3 (visited on 06/08/2020).

[33] *Low Pass Filter - Passive RC Filter Tutorial*. Basic Electronics Tutorials. Aug. 14, 2013. URL: https://www.electronics-tutorials.ws/filter/filter_2.html (visited on 06/05/2020).

[34] Gianfranco Pistoia. *Industrial Applications of Batteries: From Cars to Aerospace and Energy Storage*. Elsevier Science, 2007. 792 pp. ISBN: 978-0-444-52160-6.

[35] *CAN-BUS Shield V2*. URL: https://www.antratek.com/can-bus-shield-v2 (visited on 06/09/2020).

[36] *CAN Bus Or SAE J1939 Development Kit With Arduino Uno*. Copperhill. URL: https://copperhilltech.com/can-bus-or-sae-j1939-development-kit-with-arduino-uno/ (visited on 06/09/2020).