Pedro Filipe Pinto Pinheiro

Sistema Automático para Gestão da Caixa de Velocidades do AtlasCar

Automatic Management System for the AtlasCar Gearbox

Pedro Filipe Pinto Pinheiro

Sistema Automático para Gestão da Caixa de Velocidades do AtlasCar
Automatic Management System for the AtlasCar Gearbox

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

Para ti, Xaninha :)

O júri / The jury

Presidente / President	**Prof. Doutor Jorge Augusto Fernandes Ferreira** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor António Manuel Ferreira Mendes Lopes** Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor Vítor Manuel Ferreira dos Santos Professor Associado da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Ao Professor Doutor Vítor Santos, pelo acompanhamento constante, pela motivação, por estimular o interesse pela robótica e pela programação e pela luta constante para que eu soubesse sempre qual o caminho a seguir.
Ao Miguel Oliveira e ao Jorge Almeida, por tantas vezes partilharem o conhecimento, e apontarem a direção mais certa a seguir, em alturas cruciais do trabalho.
Ao Doutor Ricardo Pascoal, por ter partilhado a sua vastíssima experiência, transversal a inúmeras áreas técnicas e científicas, pela sua extraordinária paciência para ensinar e pelas achegas que foi dando na área da eletrónica.
Aos meus colegas e amigos do LAR, por manterem a boa disposição, pela companhia nas noitadas de trabalho e por me aturarem nos momentos de pessimismo e de mau humor.
Aos meus pais, avós e irmã, pelo apoio exemplar ao longo de toda a minha vida. Sem eles nada teria sido possível.
Resta-me mencionar os meus enormes Amigos Ana Matos e Dilas Fortes, a quem terei de agradecer pessoalmente e com um grande abraço. Ao longo destes cinco anos ajudaram-me tanto, de tantas formas e em tantos momentos, que sinto que tentar fazê-lo aqui, por escrito e numa só frase, não seria suficiente.

Palavras-chave Controlador Lógico Programável; Caixa Automática; *Arduino*; Motor DC; Ponte H; Simulador *Hardware-in-the-Loop*

Resumo

O veículo AtlasCar é um protótipo desenvolvido pelo Laboratório de Automação e Robótica do Departamento de Engenharia Mecânica da Universidade de Aveiro, e tem como principais objetivos o estudo de sistemas de segurança ativos e passivos, técnicas de apoio à condução e soluções para a condução autónoma.

Até ao momento, uma das maiores limitações a nível da atuação de mecanismos essenciais para a condução autónoma, verificada no AtlasCar, era a ausência de um sistema que permitisse o controlo da caixa de velocidades. Embora o comutador de caixa estivesse já projetado e construído, não possuía nenhum sistema de controlo que permitisse a sua utilização. Com este trabalho pretende-se executar todo o projeto, a construção e a programação de baixo nível de um controlador robusto e eficaz para desempenhar o seu papel no âmbito do projeto AtlasCar. Um protocolo de comunicação fiável entre o *firmware* do presente mecanismo e o *software* do AtlasCar será também implementado, devido á grande responsabilidade do dispositivo em questão, de forma a permitir uma condução segura.

Outros softwares acessórios á correta utilização deste mecanismo no âmbito do projecto AtlasCar, como um software de calibração e um nodo de ROS para comunicação com o mesmo serão apresentados.

É também objetivo deste trabalho a realização de alguns testes de bancada, necessários de forma a comprovar o funcionamento correto quer da programação, quer do protocolo de comunicação criado, recorrendo a um simulador com *Hardware-in-the-Loop* simplificado, programado em Matlab.

Keywords Programmable Logic Controller; Gearbox; *Arduino*; DC Motor; H-bridge; Hardware-in-the-Loop Simulator

Abstract

The AtlasCar vehicle is a prototype developed by the Laboratory of Automation and Robotics at the Department of Mechanical Engineering at Aveiro University with the purpose of studying active and passive safety systems, assisted driving techniques and new solutions for autonomous driving.
Until now, one of the major faults in what concerns to the actuation of the AtlasCar vehicle's main driving systems was the absence of a mechanism that would allow the control over the AtlasCar gearbox.
Although a mechanism had already been built for this purpose, it had no control system that would allow it to be automatically actuated. The purpose of this work is to present an electronic project for a robust and effective controller to this AtlasCar's gear selector mechanism. The controller's construction and its low-level programming is also executed. A reliable communication protocol between the firmware of the mechanism, and the AtlasCar software is also implemented, due to the high responsibility task to be performed by this actuator in the driving process.
Other accessory software, like a calibrator for the gear selector mechanism and a ROS Node to perform the communication between the AtlasCar control PC and the mechanism, are also presented.
It is also within the scope of this work the execution of several laboratory tests, in order to determine the robustness of both the programming and the communication protocol. These tests will be executed with a simplified Hardware-in-the-Loop simulator, written using Matlab.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The ATLAS Project

The Atlas project has been developed at the Laboratory of Automation and Robotics at the Department of Mechanical Engineering of the University of Aveiro [1]. The project started in 2002/2003, and its primary mission is to develop, study and implement advanced sensing and active systems for further implementation in the automotive industry, or other similar platforms. After repeatedly achieving excellent results while participating in the Autonomous Driving Competition, taking place at the Portuguese Robotics Open, the Atlas team decided it was time to move on to a more ambitious and challenging environment: the real world scenario. In order to do so, a new platform was acquired and a brand new branch of the project was created: the AtlasCar.

The AtlasCar is a real-scale prototype vehicle used for research on Advanced Driver's Assistance Systems (ADAS), and consists of a modified and adapted 1998 Ford Escort Station Wagon, shown in Figure 1.1. The vehicle is equipped with several sensor systems that allow it to perceive the surrounding environment, as well as some of the driver's actions. The main sensors currently available in the car are the following:

- Stereo Head;
- Foveated Vision;
- 2D Laser Scanners;
- Custom 3D Laser Scanner;
- MEMS IMU;
- GPS Receiver;
- Pedal pressure Sensors.

The vehicle has also suffered a major set of interventions on a hardware level. One of the most important changes was the addition of a second DC generator, independent from the original AtlasCar alternator, but also propelled by the engine shaft. In order to power up the vehicle's new and more demanding electric circuitry, including all the computers, monitors, cameras, sensors and actuators, this DC generator is connected to an AC inverter that raises the voltage to 220/230 V. This inverter is connected to an

Figure 1.1: AtlasCar prototype vehicle.

Uninterruptible Power Supply (UPS) to ensure a stable and safe power supply to all the mentioned equipments. The UPS is the interface from where all equipments' power is derived. The car also includes two DC power rectifiers: a 12 V and a 24 V, used to feed the large variety of electronic and electromechanical equipment. The power management of each sensor or actuator is made by software using an industrial Mitsubishi Programmable Logic Controller (PLC). Another PLC, a Siemens SIMATIC S7-1200 depicted in Figure 2.2, is the responsible for the majority of the actuation systems, the low-level management of the security systems and the monitoring of several driving parameters.

1.2 The AtlasCar Actuators

As the AtlasCar project consists of an automatic driving prototype, there is the need to control as many driving parameters as possible. In order to achieve this main objective, AtlasCar uses some electronically controlled actuators on most of the main car systems. The systems currently being actuated are listed below:

- The Throttle;
- The Brake pedal;
- The Clutch pedal;
- The Handbrake;
- The Ignition;
- The Steering;
- The Lights.

A brief description of these actuator systems is presented from Subsection 1.2.1 to Subsection 1.2.6.

1.2.1 The Clutch and Brake Pedals

The AtlasCar clutch and brake pedals are actuated in a very similar way. These pedals are mechanically connected, through a steel wire attached to the pedals, to the respective actuator boxes, currently located below the driver's seat to allow easy access. These actuator boxes are actually adapted lock brake actuators from a *Renault Vel Satis*, whose circuitry was remade in order to allow the local control of the speed and sense of rotation of its DC motor [2]. The actuators simulate the pedal movements normally caused by a driver's foot pressing them, without interfering with the normal driving of the car. The control signal used to actuate the clutch and the brake is controlled using a digital communication protocol with the Siemens Programmable Logic Controller. The force that the driver exerts in each pedal can also be monitored, using variable resistance sensors, and the values are communicated directly to the PC.

The clutch actuator is particularly important for the future of this this work, as it presents itself as a fundamental element on the gear shifting process. The clutch allows the engine to be mechanically detached from the gearbox, which needs to be stopped in order to perform a smooth gear shifting process.

1.2.2 The Handbrake

The system used to drive the handbrake uses a chain and sprocket wheel attached to the handbrake itself. Another adapted *Renault Vel Satis* lock brake actuator box was used to drive this system. An electric linear actuator, depicted in Figure 1.2, pulls the handbrake blocking button when necessary. All the handbrake actions are controlled locally using a programmable integrated circuit. The control signal used in order to actuate this system is also controlled using a digital communication protocol with the Siemens PLC.

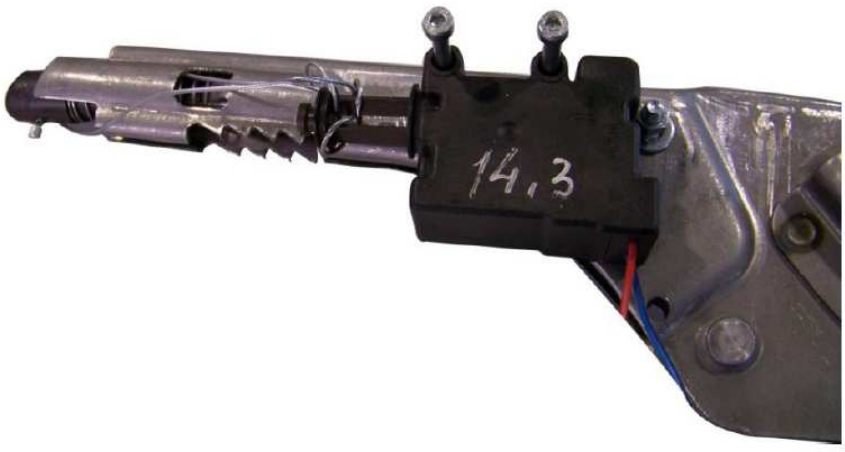


Figure 1.2: AtlasCar Handbrake mechanism.

1.2.3 The Throttle

The AtlasCar original vehicle, a 1998 Ford Escort Station Wagon, was not equipped with an electrically controlled throttle valve, and consisted on a common throttle, actuated

directly by a steel wire from the pedal to the throttle valve. This system was replaced by the electronic throttle depicted in Figure 1.3, which uses digital potentiometers and an H-bridge, controlled by an *Arduino*, in order to set a desired position to the valve. This system communicates directly with the control PC, via TCP/IP.

Figure 1.3: AtlasCar electronic throttle valve.

1.2.4 The Ignition

The AtlasCar current automatic ignition uses a two relay system, as depicted in Figure 1.4, because there are two main circuits on the original lock cylinder:

- A pair of Power Cables;
- A pair of cables for the Starter Motor;

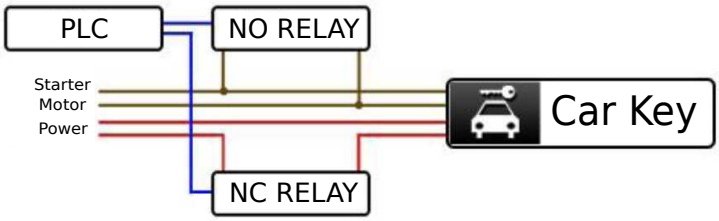


Figure 1.4: AtlasCar electrical ignition diagram.

In order to activate this system, a normally closed relay is used in series with the Power cables, and a normally open relay controls the state of the starter motor. This system allows the automatic control of the ignition, but the manual system remains fully operational. The key still needs to be inserted on the cylinder and rotated to the second position (*Power On*) for the system to start, because of the steering wheel blocking system.

1.2.5 The Steering

The steering actuator is an adapted electric power steering system, depicted in Figure 1.5. The original power steering system was hydraulic, and it is currently only aiding the new electric system. The electrical actuator is locally controlled by a Programmable Integrated Circuit, the position of the steering wheel is monitored using a multi-turn potentiometer and the communication with the PLC is made using analogue signals.

Figure 1.5: AtlasCar electrically controlled steering column.

1.2.6 The Lights

The AtlasCar lights are currently being both controlled and monitored. The lights control is achieved using the PLC digital outputs, which command a relay box, currently placed below the passenger's seat. The lights currently being actuated are listed below:

- Dipped Beam Headlamps;
- Main Beam Headlamps;
- Right Directional Indicator;
- Left Directional Indicator;
- Roof-mounted Yellow Beacon;

The lights' actuation system is connected in parallel with the vehicle's original light circuitry, so all the lights can still be normally used by a human driver. This fact is particularly useful if the automatic systems are turned off, or in the case that the car is only used to monitor the driver's behaviour, without interfering with it.

1.3 Objectives

The AtlasCar vehicle is not originally equipped with an automatic transmission system, which makes the implementation and testing of new autonomous driving algorithms very

difficult, or even impossible. One of this work's main objectives is to project, build and test extensively a new gearbox management system based on the control of two DC motors. It is also an objective of this project to prove that this controller is solid and robust enough to control the existent AtlasCar gear selector mechanism.

The other main objective of this work is to develop a reliable firmware solution to control the gear selector mechanism, which can automatically perform the trajectories required to obey any gear change order given by the software controlling the gearbox on the AtlasCar PC.

The communication process with the device will also be explored and tested, to ensure the proper operation required for such a vital element of the AtlasCar vehicle: the gearbox.

It is also in the scope of this work the development of a simple, yet highly configurable Hardware-in-the-Loop simulator, in order to perform laboratory tests on the gear selector mechanism controller.

1.4 Dissertation Structure

This dissertation consists of seven chapters.

This first Chapter is a brief introduction to the Atlas project. It presents some of the project's history, and its current developments stage. In this Chapter, the most common automatic transmission systems used nowadays are also presented, as well as some of their advantages or limitations. The first Chapter also defines the primary objectives and goals of this work.

In the second Chapter, the current AtlasCar gear selector mechanism is explained in detail, and the electronic controller solution built to control it is also described. The firmware programming, the methods used to perform it and the details about the communication messages received and sent between the control computer and the firmware are also explained in the second Chapter. Other accessory software, namely the calibration software and the ROS Node that was built to facilitate the introduction of the mechanism in the AtlasCar, are also shown in this Chapter, along with the necessary instructions or programming details about their operation.

The third Chapter of this dissertation explains in detail the simple Hardware-in-the-Loop partial gearbox simulator, written in Matlab in order to simulate and test the behaviour of the gear selector mechanism, its communication protocol and the robustness of its programming, in the laboratory.

The fourth Chapter explains some of the laboratory tests that were performed on the gear selector mechanism, both using the manual mode and the automatic mode. This second test is performed using the Hardware-in-the-Loop simulator described in the fourth Chapter.

The fifth Chapter presents the conclusions about this work. This Chapter also proposes some future work that still needs to be executed in order to perform successful gear changes on the Atlascar vehicle using this mechanism.

At the end of this work, both the bibliographic material used to develop it is described, and the annexes necessary to the complementation and further comprehension of certain aspects and parts of this work are presented.

1.5 Automatic Gearboxes

In order do develop a gear changing mechanism for the AtlasCar vehicle, and to achieve a better understanding of the problem at hand, the most common solutions commercially available were studied and explored.

An automatic gearbox, or automatic transmission is, by definition, "an arrangement of gears, brakes, clutches, a fluid drive and governing devices that automatically changes the speed ratio between the engine and the wheels of an automobile" [3]. This system was initially created to simplify the driving process, freeing the driver from the process of shifting gears or clutching. There are three main types of automatic transmission used nowadays:

- The traditional Fully Automatic Transmission;
- Continuously Variable Transmissions (CVT);
- The Semi-automatic transmission.

The most common automatic transmission system is the hydraulically operated fully automatic transmission. One of the most significant differences between the manual transmission and the hydraulic automatic transmission system is that the last uses a torque converter instead of a common mechanical clutch.

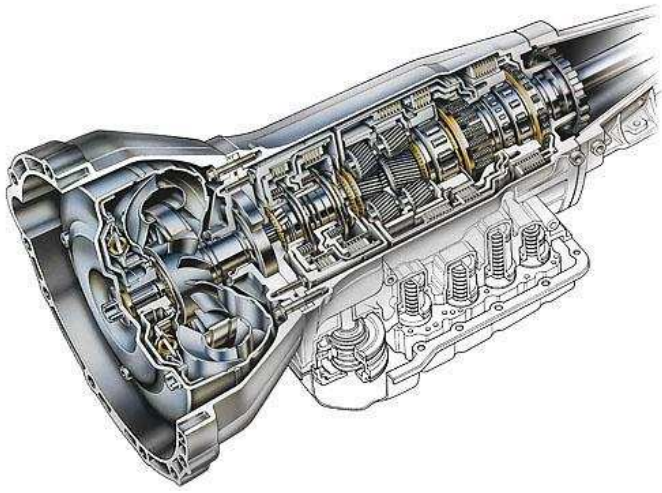


Figure 1.6: Automatic transmission system. The torque converter can be seen in the left, and the planetary gears can be observed in the middle [4].

The torque converter is a type of hydrodynamic fluid coupling used to transfer the rotating power from the engine to the gearbox itself, allowing the two systems to be mechanically separated when convenient, like the common clutch system. This system differs from the classic clutch, as the one present in the AtlasCar vehicle, because the clutch can completely mechanically connect two different shafts, contrarily to what happens with the torque converter, where a thin film of fluid is always present between the gearbox's and engine's disks [5]. The clutch is not an hydraulic system, and uses springs in order to engage or disengage the engine shaft from the gearbox shaft. The use of

clutches, instead of hydraulic torque converters, on the manual transmission systems is the main reason why this type of transmission presents more efficiency, as there is no energy loss to the operating fluid [6].

The Hydraulic Automatic transmissions also use planetary gearing with independent internal clutches, instead of the simpler meshing gears system used in the manual transmission systems, thus increasing its maintenance cost. The number of moving parts used in hydraulic transmission is considerably higher than their manual equivalent, because of all the auxiliary hydraulic circuitry, such as the necessary pumps and pipes. The hydraulic system is also composed of at least two different fluids: the lubricant and the operating hydraulic fluid, which also requires a more careful maintenance.

Figure 1.7: Steering column automatic mode selector lever from a 2010 Mercedes-Benz E350, with the four common modes: R, N D and P [7].

The fully automatic transmission interface with the driver consists of a selection lever, normally located on the steering column like the one represented on Figure 1.7 or in the car floor, on the place occupied by the gear lever on a manual transmission car. This lever allows the driver to choose from a variety of running modes, the most common of which are presented and described in the following list:

- Park (P) → The Park mode mechanically locks the output shaft of the transmission and prevents it from turning in any direction. This mode is used when the vehicle is in a stationary position;
- Reverse (R) → The Reverse mode allows the vehicle to be driven backwards, in order to perform special manoeuvres. The insertion of this mode is only allowed when the vehicle reaches a complete standstill;
- Neutral (N) → The Neutral mode works in a very similar way to a Neutral position of a manual transmission vehicle, as it disengages all gears in the transmission system. The automatic transmission drivers must always start in this position in order for the engine to start;
- Drive (D) → The Drive mode allows the transmission to use all the forward gear ratios available. This is the mode used for normal car driving.

An autonomous driving project named CADU, currently in progress at the *Universidade Federal de Minas Gerais*, in Brazil, takes advantage of the simple lever design of the semi-automatic transmission already available on the vehicle, and actuates it externally using a linear actuator [8]. This actuator only pushes or pulls the lever in the forward or backward direction, in order to select the desired running mode.

The Continuously Variable Transmissions are another type of automatic transmission system where the gear ratio can vary steplessly between a maximum and a minimum limit, defined by the manufacturer. These systems are used to provide better fuel economy, as the engine can work at its most efficient revolutions per minute value for a wide range of vehicle speeds. This system is not relevant for the study at hand, and is not described in more detail, because its functioning principle is drastically different from the transmission used in the AtlasCar vehicle.

Due to all these notorious differences between the fully automatic hydraulic transmission or the Continuously Variable Transmission described above, and the common manual transmission present on the AtlasCar vehicle, the gear shifting system principle used in the AtlasCar presents more similarities with the third automatic transmissions category presented: the semi-automatic transmission.

Unlike the automatic hydraulic transmission, which takes care of the whole gear shifting process for the driver, the semi-automatic transmission gives much more control to the driver, or the controlling computer system. The main advantage of this system is that it automatically controls the clutching process, leaving only the decision about the gear at which the system should run to the driver, or the electronic system in charge of the gear shifting logic. Typically it uses electronic sensors and actuators in order to engage or disengage the clutch and carefully synchronize its timing. This accurate timing process results in fast, smooth and effective gear shifts, which makes it the preferred transmission type for high performance applications.

The semi-automatic transmission interface with the user is similar to a manual one, but instead oh using the characteristic H-pattern gear distribution, the gear lever only moves forward and backwards to shift up or down, respectively.

Chapter 2

Automatic Gearbox

2.1 Solution Study

The original AtlasCar vehicle is equipped with a fully manual, five speed plus reverse transmission, which is still the most common type of transmission in Portugal. This system presents some important advantages for the driver. A manual transmission allows the driver to have more control on the driving, improving the behaviour of the car during high speed turns, or even avoiding unexpected and dangerous situations. It is also known for making the driving process more efficient in what concerns the engine fuel consumption, even when compared against more modern, electronically controlled transmissions available in the market [9].

Although this system presents more advantages for a human driver, it brings some new challenges and problems when the control needs to be made by a computer or a machine. The ideal approach would be to use an automatic transmission system compatible with the current engine model. Unfortunately, this solution turned out to be impossible because there were no available automatic transmissions for the engine model currently installed in the AtlasCar prototype. Although an automatic transmission was available for the 1600 cm^3 engine, there was no such system compatible with the 1400 cm^3 engine, which is the one present in the AtlasCar vehicle. A drastic measure, like changing the engine itself, would also not be feasible because of all the changes and adaptations already made on the engine and mentioned in Section 1.1, like the electronic accelerator or the second alternator. In order to keep the car as unchanged as possible, it was clear that the current fully manual transmission system should be used, with all the necessary adaptations to automatically actuate it. Given the situation, the most reasonable approach to use the current transmission would be to control the position of the gear lever, in the same manner a human driver controls it. A mechanical system, projected and built prior to the beginning of this work, was intended to do just this. This mechanism is represented in Figure 2.1.

This gear selector mechanism had already been built several months earlier, and a first attempt to control it was made during the previous term. This first attempt resorted to limit switches placed at some key points in the mechanism, in order to determine the current position of the lever, and to drive it to the next. The system was driven by a Microchip PIC. The AtlasCar control PC would calculate the direction the gear should shift, and it would communicate this order, via TCP/IP, directly to the AtlasCar PLC. The PLC would then communicate with the PIC on the gearbox mechanism controller,

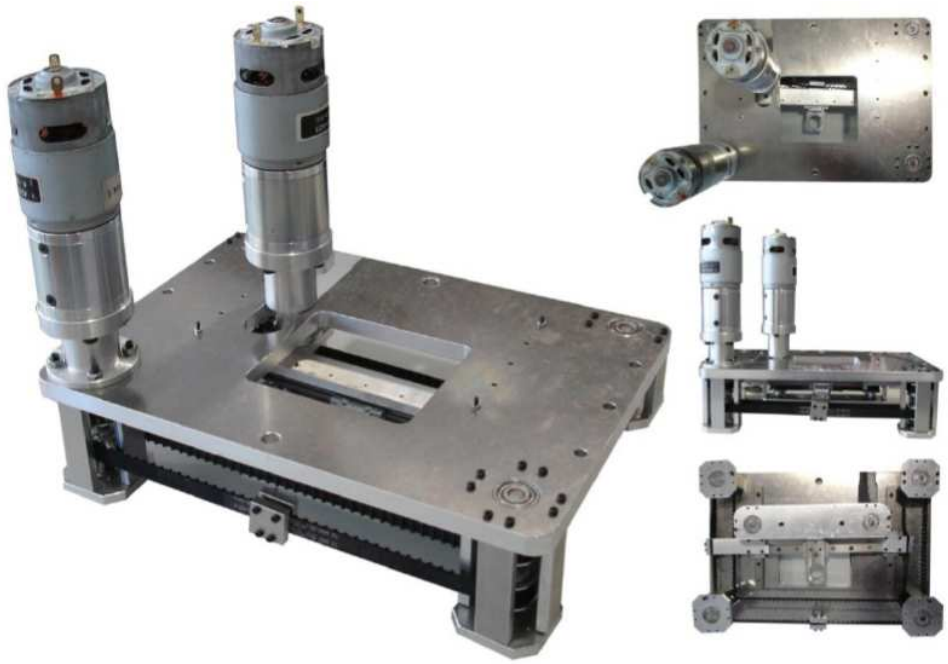


Figure 2.1: Atlascar gear selector mechanism.

and the micro-controller program would decide which movements to make in order to change to the requested gear.

This method has proven to be very prone to failure, and brought some expected and unexpected problems. The PIC programming written using limit switches was made in a purely sequential mode. To make the system memorize every path possible from each point to the another would originate a rigid and complex code. Another problem is that the control type made using only limiting switches is a bang bang with dead band control, which is not the most indicated for high responsibility applications, such as a vehicle automatic gearbox, because the control system would not know the position of the gear lever at all times. The switches also presented some issues: they jammed frequently, got run over by the DC motors, they easily got out of their mounting places and the repetitive movements of the system caused the signal wires to break from their respective leads.

The electronic circuits used to drive the DC motors in this early version had also some issues. The system was divided into three separate printed circuit boards: two individual H-bridges to drive each one of the DC motors, and one signal board used to hold the Microchip PIC, and its auxiliary circuitry. The H-bridges power circuit often overheated, and turned out not being stable enough to mount permanently on the vehicle, where they would be subjected to both vibration and heat, in a much more heavy-duty environment than the laboratory.

Another main issue presented in this first system was its difficult tuning at the moment of being placed in the AtlasCar prototype. Mounting each one of the switches on its correct position would be difficult and very time consuming. Any recalibration of the system would also take hours of effort, and should be executed by someone very knowledgeable of both the mechanical system and its programming. Due to all these known

issues, this solution was abandoned.

Most recently, some important changes took place in the AtlasCar project. The most important of these changes was the migration from the previous CARMEN based system, to the new ROS architecture, which simplified many of the complexities present in the previous and limited CARMEN system. ROS significantly improved and simplified the interprocess communication, and the constantly updated drivers and libraries, available on-line, are very helpful for those working with both hardware and software in the robotics field.

It was also decided to free the Programmable Logic Controller, a Siemens SIMATIC S7-1200 shown in Figure 2.2, which is AtlasCar's current Engine Control Unit (ECU), of some of its previous tasks. Any change on the PLC software is not easily made, because the Siemens TIA Portal proprietary software must be used. Another known issue associated with a big PLC code is the decreased TCP/IP communication frequency, which is currently performed at about 10 Hz.

According to the ROS philosophy, the big systems should be subdivided into many simpler subsystems, each one running their own simpler processes. All these subsystems should constantly exchange data between them via a solid interprocess communication system.

Figure 2.2: Siemens SIMATIC S7-1200, used as the AtlasCar Engine Control Unit (ECU) [10].

All these profound changes also affected the way in which a new gear selection system would be built. The most important change on the original project was the implementation of two multi-turn potentiometers to accurately measure the position of the lever in real-time, one for each motor. The other main change was the decision to use an *Arduino* instead of a Microchip PIC, to handle all the control logic of the system. Although they are not as powerful as the Microchip PIC systems, the *Arduino* boards are cheap, well tested and robust enough for this kind of simpler applications.

The high modularity of the *Arduino* system also brings another advantage: with small and cheap hardware modifications, big changes in the communication system can be achieved. For example, the communication process between the *Arduino* and the PC can be changed simply by replacing or adding a "Shield", and writing a few more code lines on the *Arduino* code in order to use it. The fact that the *Arduino* is already integrated in a board alongside with all its auxiliary circuitry, eliminates the need to design another new expensive PCB, print it, solder it and debug it, in order to have the platform for the micro-controller. Similarly to what happens in the case of ROS, there is also a very active, open source community constantly developing, using and testing software or hardware

solutions and applications for the *Arduino*. This constant development results in both helpful up-to-date libraries to handle specific hardware systems, and the on-line support to use them correctly.

Some communications changes were also implemented in this new gear system. Instead of a TCP/IP communication between the controlling PC and the PLC, and then a digitally coded message between the PLC and the gear selector mechanism PIC, the new communication process is made directly via TCP/IP with the *Arduino* micro-controller, using an *Arduino* Ethernet "Shield". A server is run on the Ethernet "Shield", while a client is running on a ROS node in the AtlasCar controlling PC. This client only needs to communicate to the *Arduino* server the number of the new gear. The use of an Ethernet "Shield" is necessary because the AtlasCar prototype is a very electrically noisy environment, due to all the sensors, cameras and power actuators working at the same time. The *Arduino* default communication port, a Serial USB port, is also not recommended for longer distances, such as the ones that will be required at the time of mounting the system on the vehicle.

In what concerns the power control of the DC motors, instead of using custom made H-bridges' printed circuit boards, a commercial automotive H-bridge integrated circuit is used. This simplifies the system because instead of a three custom made PCB system (one for each DC motor, and one for the controller), only one shield-like PCB would be used to drive both DC motors. The following Sections give a detailed description and explanation about each of the above subjects.

2.2 Mechanical System

The original AtlasCar gear selector mechanical system, shown in Figure 2.1 allows the movement of a metal ring in a two dimensional, horizontal plane. The gear lever is inserted in this metal ring, and the conjugation of vertical and horizontal movements allow it to drive the lever to the correct position, according to the gear desired.

The movement of the positioning ring is obtained using the combined movements of two 12 V DC motors. As the DC motors have a good rotational speed, but can not develop high torques by themselves, each one of these motors is mechanically connected to a 104:1 planetary gearbox, in order to achieve the necessary torque specifications required to move the gear lever.

The system also uses a set of toothed belts and pulleys, attached to their respective shafts. The use of toothed belts, instead of more rigid power transmission systems like worm gears, is justified to prevent the catastrophic situation of control loss over the motors. If the motors start, for some unexpected reason, moving uncontrollably, the belts will simply skip pulley teeth, and the car's transmission system, namely the lever and the vehicle's mechanism below it, will not suffer any relevant damage.

The only large modification made to this initial system was increasing the length of two of the shafts, in order to place multi-turn potentiometers. The potentiometers are coupled using two double aluminium screw couplers. Initially, a metal, possibly aluminium, potentiometers' fixation system was thought, but such rigid structure would not allow the potentiometer to perform small movements during the rotation of the shaft, in order to compensate possible misalignments between the shaft of the potentiometer and the pulley shafts, since this is an addition to the original project. The solution adopted was to use two acrylic potentiometer supports, bolted to the surface of the gear

selector mechanism, which allows the fixation system to be more flexible, thus prolonging the lifetime of the potentiometers.

2.3 Control System

As referred at the beginning of this Chapter, in order to drive both DC motors in the gear selector mechanism, an *Arduino* "Shield" was built. Actually, this "Shield" is responsible for handling the following four tasks:

- Controlling the DC motors;
- Reading the analogue input from both multi-turn potentiometers;
- Reading the MAN/AUTO switches;
- Controlling the Seven Segment Display.

Each one of these subsystems will be described in detail in the following Subsections.

2.3.1 DC Motor Controller

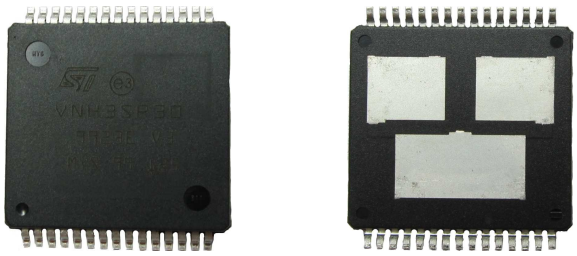
The DC motor controller used for the purpose is a commercial fully integrated H-Bridge circuit, the VNH3SP30-E, which comes in a MultiPowerSO-30 package [11], as shown in Figure 2.3. This H-bridge allows a maximum output current of 30 A and 40 V of maximum supply voltage. This integrated circuit also has integrated protection circuitry, making it easy to use and robust enough for automotive purposes, such as required for the task at hand. In order to drive each of these integrated circuits, the following signal pins must be connected to the *Arduino*:

- One Pulse Width Modulation (PWM) input for the duty cycle square wave;
- Two digital inputs for controlling the DC motor rotation sense;
- Two enable/diagnostic pins;

There is a big limitation on the number of available pins in the *Arduino* UNO, specially if an Ethernet "Shield" is used. In order to save some *Arduino* I/O pins, the two "enable/diagnostic" pins of each H-bridge can stay always active, thus always connected directly to the 5 V, since the motors can still be easily stopped if the duty cycle is set to 0%. The motors can even be locked to Ground if both In_A and In_B are set to a LOW state. Both these measures would reduce the *Arduino* pins needed to drive each H-bridge to only three: the PWM pin, and the two rotation sense pins. There is still the need for reducing the number of pins needed, in order for the *Arduino* to perform all the above tasks.

The solution found was to decode two of the *Arduino* output pins, in order to control four bits, using a 1-of-4 decoder/demultiplexer. The use of this decoder is only possible because the motors only need to operate one at a time, since the trajectories travelled along the gear selector are always in vertical or horizontal straight lines.

The decoder selected to decode the *Arduino* pins, and consequently to select the direction in which the motors should turn was the HEF4555 [12]. As shown in Figure 2.4,



(a) (b)

Figure 2.3: VNH3SP30-E top (a) and bottom (b) view. Note the three heat slugs on the bottom view of the Integrated Circuit, used to conduct the high currents allowed by this Integrated Circuit.

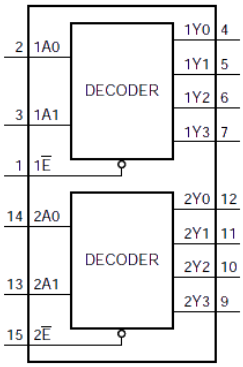


Figure 2.4: 4555 Functional Diagram.

this decoder needs only three input bits to drive four output bits. In fact this integrated circuit has two 1-of-4 decoders included, but only half of the circuit is used in the scope of this work. The input pins are the normally closed *Enable*, the $A0$ and the $A1$. The four output pins are $Q3$, $Q2$, $Q1$ and $Q0$. The truth table for this particular decoder is represented in Table 2.1, where the output pins' response to input pins combinations can be observed.

INPUTS			**OUTPUTS**			
$\overline{\mathbf{E}}$	**A1**	**A0**	**Q3**	**Q2**	**Q1**	**Q0**
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0
1	X	X	0	0	0	0

Table 2.1: The 4555 decoder Truth Table.

These measures reduce the number of *Arduino* pins needed for driving both H-bridges to only five:

1. One PWM output for the duty cycle wave of Motor 1;

2. One PWM output for the duty cycle wave of Motor 2;
3. One digital pin to drive the decoder pin A0;
4. One digital pin to drive the decoder pin A1;
5. One digital pin to enable the decoder.

2.3.2 Analogue value from the Potentiometers

As referred above, the *Arduino* needs to read the analogue value from both two potentiometers. The potentiometers used in the system to read the axis angle are the wire-wound Model 533 from Vishay Spectrol [13], depicted in Figure 2.5. During the whole physically possible path of the gear selector system, the measured axis angle is of about 450°, so the use of a three turn potentiometer is enough for this purpose.

Figure 2.5: Vishay Spectrol Mutiturn Wirewound Potentiometer [14]

The *Arduino*'s analogue input pins read values ranging from 0 to 1023, corresponding to a voltage level ranging from 0 V to 5 V, respectively. That would be the interval read from those inputs if the whole extent of the potentiometer's coil was used, which corresponds to 1080° in the axis. As the angle is only of approximately 450°, the voltage interval perceived by the analogue inputs can easily be calculated, as shown in Equation 2.1:

$$Voltage\_Interval = \frac{450^\circ}{1080^\circ} \times 5 = 2.083(3) \quad [\text{V}] \tag{2.1}$$

From which the sensibility of the potentiometer system can also be calculated, as shown in Equation 2.2:

$$Sensibility = \frac{2.083(3)}{450^\circ} \approx 4.62 \quad [\text{mV}/\circ] \tag{2.2}$$

This means that variations of almost a degree are considered by the *Arduino* analogue inputs, which is more than enough for the gear system.

In what concerns to the pins needed to read the two analogue values, each one of these potentiometers needs three independent connections:

- The 5 V pin;
- The Groung (GND) pin;
- The Output Signal pin;

Only the Output Signal pin is connected to the *Arduino*, so both potentiometers occupy two analogue inputs.

2.3.3 Manual/Automatic Switches

The system can be switched by the user from two available running modes:

- Manual Mode;
- Automatic Mode.

Each one of these modes will be explained in detail on Sub-subsection 2.3.3. It consists of a three-button system of an adapted Renault Mégane audio pad:

- Gear UP Button: Sends a pulse to increment the current gear, when the system is running on Manual mode. This signal is ignored when the system is running on Automatic mode. This button is ignored when the system is moving from one gear to another.;
- Gear DOWN Button: Sends a pulse to decrement the current gear, when the system is running on Manual mode. This signal is also ignored when the system is running on Automatic mode. This button is also ignored when the system is moving from one gear to another.;
- Man/Auto Button: This signal changes the system's running mode, and is active both in Manual or Automatic modes. This button is also ignored when the system is moving from one gear to another.

Figure 2.6: Renaul Mégane adapted audio pad.

Each one of these buttons uses an *Arduino* pin, so it occupies three additional digital inputs.

2.3.4 Seven Segment display

In order for the user to see the current gear the car is running on, a simple seven segment display was added to the system [15]. This seven segment display always represents the current gear, the car being on Automatic or Manual mode. As the number of *Arduino*

pins is very limited, a BCD to Seven Segment decoder was used, specifically the 7447 BCD to 7-Segment Decoder/Driver. This decoder uses four normally LOW input bits to drive seven output bits, used to directly drive the Seven Segment common anode display [16].

There is no need to use all the input combinations allowed by the decoder. In the AtlasCar gear system, there are only seven possible fixed positions, corresponding to the 6 gears and the neutral. Using only three of the BCD Driver's four input pins a, b and c, it is possible to display the eight integer numbers from 0 to 7, as shown in Figure 2.7. Table 2.2 shows the meaning of each one of the numbers appearing in the Seven Segment display.

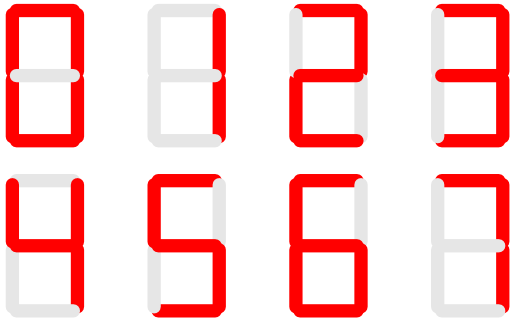


Figure 2.7: Seven Segment Display used digits.

Display Digit	Meaning
0	Neutral
1	First Gear
2	Second Gear
3	Third Gear
4	Fourth Gear
5	Fifth Gear
6	Reverse Gear
7	System in a moving state

Table 2.2: Display Number Meanings

The final 7-Segment display was assembled using a circuit prototyping board, and is depicted in Figure 2.8.

2.3.5 Printed Circuit Board

After the definition of all the control board functions it is possible to precise the number of *Arduino* input and output pins in order to build the final Control Printed Circuit Board. Table 2.3 shows all the final *Arduino* pin functions, and Figure 2.9 shows the actual location and layout of these used pins in the *Arduino* UNO board.

The most important part of the printed circuit board is the power circuit, where the two H-bridges manage the electric current in order to drive the DC motors. According to the VNH3SP30-E H-bridge driver datasheet, some external protective circuitry should be used, in addition to the protective circuitry already included in the Integrated Circuit

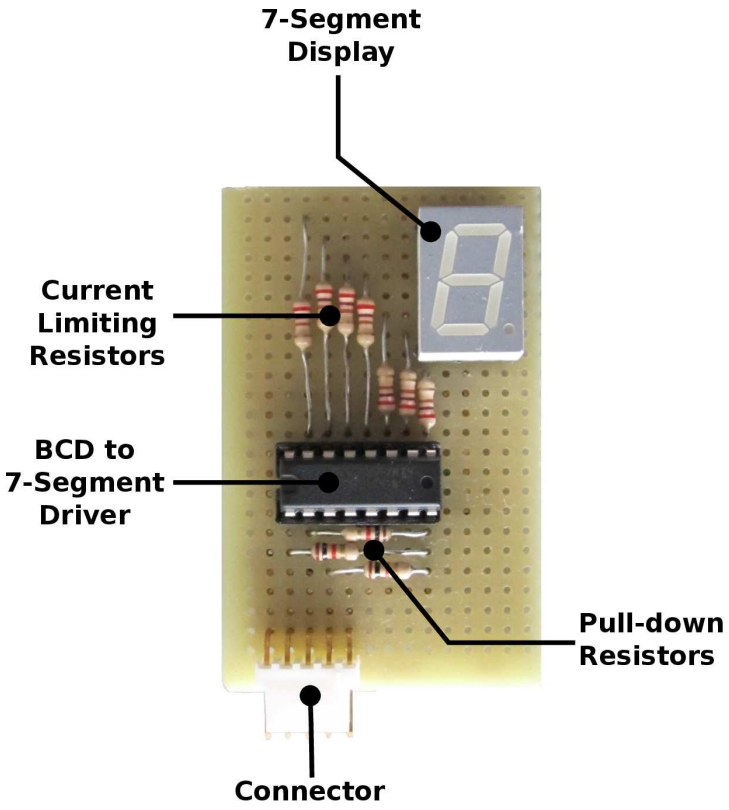


Figure 2.8: 7-Segment Display final board.

itself. The most important measure is the inclusion of a N-channel MOSFET connected to the Ground pin. This action protects the H-bridge against being connected in reverse to the car battery.

Other measure taken to prevent voltage peaks in the circuit, is the use of a zenner diode per H-bridge, so that the maximum voltage of the MOSFET gate pad is never exceeded.

An 8 A fuse is also used in the power source cables for the H-bridges. Because the circuit is directly connected to the AtlasCar vehicle battery, the fuse is used to prevent the circuit from possible high current peaks provoked by accidental short circuits.

The projected printed circuit board also contains a connection to an Emergency Stop, placed directly on the power source cables. This button allows the user to completely shut down the power part of the circuit, leaving only the signal systems still functioning.

Another common and recurrent problem in this kind of electronic applications is the electrical noise. In order to protect the circuit from the electrically noisy environment present in the AtlasCar vehicle, two capacitors were also added to filter the power voltage supply of the H-bridges:

- A 470μF electrolytic capacitor to withstand sudden voltage drops;
- A 100nF ceramic capacitor to filter high frequency noise.

The signal coming from the potentiometers has been proven to be very stable. However, and due to the inevitable proximity between the potentiometers' analogue signal

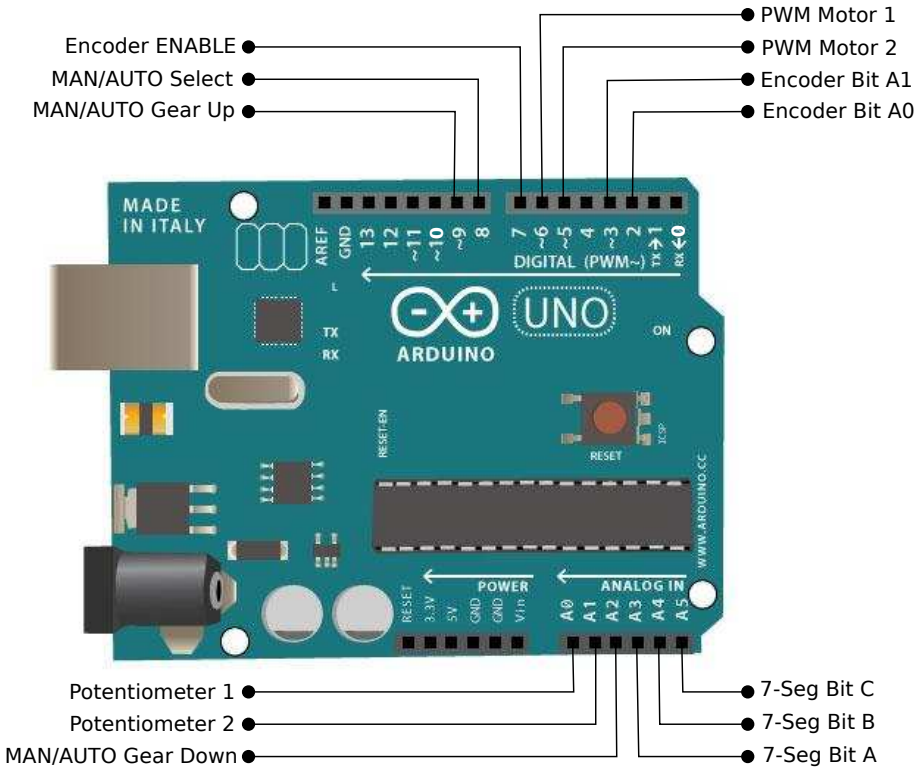


Figure 2.9: *Arduino* UNO and the available pins' functions used for this project.

cables and the DC motors, a low-pass, first order, passive filter is also used in the analogue potentiometer input pins, to attenuate possible high frequency noise.

All the necessary pull-down or pull-up resistors were placed on the necessary digital input and output pins, to avoid problems with random fluctuations on the digital signals. All the vertical connectors, which allow the Printed Circuit Board to be inserted on the *Arduino* are carefully aligned with its respective pins, to facilitate the coupling the DC motor control "Shield" and the *Arduino*.

After considering all these details and electric protections, a circuit was projected using the EAGLE PCB Design software. Both layers of the final PCB drawing can be seen in Figure 2.11, and a full scale representation of each layer can be seen in the Annexes. The layout of the components in the Printed Circuit Board was carefully defined, with particular attention paid to the width of the pathways, given the considerably high currents passing through the circuit. EAGLE 3D was also used during the project process to obtain a partial representation of the final board, as depicted in Figure 2.10. The final printed circuit board, after the soldering of all its components, is represented in Figure 2.12, and the complete system is represented in Figure 2.13.

	Pins	*Arduino* UNO	Ethernet "Shield"	Used For	I/O
Digital IO	0	RX	RX	-	-
	1	TX	RX	-	-
	2	FREE	FREE	Encoder Bit A0	Output
	4	FREE	SDCS	-	-
	7	FREE	FREE	Encoder ENABLE	Output
	8	FREE	FREE	MAN/AUTO Select	Input
	12	FREE	SPI	-	-
	13	FREE	SPI	-	-
PWM	3	FREE	FREE	Encoder Bit A1	Output
	5	FREE	FREE	PWM Motor 2	Output
	6	FREE	FREE	PWM Motor 1	Output
	9	FREE	FREE	MAN/AUTO Gear Up	Input
	10	FREE	ETHCS	-	-
	11	FREE	ETHCS	-	-
Analogue Input	A0	FREE	FREE	Potentiometer 1	Input
	A1	FREE	FREE	Potentiometer 2	Input
	A2	FREE	FREE	MAN/AUTO Gear Down	Input
	A3	FREE	FREE	7-Seg Bit A	Output
	A4	FREE	FREE	7-Seg Bit B	Output
	A5	FREE	FREE	7-Seg Bit C	Output

Table 2.3: Available *Arduino* Pins

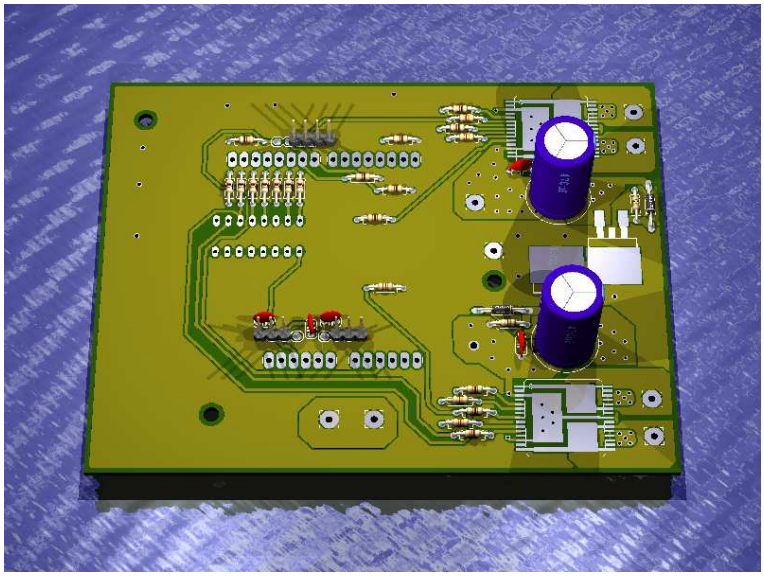
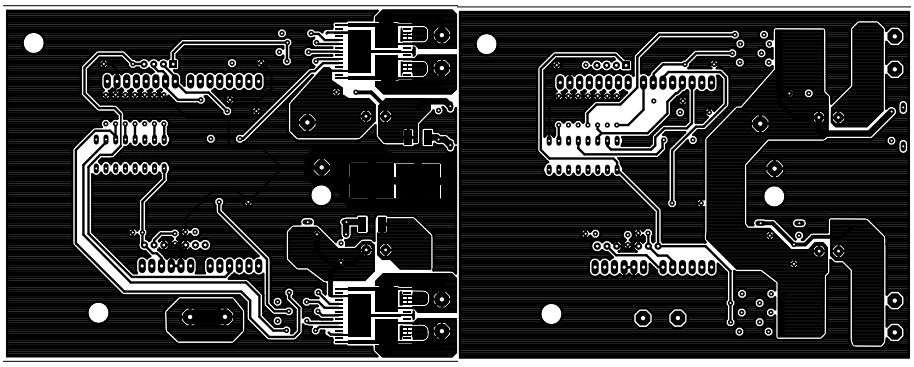


Figure 2.10: Partial PCB 3D model, created using EAGLE 3D.



(a) (b)

Figure 2.11: Printed Circuit Board Top (a) and Bottom (b) layers. The image has been scaled to fit on the page, but the real scale image can be consulted on the Annexes.

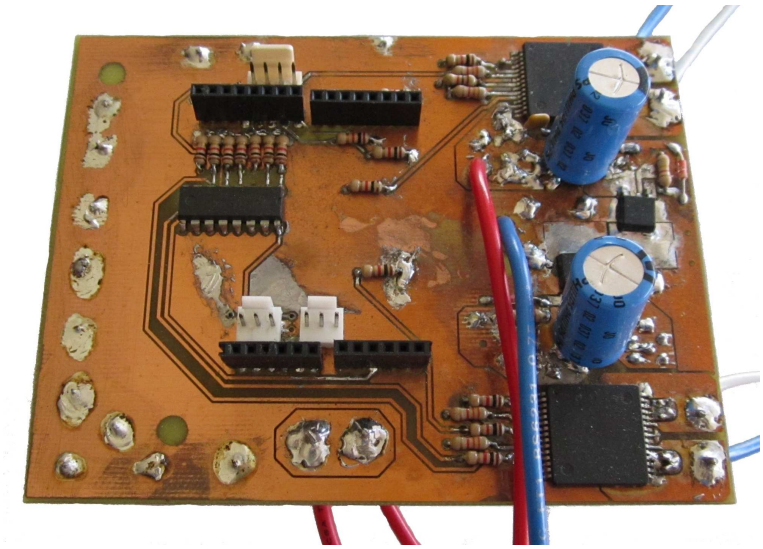


Figure 2.12: Final version of the Motor Controller Printed Circuit Board.

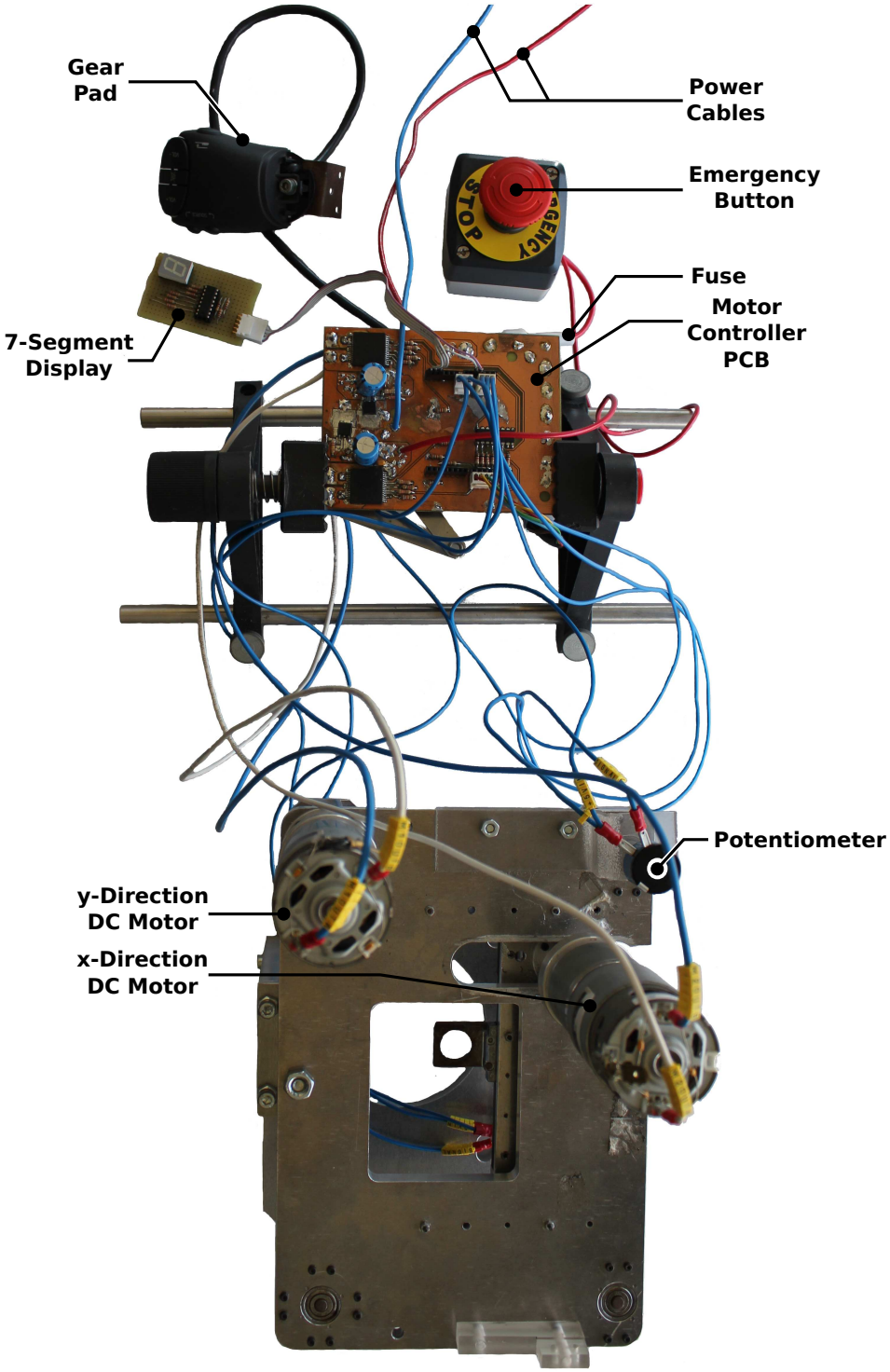


Figure 2.13: Complete gear selector system. The second potentiometer is hidden below the metal surface. The *Arduino*, and the respective Ethernet Shield are both positioned below the DC Motor Controller PCB.

2.4 Arduino Programming

The *Arduino* has two types of internal cycles: the *setup* and the *loop*. The *setup* cycle runs only once when the *Arduino* resets, or each time a TCP/IP or Serial connection is successfully established. This cycle is used to set connection parameters and run some initial routines, prior to the main program itself. The *loop* cycle runs in a continuous loop, and is where the main program is located. Both these cycles will be explained in the following Subsections.

2.4.1 The *Setup* Cycle

The gearbox *setup* cycle is used to start the communication processes, specifically the Ethernet Server which must be continuously running on the *Arduino*. The *setup* is also used to define the working mode of certain *Arduino* pins, according to their function. For example, in order to use the ADC (Analogue to Digital Converter) which is available on the *Arduino* to allow the use of certain analogue pins as digital ones, their mode must be defined here, in the *setup* cycle.

This *setup* cycle is also used to put the car in a neutral position. This is an extra safety measure to avoid any problems while starting up all the ATLASCAR systems, like the case of the clutch not being pressed at the time of the ignition. A representative flowchart of the described functions of the *setup* cycle can be seen in Figure 2.14.

2.4.2 The *Loop* Cycle

The *Arduino loop* is used to run the *Arduino* program that really controls the gear selector mechanism. During this cycle, the necessary *Arduino* inputs and outputs are sequentially scanned and made active or inactive, according to the case at hand and the programmed orders.

As mentioned above, the AtlasCar gearbox is a common H-pattern transmission. To make the program as robust and flexible as possible, the use of a sequential gearbox is to be avoided, especially because the long shifting times inherent to the vehicle actuator's physical mechanism itself. In order to make this, the mechanism must compare the current gear with the gear asked by the control PC and, if necessary, execute the appropriate movements to obey and successfully execute that command.

Figure 2.15 represents the layout of the gear positions present in the AtlasCar.

In terms of control, there's the need to analyse this system to determine its interest points, as well as the possible paths to connect those points between them. A good way to get a mathematical model for the gear positions layout is to use some concepts of the graph theory. Although Figure 2.15 is already a graph by itself, the classic graph representative of the system is shown in Figure 2.16, which shows all the physically possible paths connecting each lever position.

At this point the system can be mathematically modelled, and a solution to determine the path needed to go from point A, which represents the current gear of the system, to point B, which represents the next gear, asked by the computer or the user, can be found.

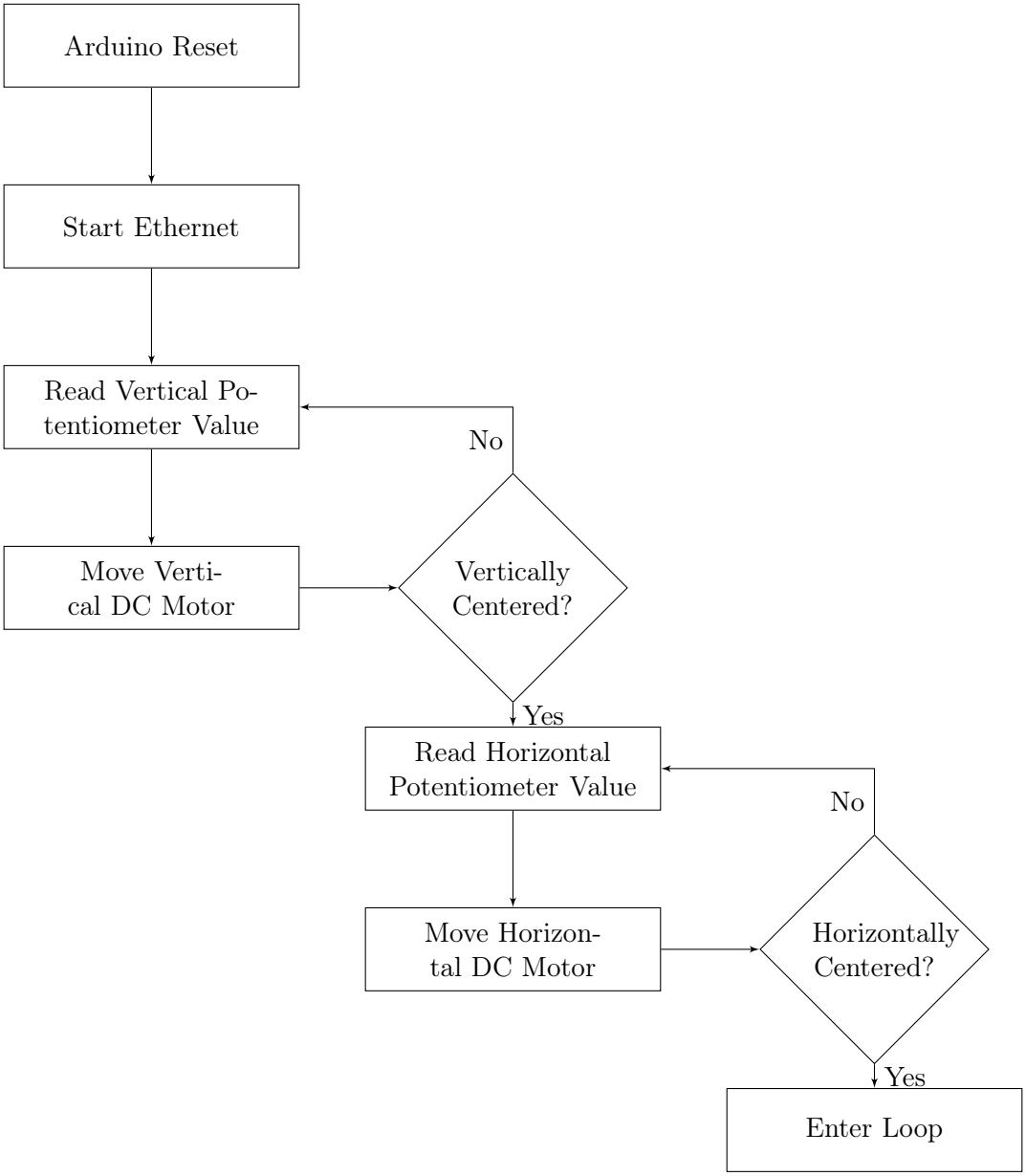
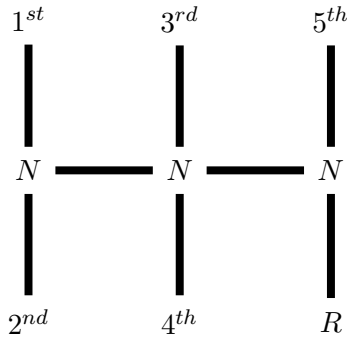


Figure 2.14: Flowchart for the *setup* cycle.

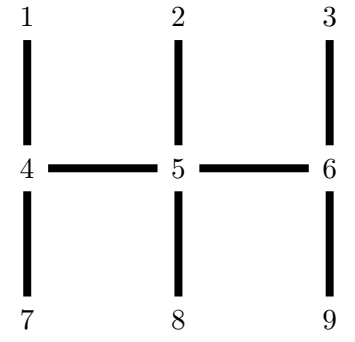
There are several methods for computing and determining the solution for a problem which is commonly known as the "Shortest Path Problem".

In the situation at hand, the system goes under a particular case of the "Shortest Path Problem", which is called the single-source shortest path problem. The most commonly used algorithms to solve this particular case of the "Shortest Path Problem", and the ones that were considered in this work to solve it are:

- The Bellman-Ford algorithm;
- The Dijkstra's algorithm;
- The A* search algorithm.



(a)



(b)

Figure 2.15: Layout for the Gear positions (a) and the respective numbering used for the *Arduino* programming (b).

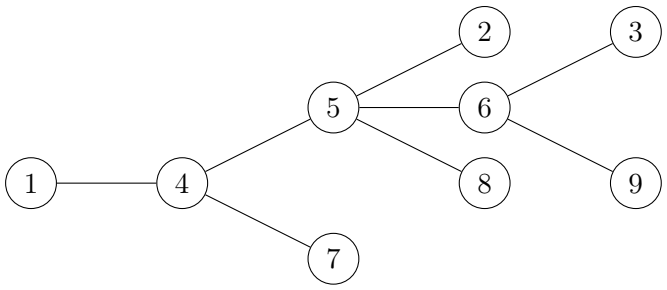


Figure 2.16: Graph representative of the gear positions and the physically possible paths between them.

All these algorithms are based on possible paths organized according to a pre-attributed weight, so that the algorithm can choose one path instead of another one, in the case there are several paths between two nodes belonging to the same graph.

The Bellman-Ford algorithm is used when these path weights can assume negative values. There is no need to use the Bellman-Ford algorithm, because the weights of each path of the gear system, used later to compute the shortest and most interesting path to follow, are never negative in the system at hand. Given the simplicity of the graph, and the need for a fast and simple method, capable of running efficiently on the limited processing capabilities of the *Arduino*. There is also no need to use a more

complex algorithm such as the A* search algorithm. This algorithm is an extension of the Dijkstra's algorithm, but it uses heuristic processes to achieve better performance, especially in what concerns to the computing time of large graph systems. These heuristic functions define the order in which the algorithm will compute each point. The system at hand is not complex enough to justify the use of this search algorithm.

Dijkstra's algorithm has shown to be the best candidate to solve this problem, and to do so in a fast way. The Dijkstra's algorithm requires the construction of a cost/adjacency matrix, representative of the graph, that the algorithm uses as a criteria to determine the shortest path between a given pair of points. For this particular system, the cost matrix M is shown in 2.3. The numbering used to build this matrix is the one present in Figure 2.15, so the first column represents the first node, the second column represents the second node, and so on. The same happens with the numbering of the matrix rows.

$$M = \begin{bmatrix} 5 & 5 & 5 & 1 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 1 & 5 & 5 & 5 \\ 1 & 5 & 5 & 5 & 0 & 5 & 1 & 5 & 5 \\ 5 & 1 & 5 & 0 & 5 & 0 & 5 & 1 & 5 \\ 5 & 5 & 1 & 5 & 0 & 5 & 5 & 5 & 1 \\ 5 & 5 & 5 & 1 & 5 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 1 & 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 & 5 & 1 & 5 & 5 & 5 \end{bmatrix} \tag{2.3}$$

Each connected path assumes a cost of 1 or 0, respectively vertical or horizontal paths. This difference between the cost values is due to the necessity of distinguishing between vertical and horizontal paths, allowing this to be done using the same matrix to represent both the adjacency between nodes and the cost of the paths linking them. The longest path that can be travelled by the gear lever is from point 1 to point 9 and vice versa (which corresponds to the shifting from a first gear to a reverse), or the path from point 3 to point 7 and vice versa (which corresponds to the transition from a fifth gear to a second gear). The cost associated with the longest path is four, so every impossible path existing in the system will take a cost value greater than three(two paths with a 0 cost, and two paths with a 1 cost), so the impossible paths are set as being fives.

During the calibration process of the mechanism, a table is constructed with the horizontal potentiometer's values of the points 1, 2 and 3, and the vertical potentiometer's values of the points 1, 4 and 7. These values are defined at the beginning of the *Arduino* code, so that the user can easily change them if necessary. The user can also define an admissible error interval for each one of the directions. These intervals are used to determine the current location of the gear lever, because the analogue values read from the potentiometers, and corresponding to a stationary position, have small fluctuations and vary overtime. When the lever is being positioned at a given point, the nominal values are used, but when determining the current gear lever position, both an x and y interval, in the vicinity of the nominal point, is considered admissible.

As soon as a new gear value is requested, and only if that value is in the interval $[0, 6]$, the program checks its current position, and determines the corresponding point. Then it calculates the corresponding point of the requested gear. Next, the Dijkstra's algorithm, running in the *Arduino*, takes as inputs the matrix M, the current point and the requested gear's point. The output of the function is a path vector containing all it's

points, including the beginning and the end points.

Next, the system distinguishes vertical from horizontal paths, using the values from the matrix M. A value of 0 stands for horizontal paths, and 1 corresponds to vertical paths. The Dijkstra's algorithm function builds the transitions vector. If the path vector has n elements, this transitions vector has $n-1$ elements. The transitions vector is the one responsible for determining which motor should move to execute the linear trajectory desired. Following this operation, a correction is made to both the path and the transitions vectors, so that the motor movements do not stop at each passing point.

The following example is useful to understand this problem. If the user asks a reduction from the 1^{st} gear to the Reverse gear, the corresponding path vector would be the following, using the numeration from Figure 2.15:

$$Path = \begin{bmatrix} 1 & 4 & 5 & 6 & 9 \end{bmatrix} \tag{2.4}$$

And the corresponding Transitions vector is the following, admitting that the horizontal paths are represented by 0 and vertical paths are represented by 1, as referred above:

$$Transitions = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

If these vectors were used directly to drive the motors, the horizontal motor would stop when reaching the point number 5, and then would accelerate again to reach point 6. This would delay the mechanical system's response even more, because these stops are unnecessary. According to the referred correction, the corrected transition vector is:

$$Transitions_{corrected} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Then, the corrected path vector is calculated, and differs from the first one in point number 5:

$$Path_{corrected} = \begin{bmatrix} 1 & 4 & 6 & 9 \end{bmatrix} \tag{2.7}$$

At this stage, both vectors are ready to drive the motors. As depicted in Figure 2.17, Motor 1 is responsible for the y direction, and Motor 2 is responsible for the x direction. The directions, represented by arrows in Figure 2.17, show the positive increment of both potentiometer's values.

A *for* cycle travels across the Transitions vector to determine which motor must move. Inside each cycle run, the system computes the direction in which the chosen motor must turn, using the values in the corrected Path vector. According to the numeration adopted, and depicted in Figure 2.15, if the current point is greater than the next point, the motor must turn in one direction, and if the current point is smaller than the next point, the motor must turn in the opposite direction. This process becomes even simpler because each motor is represented by a C++ class, and this class already contains direct commands to make each motor turn clockwise or counter-clockwise, as desired.

An acceleration and deceleration ramp, similar to the one depicted in Figure 2.18 is also calculated according to the path each motor must travel. The function that calculates this ramp is given the starting and the ending potentiometer values, as well as the minimum and maximum PWM values. These PWM values are outputs from the *Arduino*, and range from 0 to 255, which correspond to 0% and 100% motor duty cycle,

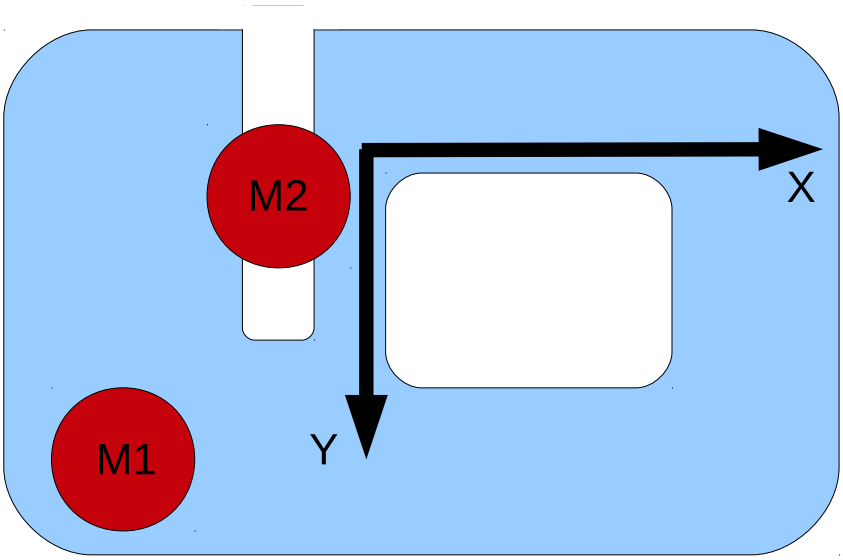


Figure 2.17: Orientation and layout of the gear selector mechanism.

respectively. The minimum, non-zero PWM value is necessary, because a zero starting value would lead the motors to a premature stop, prior to reaching the required position. Tests performed in laboratory on these DC motors showed that any duty cycle below 20% was ineffective and the motors remained still. If the acceleration ramp began at zero, the system would assume that the motors were still in motion and it would continue indefinitely to impose the physically impossible interpolated PWM value.

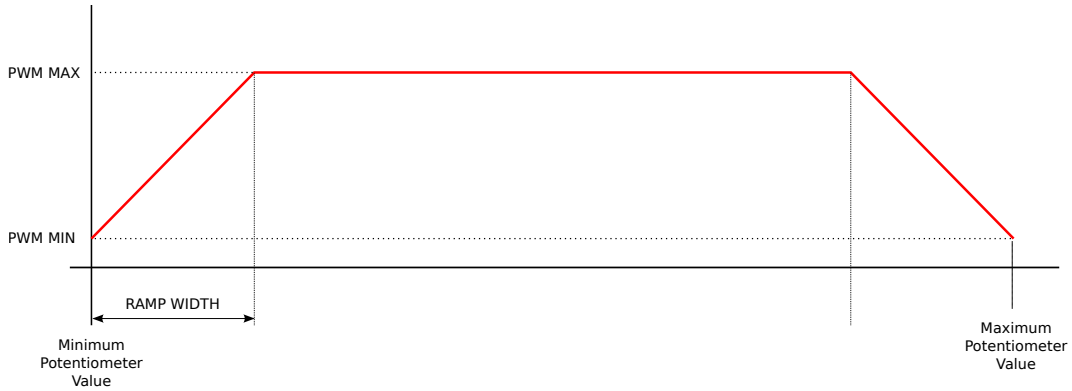


Figure 2.18: Parametrizable acceleration and deceleration ramps for the motor PWM value.

These ramps are necessary not just because they make the movements of the gear selector mechanism softer and more fluid, but because it is also a good practice to use this kind of approach to relieve the DC motors from current peaks and drastic accelerations, thus prolonging their lifetime.

After calculating the acceleration and deceleration ramps, the program enters a while

cycle, in which the correspondent potentiometer analogue value is monitored, and the interpolated PWM value is imposed to the respective motor. This procedure is repeated until all the paths in the corrected Path vector are successfully travelled.

The process described above is common to both Automatic and Manual modes, but there are some differences between these two modes in what concerns to the behaviour of the mechanism. Sub-subsections 2.4.2.1 and 2.4.2.2 describe in detail each one of these running modes.

2.4.2.1 The Manual Mode

The Manual mode is implemented as a purely sequential gearbox mode, and is controlled directly by the *Arduino* and the user. This control is possible using the *Renault Mégane* adapted sound pad, presented in Figure 2.6. All the logic is controlled by the *Arduino*, and the communication between the *Arduino* and the PC is mono-directional; this means the *Arduino* only communicates via TCP/IP the current gear of the car, and ignores any information coming from the control PC.

This sequential gearbox can only execute the shifting in the following order, without any possibility of skipping any of the steps, and without any possibility of going from the last position to the first (5^{th} gear to $Reverse$) and vice-versa ($Reverse$ to 5^{th} gear):

$$Reverse \longleftrightarrow Neutral \longleftrightarrow 1^{st} \longleftrightarrow 2^{nd} \longleftrightarrow 3^{rd} \longleftrightarrow 4^{th} \longleftrightarrow 5^{th}$$

The $5^{th} \longleftrightarrow Reverse$ transition is not allowed in this mode, not only because it is a dangerous manoeuvre which makes no sense in normal car usage, but also because the AtlasCar gearbox does not mechanically allow this direct operation. The original manual gear system, currently installed in the car, is protected against this situation, since the user must at least bring the gear lever to a central $Neutral$ position before any attempt of forcing a $Reverse$, after a 5^{th} gear.

The user can also, and at any time while the mechanism is in a stationary state, change between modes using the MAN/AUTO switching button, also available on the described button pad.

2.4.2.2 The Automatic Mode

The Automatic mode is implemented in the form of a traditional H-pattern gearbox. The user has no direct control over the shifting process, and can only switch mode using the MAN/AUTO switching button, when the gear selector mechanism is at a stationary state. When in Automatic mode, the communication between the *Arduino* and the PC is bidirectional: the *Arduino* sends the current gear of the car to the controlling PC, and the PC can set the value of the next gear.

The Automatic mode is not sequential. The *Arduino* constantly checks if a gear shifting is needed by comparing the current gear lever position with the gear asked by the PC, and it is able to calculate the next position from the previous one and execute it, using the methods described in Subsection 2.4.2. The non-sequentiality can be useful during later project stages of the AtlasCar software and hardware development, widening the window of real world situations that could get an appropriate response from the gear selector mechanism.

2.4.3 Communication Protocol

The communication between the *Arduino* and the PC is performed using an Ethernet *Arduino* "Shield", similar to the one represented in Figure 2.19. There is an Ethernet class available on the *Arduino* website, which has available a complete set of useful communication functions to use the Ethernet "Shield".

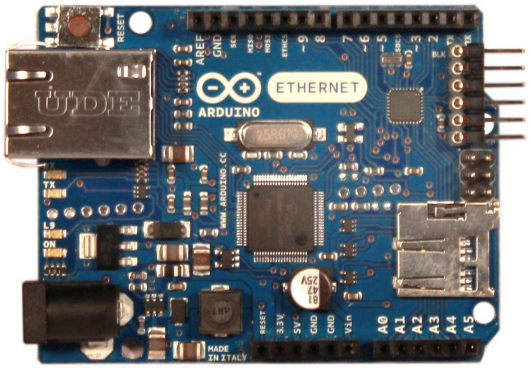


Figure 2.19: *Arduino* Ethernet "Shield" [17].

A simple, yet complete communication message system was defined to monitor the state or to set gears on the gear selector mechanism. The communication process with the *Arduino* is an "Answer to Question" type, as the server running in the *Arduino* only generates an answer if the client, running on the control PC, requests it. This communication process is used because it allows the higher level systems to constantly monitor the device, and take the necessary measures if, for some reason, the device stops responding to the commands sent. There are two main distinct message types:

- PC $\rightarrow$ *Arduino*: This message is the simplest, and its only purpose is to set a car gear or to read the current state of the gear selector mechanism;
- *Arduino* $\rightarrow$ PC: This message is more complex, because it is able to give accurate and real-time information about the current state of the gear selector mechanism.

These message types will be described in detail in the next Sub-subsections.

2.4.3.1 Pc to Arduino Messages

The message going from the control PC to the *Arduino* has the following format:

<STX> *opcode value* <ETX>

The possible *opcode* strings coming on this message, and their corresponding meanings, are represented in Table 2.4. The *opcode* is not case sensitive, and the valid *value* is an integer number in the interval $[0, 6]$.

***opcode* string**	**Message Meaning**
SG	Sets a gear in the gear selector mechanism. The *value* is not ignored in this case, and is used as the value of the gear to set.
GG	Gets the value of the current gear of the car. The *value* is ignored, if existent.

Table 2.4: Communication codes from the control PC to the *Arduino*.

2.4.3.2 Arduino to Pc Messages

There are three main message types coming from the *Arduino* to the PC:

- Answer to a "Set" command;
- Answer to a "Get" command;
- Unrecognised command message.

The first message type is used to answer to a "Set" message coming from the PC. The structure of this message is the following:

<STX>a sg *message_ code* <ETX>

The character "a" at the beginning of the message informs that it is an "Answer" message from the *Arduino*. The "sg" string indicates that the answer is made to a "Set" command. These message elements are all separated by spaces, except the *Start of Text* character and the "a", at the beginning of the message. Table 2.5 shows the possible characters sent inside the *message_ code* field, as well as their respective meanings.

***message_ code* character**	**Message Meaning**
i	Indicates that the gear asked is invalid, i.e. not belonging to the integer interval $[0; 6]$. For security reasons, the *Arduino* always checks the validity of the gear sent by the PC.
m	Indicates that the computer tries to set a gear while the system is in manual mode. The set operation is only allowed in automatic mode.
o	Indicates that the system is already at the asked gear. The "o" stands for "OK".

Table 2.5: Communication codes from the *Arduino* to the control PC - Stationary State.

The second message type is used to answer to a "Get" command sent from the control PC. This message has two different formats, which can be used to determine if

the mechanism is moving or stopped at a certain gear. In the case the system is moving from a gear to another, the message has the following format:

<STX>a gg c *previous_gear* - *next_gear* <ETX>

The "a" at the beginning of the message informs that it is an "Answer" message from the *Arduino*. The "gg" string indicates that the answer is made to a "Get" command. The "c" character stands for "Changing". The *previous_gear* field indicates the gear from where the system is moving, and the *next_gear* field indicates the gear number the system is moving to. These message elements are all separated by spaces, except the *Start of Text* character and the "a" at the beginning of the message. The purpose of this message is to let the computer or the higher level software know that the system is in motion and not in an error state.

If the system is currently stopped at a certain gear, the message sent to the control PC is the following:

<STX>a gg a *current_gear* <ETX>

Similarly to what happens in the previous message, the "a" at the beginning of the message informs that it is an "Answer" message from the *Arduino*, and the "gg" string indicates that the answer is made to a "Get" command. The "a" character, next to the "gg" string, stands for "Actual". The *current_gear* field contains the number of the selector system's current gear. All the message elements are separated by spaces, except the *Start of Text* character and the first "a" character.

The last message type is an error message generated whenever a message arriving to the *Arduino* is unrecognised, or written with a wrong syntax. This message presents the following format:

<STX>a u <ETX>

Once again, the message starts with an "a" character, informing that it consists of an "Answer" message from the *Arduino*. The "u" character stands for "Unrecognised".

2.5 The Calibration Software

As said in Subsection 2.4, the gear selector mechanism needs to be calibrated prior to its usage, to make the firmware memorize the necessary potentiometers' values to define completely the points on the H-pattern, depicted in Figure 2.15, and execute the desired gear changes effectively. This process normally consists on the following steps:

1. Connection between the *Arduino* and the PC;
2. Compilation and transferring the calibration program into the *Arduino*;
3. Monitoring the Serial port and taking note of the x and y potentiometer's values for each necessary point;
4. Redefinition of each point on the *Arduino* gear changing program, by direct code edition;

5. Compilation and transferring the gear changing program into the *Arduino*.

Although there has been the care to simplify this calibration process by defining these potentiometers' values in a very clear way at the beginning of the *Arduino* code, along with the respective comments informing about the meaning of each value, some difficulties may still arise. A user less familiarized with the *Arduino* programming language or some technical characteristics of the gear selector mechanism may not be able to calibrate it successfully, especially if an axis direction diagram, like the one depicted in Figure 2.17, is not available. The above steps may also be of difficult execution at the time of mounting the gear selector mechanism in the car. Step number 2 is especially demanding in what concerns to the user attention and to the stability of the connection, both affected by the *in-loco* calibration, as the mechanism needs to be on its final position on the car in order to perform the calibration correctly.

To allow the simplification of the calibration process described, a calibration software was created, along with a simple Graphical User Interface to use it. In what concerns to the calibrator programming, it is divided in two distinct processes that fork at the time the program is launched:

- The parent process, which is responsible for the Graphical User Interface;
- The child process, which is responsible for the communications process.

The following Subsections explain in detail the operation of each of these processes.

2.5.1 The Calibrator Graphical User Interface

The calibrator parent process launches the Graphical User Interface and manages its button callbacks and the necessary functions. The Graphical User Interface used in this project was built using GLADE, which is a programming language-independent graphical user interface builder for GTK+ [18]. All the code necessary for the handling of events, like clicking on a button or updating a text box, is written in this part of the program, as GLADE only produces a XML file containing the Graphical User Interface appearance. This file is loaded and the events are bound to its corresponding callback functions at the time of launching the program. The Calibrator Graphical User Interface can be seen in Figure 2.20.

Although the Graphical User Interface is very simple and self explanatory, the meaning of each of its fields and elements is explained in detail, according to the numeration in Figure 2.20:

1. The *Y VALUE* text box → This text box displays the current y-axis' potentiometer value. This value is constantly updated directly from the *Arduino*, using the calibration software written for this purpose;
2. The *X VALUE* text box → This text box displays the current x-axis' potentiometer value. This value is also constantly updated using direct information from the *Arduino*;
3. The *Memorize Point* buttons → These buttons store the information on the text boxes inside internal variables when the user presses them. During the calibration process, the user must place the gear lever on the corresponding point position, and click on each of these buttons in order to memorize the necessary point;

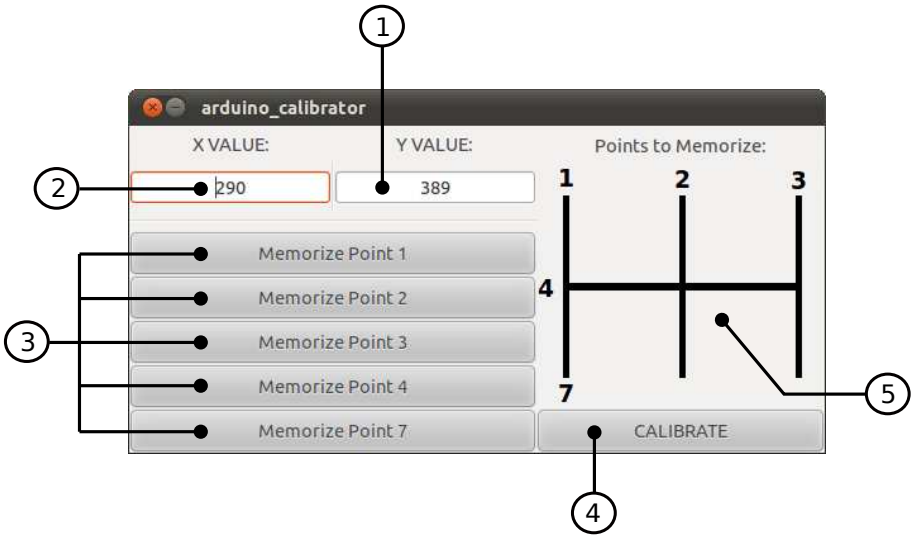


Figure 2.20: Gear Selector Mechanism Calibrator GUI.

4. The *CALIBRATE* button → When clicked, this button automatically generates the *Arduino* code, with the new calibrated values, in the same folder where the program is being executed;

5. The *Points to Memorize* Figure → This figure is not interactive, and it is only placed on the calibrator program for the user to be aware of the equivalence between the point numbering used in the *Arduino*, and its positions on the H-pattern used in the AtlasCar gear lever.

When the calibration process is terminated, the calibrator program must be closed and the file recently created, named *ARDUINO_MOTOR_CONTROLLER.pde*, must be transferred to the *Arduino* in order to start the Gear Selector Mechanism program.

2.5.2 The Calibrator Communication Process

The child process, running in parallel to the Graphical User Interface is in charge of the communication process between the PC and the *Arduino*. The communication protocol used to calibrate the *Arduino* is the USB standard Serial communication. The USB communication is used during the calibration process because it is a maintenance operation, that needs to be done near the device. The calibration process also requires the user to consecutively transfer programs to the *Arduino*, namely the calibration program followed by the calibrated gearbox firmware control code itself, which is not possible using the Ethernet communication due to the non-existence of the auto-reset necessary to the programmable integrated circuit reprogramming process.

The serial port defined in the program is the *Arduino* default communication port, the "/dev/ttyACM0", and the baud-rate is also predefined to 9600, and should not be altered.

As soon as the calibrator *Arduino* program is sent to the *Arduino*, it starts sending a message via Serial communication with the following format:

<STX>X<x_pot_value> Y<y_pot_value> <ETX>

The <x_pot_value> message field is the direct x-axis' potentiometer reading, thus an integer number between 0 and 1023, and the <y_pot_value> brings and equivalent reading but from the y-axis' potentiometer. The parallel communication process running on the PC is responsible for interpreting this message and sending both potentiometer's values to a shared memory between this process the Graphical User Interface one. These values appear on the interface text boxes, depicted in Figure 2.20.

2.5.3 The Calibration Process

The simpler calibration process, executed using this calibration software, is comprised of the following steps:

1. Connect the *Arduino* to the PC using the USB cable, and transfer of the *Arduino* calibration firmware application;
2. Execute the Calibrator Software, and make it memorize all the five points, using the respective buttons;
3. When all the points are calibrated, the *CALIBRATE* button is pressed, and the Gear Selector Mechanism control code is automatically generated and stored in the same folder where the Calibrator software is being executed in, with the name *ARDUINO_MOTOR_CONTROLLER.pde*;
4. Transfer of the *ARDUINO_MOTOR_CONTROLLER.pde* to the *Arduino*.

2.6 Integration with ROS

ROS (Robotis Operating System) is an open source software framework for robot software developers. Contrarily to the previous framework used in AtlasCar, CARMEN, ROS is based on a multitude of processes, or hosts, running in a peer-to-peer topology, instead of using a centralized server, like the CARMEN architecture [19]. ROS also allows the software development written in many different languages, such as:

- C++;
- Python;
- Octave;
- LISP.

In order to achieve this cross-language development, ROS uses a simple language-neutral message definition system where the elements carried by each message and used by each process are defined.

Another important characteristic of ROS, is the large number of small tools that can be used to get or even set information about the processes or the messages being transferred between them. These tools allow the user to measure bandwidth utilization, visualize the current topology of the running processes, graphically plot data in a very easy way or auto-generate documentation.

ROS also supplies a large variety of useful drivers and libraries that allow the user to communicate with the sensors or hardware used within the project being developed. The fact that the ROS is an open-source project also enables a very active on-line community, constantly writing and using new software on the ROS project.

The ROS implementation is divided in four distinct concepts:

- Node;
- Message;
- Topic;
- Service.

The Node is the name given to a certain process running within a ROS project. A project built using the ROS architecture is normally comprised of many Nodes working separately, and constantly communicating between them.

The ROS Nodes communicate with each other using Messages. A Message is an auto-generated data structure composed mainly of primitive types, and can include other Messages nested inside it.

The Topic is a string where a certain message is published, making its contents available to all the other ROS Nodes. When a Node, or a set of Nodes, needs to read the information contained inside a message published by another Node, it subscribes the corresponding Topic.

A Service is used when the topic-based publish-subscribe model is not advised for a certain Node that requires a synchronous communication with another Node. A Service is comprised of two distinct Messages, one user for the request, and the other used for the response.

In order to allow the future installation of the gearbox mechanism on the AtlasCar, and to integrate the system on the ROS based architecture currently being used, a ROS Node was created: the Gearbox Node. The main function of this Node is to implement the TCP/IP communication protocol, described in Subsection 2.4.3, with the *Arduino* firmware. The Node generates a Message and publishes it to a specific Topic, which is made available to any other Nodes in the AtlasCar. This node also subscribes another Topic, which contains the information about the gear that needs to be set on the gear changing mechanism. Note that it is not on the scope of this node to implement any gear logic on the AtlasCar, since the only purpose of the Node is to send orders to, or receive the status update from the gear selector mechanism's controller. Any gear shifting logic must be implemented on a higher level Node that communicates with this one, which is not the objective of this work. For this reason, the ROS Messages published and subscribed by this Node were intentionally kept as simple as possible.

The command message being subscribed by this Node has only one field, apart from the "priority" and the "lifetime" fields, that are required for ROS to make the automatic Topic and Message management. This field is called "gear", and is an unsigned 32 bits integer on the interval $[0, 6]$, as said in Sub-subsection 2.4.3.1.

The status message being published by this Node is more complex, due to the fact that it needs to inform the controlling PC about the accurate status of the gear selector mechanism device. This Message has two fields: the "gear" and the "status". The gear field is an unsigned 32 bits integer and contains the current gear on the gear selector

mechanism. If the device is moving, this integer corresponds to the last gear that was inserted by the device, thus the gear from where the mechanism left from. The "status" field is a string, interpreted by this Node according to the message read from the gear selector mechanism's control system. Table 2.6 shows the possible strings that can appear on this Message field, as well as the respective meanings.

***status* string**	**Message Meaning**
manual mode	Indicates that the control computer is trying to set a new gear on the gear selector mechanism while it is in manual mode, since the set operation is only allowed in automatic mode. To prevent or correct this situation, the MAN/AUTO button on the steering column gear pad selector must be pressed.
command invalid	Indicates that the gear asked to the gear selector mechanism is invalid, i.e. not belonging to the integer interval $[0; 6]$. This check is always made by the *Arduino* firmware code.
changing to gear_number	Indicates that the system is currently moving from one gear to another. The gear from where the system left is represented in the "gear" field, on the Node Message. The *gear_number* represents the gear to where the system is moving to.
ok	Indicates that the system is currently stationary on the desired gear number.
unknown status	Indicates that the Node could not interpret the message coming from the gear selector mechanism. This string indicates that something happened with the device, and security measures should be taken.

Table 2.6: Possible Strings being published by the ROS Node.

Chapter 3

Partial Gearbox Simulator

3.1 The Power Train System

In order to perform laboratory tests on the gear selector mechanism described on Chapter 2, prior to its installation in the AtlasCar vehicle, a simplified partial power train and car physics simulator was created using Matlab. The purpose of this simulator is to test the robustness of the *Arduino* programming, the shifting logic used, the communication protocol implemented and to evaluate shift times.

The simulator is built in a way that allows the gear selector mechanism to be tested using the Hardware-in-the-Loop (HIL) simulation technique. This technique is used to simulate the real-time behaviour of a particular embedded system which is only a component of a more complex system. This more complex system, where the embedded component will be inserted, is usually emulated using representative mathematical models.

In the context of this study, the gear selector mechanism represents the embedded component, and the systems that will be mathematically modelled are the power train system and some aspects of the vehicle's physics. Note that the aim of this simulator is not to obtain a very accurate representation of the vehicle's dynamics, or even to reproduce the precise behaviour of the engine, clutch or other key power train components.

The power train system of a vehicle is the set of all the components necessary to generate, transform and transmit power, in order to move it. The most important parts on the power train system are:

- The Engine;
- The Clutch;
- The Transmission;
- The Drive Shafts;
- The Differential;
- The Wheels.

Figure 3.1 shows the energy flow between the elements of the power train. Even today, some of the elements represented have no accurate representative mathematical model, or the existing models require a lot of computational power to run in real-time applications,

like the one at hand. Other elements have non-linear behaviour, like the clutch or the interaction between the wheels and the road. New and more accurate physical and mathematical models for some of the vehicles' main components are constantly being developed and improved [20].

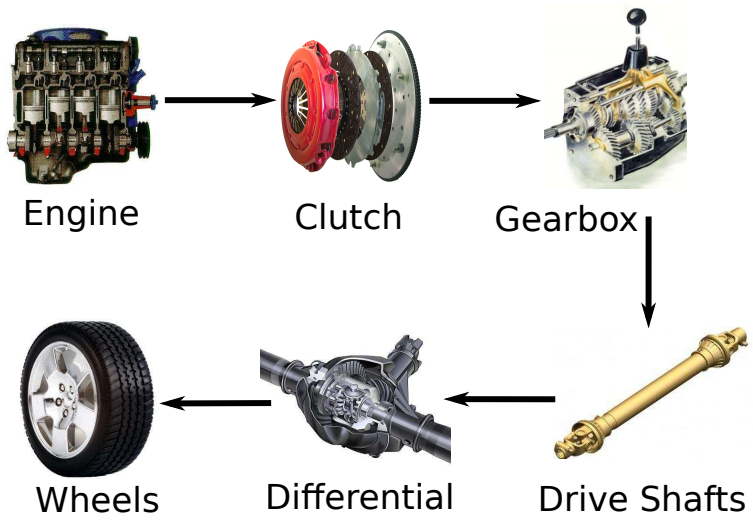


Figure 3.1: Power Train movement flow scheme between parts.

On the next Subsections, the physical model of the car is presented, and a description on how the various systems are simulated, and the way how they interact with each others, is detailed.

3.2 Car Physics

In order to model the car physics, a simple two dimensional model of the vehicle was created. This model of the car takes into account the five main forces acting on a car while it is in motion [21]. The considered forces are the following:

- Engine Force (F_{eng})
- Drag Force (F_{drag})
- Force of Gravity (F_g)
- Rolling Resistance Force (F_{rr})
- Braking Force (F_b)

Figure 3.2 shows the forces used in this model, as well as the coordinate system used. The resulting force acting on the car is the sum of all the above:

$$F_r = F_{eng} + F_{drag} + F_g + F_{rr} + F_b \tag{3.1}$$

Each of these forces must be calculated separately, and then the resulting force acting on the vehicle is computed. With the force acting on the vehicle and its mass, it is possible to calculate the acceleration value at a given time. The vehicle speed value on the next

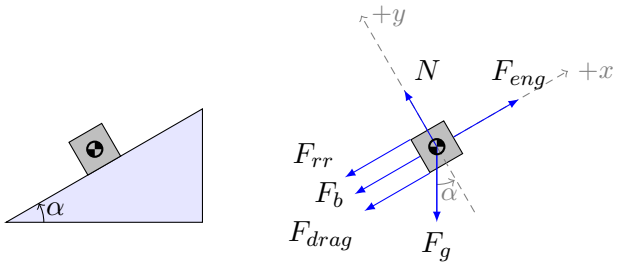


Figure 3.2: Force Diagram used in the vehicle simulator.

iteration of the simulator can then be calculated using the Euler method for numerical integration,as shown in Equation 3.2:

$$v(t + t_{step}) = v(t) + t_{step} \times a \tag{3.2}$$

Where:

- $v \longrightarrow$ The vehicle speed $[m/s]$;
- $t \longrightarrow$ Current time value $[s]$;
- $t_{step} \longrightarrow$ The time step of the integrator $[s]$;
- $a \longrightarrow$ Last acceleration value, calculated using the resulting force $[m/s^2]$.

The next Subsections explain in detail how each one of these forces, used to compute the resulting force acting on the vehicle, are calculated.

3.2.1 The Drag Force

The drag force is the main responsible for the loss of speed experienced by an object when it's travelling at higher speeds. The following equation gives a relatively reasonable value for the force exerted by the wind in the car, while in motion. It is also assumed that the fluid where the car is immersed, air in this case, is in a stationary sate. The force can be calculated using the Drag Equation [22]:

$$F_{drag} = \frac{1}{2} \times \rho \times C_d \times A \times v^2 \tag{3.3}$$

Where:

- $\rho \longrightarrow$ The density of the involving fluid $[kg/m^3]$;
- $C_d \longrightarrow$ The Drag Coefficient, a dimensionless quantity characteristic of the vehicle;
- $A \longrightarrow$ Frontal Area of the vehicle $[m^2]$;
- $v \longrightarrow$ Current speed of the vehicle $[m/s]$.

The sign of this force is opposite to the current speed vector direction. When the car is stopped, its value is zero. When the driver is driving forward, the force is exerted pointing to the back of the car, and vice-versa.

3.2.2 The Rolling Resistance Force

The rolling resistance force is the force experienced by the car caused by the friction between the tires' rubber and the surface of the road. This force is calculated using the normal force and a dimensionless coefficient, called the "rolling resistance coefficient". This is the most notorious resistant force acting on the vehicle while it is travelling at lower speeds. The rolling resistance force equation is as follows [23]:

$$F_{rr} = C_{rr} \times N \tag{3.4}$$

Where:

- $C_{rr} \longrightarrow$ The "rolling resistance coefficient";
- $N \longrightarrow$ Normal Force $[N]$;

The Normal Force can be easily calculated using the weight of the car, the angle of the road and the acceleration of gravity, using the following expression:

$$N = m \times g \times \cos(\theta) \tag{3.5}$$

Where:

- $m \longrightarrow$ Car mass $[kg]$;
- $g \longrightarrow$ Acceleration of Gravity $[m/s^2]$;
- $\theta \longrightarrow$ Angle of the ground plane.

For pneumatic tire vehicles, the rolling resistance coefficient can be estimated using an empirical expression. This expression depends on the tires' pressure and the current vehicle speed [24]:

$$C_{rr} = 0.005 + \frac{1}{P} \times \left(0.01 + 0.0095 \times \left(\frac{v}{100}\right)^2\right) \tag{3.6}$$

Where:

- $P \longrightarrow$ Tire Pressure $[bar]$;
- $v \longrightarrow$ Current speed of the Vehicle $[km/h]$.

Merging these three equations, a reasonable approximation of the rolling resistance force can be computed. The sign of this force is also given relatively to the direction of the speed vector. Similarly to what happened with the drag force, if the car is driving forward, this force is pointing backwards, and vice-versa.

3.2.3 The Braking Force

The braking force can be calculated by predefining a maximum braking force. In order to simplify the approach, it is assumed that the slip between the tires and the road is non-existent, and that the actual force exerted by the brakes at a certain time is proportional to an user input, which represents the percentage of the brake pedal pressed. The maximum braking force is left as a configurable parameter, and should be defined prior to running the program. The equation that represents this proportional behaviour is the following:

$$F_b = percent \times F_{b_{max}} \tag{3.7}$$

Where:

- $percent \longrightarrow$ Percentage of the brake pedal pressed;
- $F_{b_{max}} \longrightarrow$ Maximum Braking Force $[N]$.

The braking force is also always pointing in the opposite direction of the speed, and it assumes a zero value when the car is stopped.

3.2.4 The Gravity Force

The gravity force is calculated simply by projecting the weight of the vehicle on the x direction, and can be expressed by the following equation:

$$F_g = m \times g \times \sin(\theta) \tag{3.8}$$

Where:

- $m \longrightarrow$ Car mass $[kg]$;
- $g \longrightarrow$ Acceleration of Gravity $[m/s^2]$;
- $\theta \longrightarrow$ Angle of the ground plane.

Note that there is a negative sign on the equation, because when the angle is negative, it means that the car is descending a road, so the force exerted on the car is positive.

3.2.5 The Engine Force

The internal combustion engine is the power source for the whole car system. The engine is responsible for the conversion of the chemical energy, stored in the vehicle's fuel, into rotational movement. The interaction between all the thermodynamic and mechanical processes that occur inside an internal combustion engine make it a particularly difficult system to model, as a whole.

The engine simulation is another subject still being developed and studied further, and there are some works in the subject, but the model systems proposed are too complex to be applied to the subject at hand. Because of all the complexities associated with the modelling the internal combustion engine itself [25], the approach used in this simulator is a very simplified model, and it requires the user to know or measure some parameters

about the engine itself, as well as some parameters regarding the transmission process itself. Although the engine simulation used is simplified, it is obtained using a single function, that can easily be replaced by a more complex simulation process, if necessary.

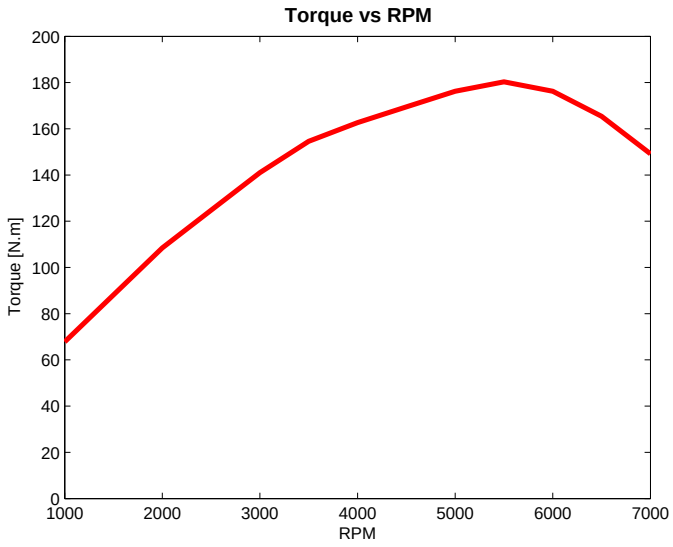


Figure 3.3: Torque vs RPM of a 1999 Dodge Neon DOHC engine.

First, there is the need to define the Torque versus RPM curve for the engine. This curve is obtained performing a dynamometer test on the engine, and it allows the knowledge about the maximum torque value that can be achieved by a certain internal combustion engine at a given RPM value. A 1999 Dodge Neon DOHC engine curve is represented in Figure 3.3 [26]. The problem in using the values from this curve is that the dynamometer test is driven at the maximum load of the engine, meaning that the throttle is fully opened. In this simulator, and in order to simplify the system as much as possible, the torque value for a given RPM is assumed to be proportional to the percentage of throttle pedal pressed, so:

$$\tau = throttle\_percent \times \tau_{max}@RPM \tag{3.9}$$

Where:

- $throttle\_percent \longrightarrow$ Percentage of throttle pressed;
- $\tau_{max}@RPM \longrightarrow$ Maximum Torque at a given RPM, directly interpolated from the Torque versus RPM curve $[N \cdot m]$.

The RPM value is calculated directly from the vehicle speed, using the gear ratios and assuming that there is no slip between the tires and the road.

At this point, the engine force can be calculated using the following equation:

$$F_{eng} = \frac{\tau \times x_g \times x_d \times \eta_{trans}}{R_{tire}} \tag{3.10}$$

Where:

- $\tau \longrightarrow$ Current Torque $[N \cdot m]$;
- $x_g \longrightarrow$ Current Gear Ratio;
- $x_d \longrightarrow$ Differential Ratio;
- $\eta_{trans} \longrightarrow$ Transmission efficiency;
- $R_{tire} \longrightarrow$ Tire Radius $[m]$.

3.3 The Simulator GUI

Although the use of a SIMULINK model was initially considered, the simplicity of the physics involved on this simulator, along with the nature of the simulation itself, did not justify its usage, since the aim of this simulator is not physical accuracy. The values used to calculate some of the forces described above are based on standard values, and they do not correspond to the real AtlasCar vehicle values. Some examples of this are the gear ratios or the engine Torque versus RPM curve. Extensive research has been performed in pursuit of such values, both on-line, in car catalogues, magazines and even through the vehicle manufacturer, but no information was found or provided about this particular car model. However, the simulator was written in such a way that the main force calculations are performed in separate functions, which can easily be replaced at any time when more realistic values are available, or more accurate models are developed and implemented.

A Graphical User Interface (GUI) was created to allow the user to change some real-time driving parameters, and to show the current state of the simulator. When the program starts, a first Graphical User Interface is displayed, the Parameters Graphical user Interface, as shown in Figure 3.4 . This first Graphical User Interface allows the user to change the simulator starting values, physical conditions and parameters or vehicle specifications that are constant throughout the simulation.

The values that can be modified in this Graphical User Interface are divided in two main categories: the *Physical Characteristics*, which are physical values and quantities of the environment where the car is inserted, and the *Car Characteristics*, that represent attributes of the vehicle itself. The corresponding units, that allow the simulator to work correctly, are also displayed in front of each editable text fields.

The *Physical Characteristics* that can be changed in the scope of this simulator are the following:

- The maximum road angle, which corresponds to the angle between the road plane and the plane tangent to the theoretical earth curvature at this exact point;
- The gravity acceleration, as this value is not constant along all points on the surface of the planet;
- The density of air, that can also vary according to the altitude, temperature and humidity;
- The drag coefficient of the vehicle, which is a representative dimensionless quantity;
- The initial speed of the car, which is the starting speed of the simulator;

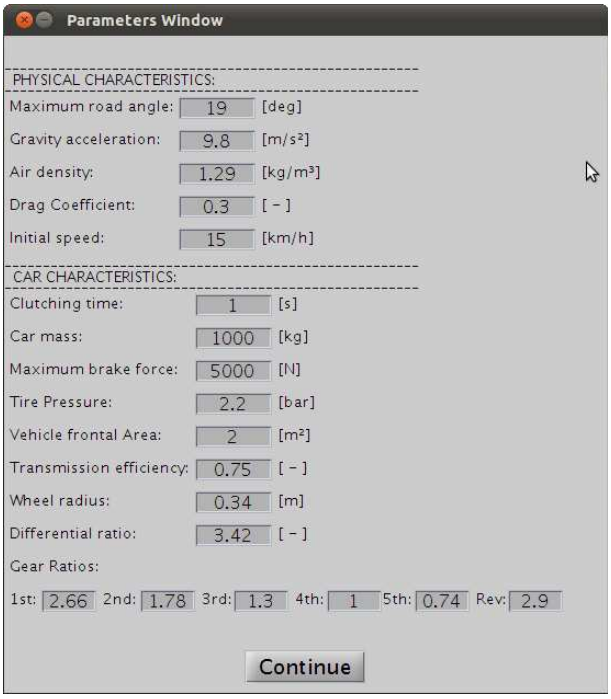


Figure 3.4: Parameters Graphical User Interface of the simulator.

The *Car Characteristics* that can be edited and changed in order to vary the response of the simulator to the user actions are the following:

- The clutching time, which represents the time in seconds that the clutch takes to go from the engaged state to the disengaged one, or vice-versa;
- The car total mass, including the load mass;
- The maximum brake force, which is the force exerted by the brakes on the car if the brake pedal is completely pressed;
- The tire pressure in *bar*, assuming that all the tires are at the same pressure.
- The frontal area of the vehicle, which is used to compute the Drag Force, caused by the movement of the car in the surrounding air;
- The transmission efficiency, which corresponds to the ratio of usable energy, or the fraction of energy not lost during the movement of the transmission;
- The wheel radius, or the radius of the car tires;
- The differential ratio;
- The gear ratio for all the seven different gears of the car.

The main Graphical User Interface, depicted in Figure 3.5, has a set of vertical bars that allows the user to manually change some of the driving variables. Real-time graphics

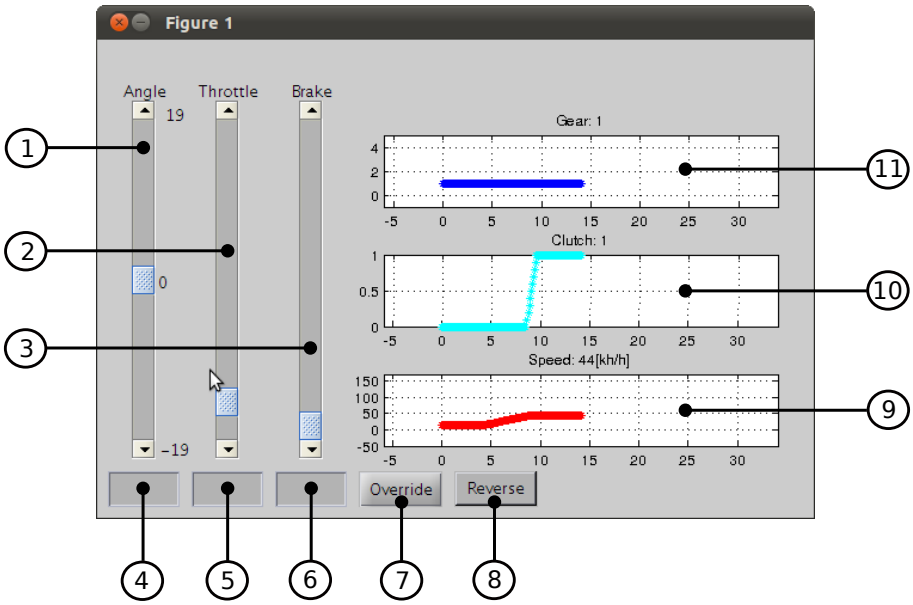


Figure 3.5: Main Graphical User Interface of the simulator.

are constantly updated accordingly to the changes. The user can also override the values present in the vertical bars at each time, in order to impose specific values to the system.

The meaning of each of the Graphical User Interface fields and elements is explained in detail, according to the numeration in Figure 3.5.

1. *Angle* bar → This vertical sliding bar allows the user to make the road angle vary overtime. This bar's default value is a zero, corresponding to a perfectly horizontal road. The *Angle* bar has a maximum and a minimum value, the last being the symmetric of the first. As described in Subsection 3.2.4, a negative angle means that the car is descending a road, thus accelerating;

2. *Throttle* bar → The *Throttle* vertical bar allows the user to vary the percentage of throttle, and consequently increase or decrease the force exerted by the engine. The default value for this bar is zero, corresponding to the bottom position. The top position corresponds to the maximum throttle;

3. *Brake* bar → The *Brake* vertical bar allows the user to change the force exerted by the car brakes. This force is proportional to the maximum brake force of the car, defined previously in the parameters Graphical User Interface, as described in Subsection 3.2.3;

4. *Angle* override → The *Angle* override editable text box allows the user to impose a specific value for the road angle. The value must be between the maximum and minimum road angle values, or it is ignored;

5. *Throttle* override → The *Throttle* override editable text box allows the user to impose a specific percentage value for the throttle; If the value is greater than 100%, a 100% value is assumed, and if the value is negative, a 0% value is assumed;

6. *Brake* override → The *Brake* override editable text box works in a similar way to the *Throttle* override editable text box, only it affects the brake percentage value;

7. *Override* push-button → The *Override* push-button collects the values from the override editable text fields and applies them to the respective variables, and updates the vertical bars. If a field is left blank, its value remains unchanged;

8. *Reverse* button → The *Reverse* button is used only when the car is in a stationary position, and informs the simulator about the user's intention of direction. This button is a particular kind of button called "Toggle Button", and it visually informs about its state. If the button is pressed, it means that the "reverse" is selected, and that the car will start moving backwards. If the button is not pressed, the car will start moving forward;

9. *Speed* graph → The *Speed* graph is constantly being updated and shows the current speed of the car, in km/h. The numerical value of the car is also represented in the *Speed* graph title.

10. *Clutch* graph → The *Clutch* graph shows the position of the clutch overtime. This graph is continuous and varies between 0 and 1. While clutching the graph is a parametrizable ramp, as the time the clutch needs to actuate can be changed in the initial Parameters Graphical User Interface. The clutch state can also be seen in the *Clutch* graph title;

11. *Gear* graph → The *Gear* graph is a stepped line that shows the gear on the gear selector mechanism overtime. The current gear can also be seen in the *Gear* graph title.

This simulator is based on a state machine, whose states and possible transitions between them are represented in Figure 3.6. The vehicle is considered to be stopped when its speed is too low for the engine to stay mechanically connected to the wheels, preventing it from stalling. The default value considered on this simulator for the minimum vehicle speed is 5 km/h when the car is moving forward, or -5 km/h when the car is moving backwards. In the stopped state, the clutch is fully pressed, in order to keep the engine from stalling, and the gear selector mechanism is set to Neutral, for safety reasons. The *Vehicle Stopped* state is in fact an indetermination, from the system's point of view, since there are three possible situations that can be experienced by the system:

- The car continues stopped;
- The car starts moving forward;
- The car starts moving in reverse.

The condition that needs to be fulfilled in order for the simulator to resolve this undetermined state is the increasing of throttle input from the user. If the user presses more than a certain percentage of the throttle pedal, 30% being the default value, the systems assumes the user's intention to start moving. In order to choose between the forward or reverse situations, the simulator uses the information read from the *Reverse* button.

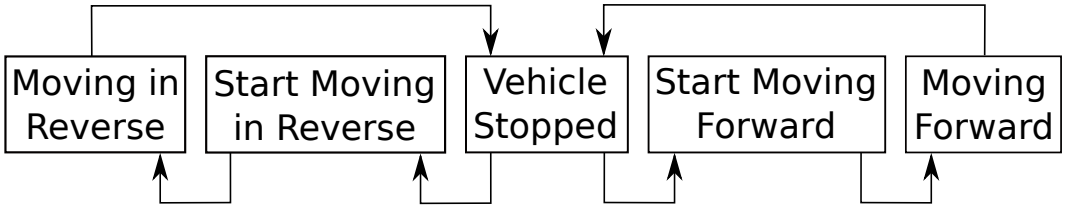


Figure 3.6: State Machine scheme used for the simulator.

If the system evolves in the forward direction, the first gear is set on the gear selector mechanism, and when the gear shift is complete, the system releases the clutch while a pre-programmed start routine is imposed to the system, increasing the speed inversely proportional to the clutch movement. This simple routine happens inside the state called *Start Moving Forward* in the scheme from Figure 3.6. When the system ends the *Start Moving Forward* state, enters automatically on the *Moving Forward* state, which corresponds to the normal behaviour of the car, shifting gears as it seems fit, according to the gear shifting logic applied.

If, on the other hand, the system evolves in the Reverse direction, the Reverse gear is set on the gear selector mechanism, and the clutch is released while a pre-programmed start routine, similar to the one described above for the *Start Moving Forward* state, is imposed to the system, only this time the speed increases in the negative sense. When this routine ends, the system automatically enters the *Moving in Reverse* state, allowing the user do accelerate or brake normally, but having only the Reverse gear available. If the user needs to start driving forward again, and similarly to what happens on the real semi-automatic transmissions, the vehicle must first enter the *Vehicle Stopped* state.

3.4 Shifting Logic

The gear shifting is a complex decision process performed in a very different way from driver to driver. Some people constantly look to the information displayed in the car's tachometer to decide if a gear shift is needed, others base this decision solely on the speed of the car, or even the engine sound. A machine, such as the controlling computer in the AtlasCar does not currently have such a high level of perception, even with the variety of sensors currently installed in the car.

In order to decide the most appropriate gear ratio for the car, at a certain time, the most obvious inputs of this control system would be the current vehicle speed, the throttle pedal position or the engine RPM value. Many other inputs could be used to improve the process, such as:

- The road angle, between its plane and the horizontal plane;
- Inertial information;
- Engine sound pitch;
- Constantly updated information about the engine torque;
- Environment information, such as the relative position to other cars;

- Information about the car trajectory, such as bumps or tight corners.

Although some of these inputs are currently present in the vehicle, its usage is complex because they are still under a development stage. It is the case of the inertial information or the environmental data collected from the lasers and cameras.

The sifting model explained in this work, and the one implemented in the Hardware-in-the-Loop simulator, only uses the information about the vehicle current speed and throttle pedal position. A similar model is used in other works of the same type, to decide which gear the vehicle should be running on. One of these examples is a Matlab SIMULINK Demo called "Modeling an Automatic Transmission Controller" [27], in which a gear schedule is implemented for a four-speed vehicle. Another similar approach can be observed on the article "Control of Integrated Powertrain With Electronic Throttle and Automatic Transmission" [28], with the slight difference that the power percentage is used instead of the throttle pedal position.

The first and simpler way to determine the current gear, at which the vehicle should be running, is to build a fixed shifting schedule. This shifting schedule can be represented as a two dimensional space, where one dimension's values correspond to the percentage of throttle pressed at a given moment, and the other dimension's values correspond to the car speed at that same moment. Although the percentage of throttle pressed and the car speed are time-dependent quantities, the schedule itself is pre-programmed in the car, and time-independent.

The only purpose of this two dimensional space is to determine the appropriate gear for the car to be running on at a given time, according to the percentage of throttle and the vehicle speed at this given time. A graph representative of this approach, and the first that was used to determine the gear shifts in earlier versions of the Hardware-in-the Loop simulator, is depicted in Figure 3.7.

The abscissa's values in the graph of Figure 3.7 correspond to the percentage of throttle pressed, and the ordinates' values represent the current vehicle speed. At each iteration of the simulator, the function that implements the shifting schedule determines the point on the graph that corresponds to that iteration's throttle percentage and car speed values, which corresponds to the gear that should be imposed on the gear selector mechanism. The coloured curves represent the points of vehicle speed and throttle position at which a gear shift should take place.

Note that when the driver is not pressing the throttle pedal, the percentage of throttle corresponds to zero, and the speed at which the car would shift corresponds to the values directly over the ordinates' axis. Another interesting characteristic of this approach is that, for a constant vehicle speed value, the gear communicated to the gear selector mechanism can be different, depending on the throttle percentage input from the user, hence the characteristic shape of the curves represented. For example, if the vehicle is running at 80 Km/h pressing the throttle pedal at only 20%, the car should be running on a 5^{th} gear, according to the graph from Figure 3.7. If, for some reason, the driver suddenly presses the throttle until 80%, the system interprets this abrupt throttle percentage change as a driver's desire for a fast increment on the car speed, and automatically shifts to a 4^{th} gear, in order to respond to this request.

The value of the AtlasCar vehicle speed is measured with good precision using the rotary encoder attached to the rear right wheel. There is also no problem in using the percentage value of the throttle, because the AtlasCar throttle is already a fully electronic

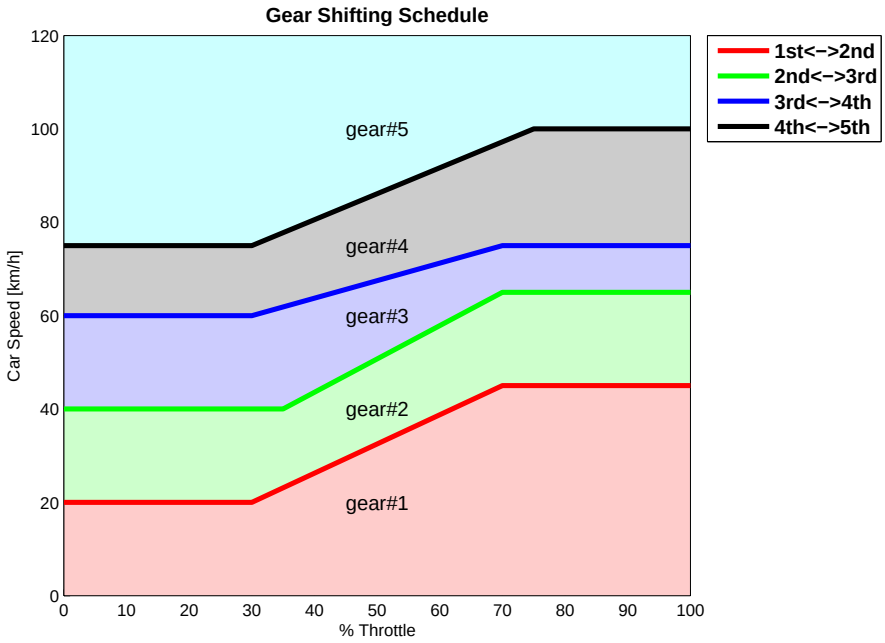


Figure 3.7: Simple Gear Shifting Schedule used on the earlier versions of the HIL Simulator.

system, and this value is used in order to actuate the throttle valve. This method is also able to compute the current gear very fast, because it uses simple linear interpolation to build the gear shifting lines present in the graph.

Although the graph from Figure 3.7 easily illustrates the approach used on the simulator's gear shifting schedule, and allowed it's earlier versions to produce relatively good results, a problem was detected while simulating a particular kind of situation: the neighbourhood of the gear shifting lines. When slight variations on the values of throttle percentage or vehicle speed are made near the gear shifting lines between only a few iterations of the simulator, thus making the calculated point go over and under these lines, there was the risk of provoking gear shifts that were not really necessary. This situation was corrected by creating up-shift lines in different positions than the down-shift ones, which did not happen in the gear shifting schedule described above, and depicted in Figure 3.7, where there is only one line to determine both the up-shift and the down-shift. This new gear shifting schedule is depicted in Figure 3.8.

In order to obtain a better and more realistic response from this kind of shifting schedule, an alternative shifting schedule is proposed. This second shifting schedule, depicted in Figure 3.8, works in a very similar way to the one on Figure 3.8, but is more complex in the way it uses distinct lines to determine the up-shift and the down-shift. As shown in Figure 3.8, the lines with the upward-pointing triangle markers and the continuous line represent the combination of throttle percentage and vehicle speed used as an up-shifting criteria, and the ones with the downward-pointing triangle markers and dashed lines represent that same combination of throttle percentage and vehicle speed used for the down-shifting criteria. The coloured spaces on the graph correspond to throttle percentage and vehicle speed combinations with a single gear solution.

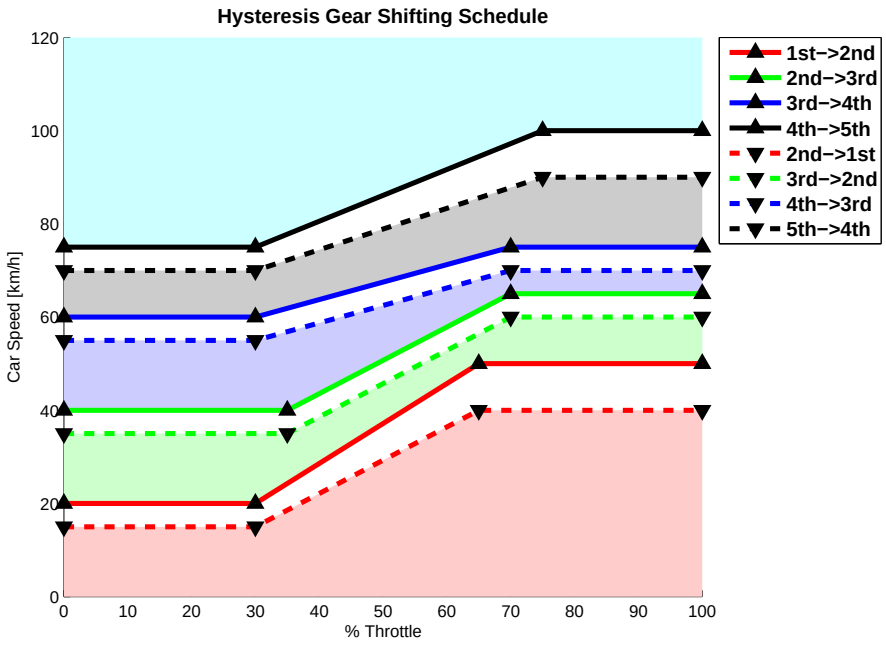


Figure 3.8: Hysteresis Gear Shifting Schedule used on the HIL Simulator, contemplating the different speeds between the process of shifting up and shifting down.

A double gear solution happens in the white spaces, between two lines corresponding to symmetric transitions, like the one existing between the dashed line that represents the down-shift from 2^{nd} gear to the 1^{st} gear and the continuous line representing the up-shift from the 1^{st} gear to the 2^{nd} gear. This double solution problem is solved by passing the current gear into the second version of the function implementing the gear shifting schedule. This function compares the current gear value with the two possible gear solutions. This is handled in two different ways:

- If the current gear value is greater than, or equal to the greatest value of both possible gear solutions, the greatest of these solutions is chosen and imposed to the gear selector mechanism;
- If the current gear value is smaller than, or equal to the smallest value of both possible gear solutions, the smallest of these solutions is chosen and imposed to the gear selector mechanism.

Tendentiously, the driver shifts down at lower car speeds than he shifts up. This is contemplated in this second gear shifting schedule, as the shift down curve is always represented below the shift-up one for gear changes between the same gear numbers, which also adds more realistic behaviour to this model, making it more similar to a gear shift performed by a human driver.

Another consideration about this mode is that, at the time of adapting the system to the AtlasCar vehicle itself, all the point values used to build the curves depicted in Figure 3.8 must be conveniently calibrated, in order to obtain the best vehicle response and behaviour.

Chapter 4

Experimental Results

Several laboratory tests were performed on the gear selector mechanism, to the motor controller printed circuit board, to the *Arduino* code itself and to the ROS Node that communicates with the gear selector mechanism's firmware. This experimental results Chapter is divided in two distinct Sections: the manual mode testing, and the Hardware-in-the-Loop simulation.

4.1 Manual Mode Testing

A variety of tests, performed under the manual mode, were taken on the gear selector system to evaluate the behaviour and performance of two distinct parts:

- The quality of the motor controller PCB;
- The robustness and effectiveness of the firmware manual mode;

4.1.1 Motor Controller PCB

After fully soldering all the components on the motor controller printed circuit board, some tests were performed in order to determine possible problems. The first test performed was a continuity test on all the printed circuit pathways along with a careful visual inspection process, to detect possible undesired connections due to errors during the printing process.

One of the biggest concerns associated with the integrity of the control board was the correct soldering of the H-bridges integrated circuits, the VNH3SP30-E depicted in Figure 2.3. The only components that were not manually soldered into the PCB were the two H-bridges and the two protective N-MOSFETS. These components were soldered using a reflow oven, due to their surface mount nature. The three "Heat Slug" connectors, placed directly under the H-bridges' integrated circuits, are particularly prone to possible soldering failure. In order to check the quality of the soldering process, the pins of both H-bridges were also tested using the multimeter in continuity mode, achieving the desired quality results.

Another major concern associated with the printed circuit board design was the electric current values consumed by the DC motors while in motion. In order to evaluate the current passing through the circuit, a laboratory power supply was used to feed the power circuit. The test consisted on repetitively making the gear selector mechanism

run through all the six possible gear positions in sequential mode, first in the upward direction, and then in the downward direction. The first part of the test was performed without any load applied on the gear selector mechanism, and during the second part of the test, a force is manually applied to the mechanism central ring.

While testing the unloaded mechanical system situation, the maximum electric current peak observed in the current gauge is approximately 1.5 A. The equivalent value observed for the loaded mechanism situation was never greater than 5 A, and even this value only occurs during the motors start-up or stop. The power supply used for the test is known to limit current values above 5.5 A, and the DC motor control system never activated the current limiting circuit during the tests performed.

Another parameter that deserved some attention during the previous test was the temperature achieved by the H-bridges' integrated circuit during the normal operation of the system. According to the VNH3SP30-E datasheet, the absolute maximum temperature that the integrated circuit package type can withstand is 150° C, which also corresponds to the thermal shut-down protection minimum temperature. Although one of the H-bridges, namely the one responsible for the vertical paths, presented higher temperatures than the other, such high temperature values were never achieved, even after extensive system testing. A termopar, connected to the respective connector on the multimeter, was used to monitor the H-bridges' temperature, showing that the value never ascended the 55° Celsius, even after several minutes of continuous operation.

4.1.2 Manual Mode Programming Tests

The manual mode consists of a sequential gear shifting process, as described in Chapter 2. Due to the high responsibility role of the gear selector mechanism in the AtlasCar, both the user interface with the *Arduino*, based on the adapted Renault Mégane audio pad, and the sequential behaviour of the manual mode code needed to be further tested.

The gear pad buttons are already equipped with the necessary pull down resistors, and the input pins on the *Arduino* do not present any unwanted fluctuations. This is the reason why no pull-down resistors are placed in the printed circuit board design on these digital input pins.

Another recurring problem in this kind of button pressing systems is the button debouncing. This problem is due to the high frequency voltage peaks that occur prior to the signal stabilization, at the moment of pressing a button. There are two different ways of solving this problem:

- Hardware de-bounce, using passive, low pass filtering;
- Software de-bounce, using timers or delays.

The up and down-shift buttons do not require any de-bounce methods, because when the user presses them while in manual mode, the system automatically starts moving to the next position, and during this movement process any pulse variation on the button, being it intentional or not, is ignored. The same does not happen with the button used to switch between the manual and the automatic modes. If no de-bounce is executed on this button, the modes can switch very quickly between them, and the user can easily loose track of the mode currently selected. The de-bounce method selected to solve this problem was the software de-bounce, applied directly on the *Arduino* firmware code.

Another situation tested during the manual mode was the communication failure with the PC. The Ethernet connection was deliberately interrupted while the system was both in a stationary position and while moving from one position to another, to ensure that a communication failure did not compromise the normal working of the gear selector mechanism. This fact was confirmed: the communication function running on the *Arduino* is not blocking, and the Ethernet connection can be interrupted at any time, without any damage inflicted to the gear selector mechanism, or the AtlasCar gear system.

4.2 Automatic Mode Testing

In this Section, the tests performed using the gear selected mechanism while in automatic mode will be described.

4.2.1 ROS Node Testing

The number of tests that can be performed using the ROS node created to communicate with the gear selector mechanism, without the system being installed on the AtlasCar vehicle, is very limited. The only two characteristics that could be tested were the frequency at which the device could communicate with the controlling PC, and the quality of the message protocol defined.

In what concerns to the communication frequency, the use of a dummy message cycle within the ROS Node showed that it communicates with the *Arduino* at a steady frequency of 13 Hz, with small fluctuations of 0.5 Hz, even when the DC motors are moving. Similar to what happened during the Manual Mode tests, the interruption of the communication process did not affect the performance of the *Arduino* programming or the gear selector mechanism. The gear change occurring at the time of the communication interruption is completed successfully.

The messaging system established between the gear selector mechanism and the ROS Node was also tested, by forcing all its predicted situations, producing the expected results, since the Node could interpret the message without any trouble, and publish it in the corresponding ROS Topic.

4.2.2 "Hardware in the Loop" Testing

Using the Hardware-in-the-Loop simulator described in Chapter 3, some tests were performed on the gear selector mechanism control system. In the following Sub-subsections, two possible real situations are simulated and explored, and the behaviour of the mechanism is monitored and described.

4.2.2.1 Test 1 - Normal vehicle driving

This test is performed using the default initial conditions of the simulator. The description of this test is the following:

1. The car is already at 10 km/h, without any pressure on the throttle or brake pedal;
2. 5 seconds later, the throttle pedal is set to 20%;

3. After all the gear changes, the car is accelerated further until it reaches approximately 100 km/h;
4. At second 53, the throttle pedal is set to 0%;
5. At second 57 the brake pedal is fully pressed;
6. The car reaches a stationary position at second 64.

The result of the Hardware-in-the-Loop test described above is depicted in the graphs on Figure 4.1.

The car starts at 10 km/h with the gear selector mechanism set to the first gear. According to the second gear logic method described in Section 3.4, and implemented on the simulator version used, the gear should start changing for a vehicle speed greater than 20 km/h. According to the clutch position graph, this gear shift is verified because the clutch starts moving towards its fully activated position at second 6.2, which is also the time value at which the vehicle speed becomes greater than 20 km/h. At this point, a signal is sent to the *Arduino*, to inform it about the number of the following gear needed, and the gear selector mechanism performs the movements required in order to obey that command. Note that while the gear selector system is moving, the clutch stays pressed, until the simulator receives confirmation from the *Arduino* firmware program confirming that the gear change was successfully performed. Similar behaviour can be observed for the following simulated gear changes, except for the final gear shift, which will be explained later.

Another interesting characteristic that can be observed directly on the clutch percentage graph, is the different gear shifting execution times. Although the time represented on the graphs is an estimated value, the time that the clutch stays activated is notoriously different for certain gear shifts. One example of this behaviour can be seen between the 1^{st} to 2^{nd} gear shift and the 2^{nd} to 3^{rd} gear shift. This is due to the more complex trajectory that needs to be travelled by the gear lever while moving from the second to the third gear. This trajectory involves moving both DC motors, contrarily to what happens while moving from the first gear to the second, which corresponds to a simple vertical movement. This situation had already been foreseen at the time of writing the *Arduino* firmware program, and is now confirmed with experimental results.

At the second 58, the brake is fully activated and a drastic deceleration is imposed to the vehicle until it reaches a complete standstill at second 64. During this deceleration process, multiple gear numbers are consecutively set on the message sent to the gear selector mechanism. Due to the firmware programming, as soon as the motors start moving the gear shift process can not be interrupted, so once a gear number is set on the message going to the *Arduino*, the mechanism will start executing the required gear change, and will ignore any incoming message updates. This will result on the gear selector mechanism executing some gear shifts that were set from a previous message, while new messages with different requested gears are being sent. However, the simulator is prepared to deal with this situation, and determines if the gear currently read from the device corresponds to the gear sent on the last message, prior to start the de-clutching process. This is the reason why the graphic representation shows a direct change from the fifth gear to the Neutral, while the real behaviour of the gear selector mechanism was a change made from the fifth gear to the fourth, and only then a change from fourth gear to the neutral.

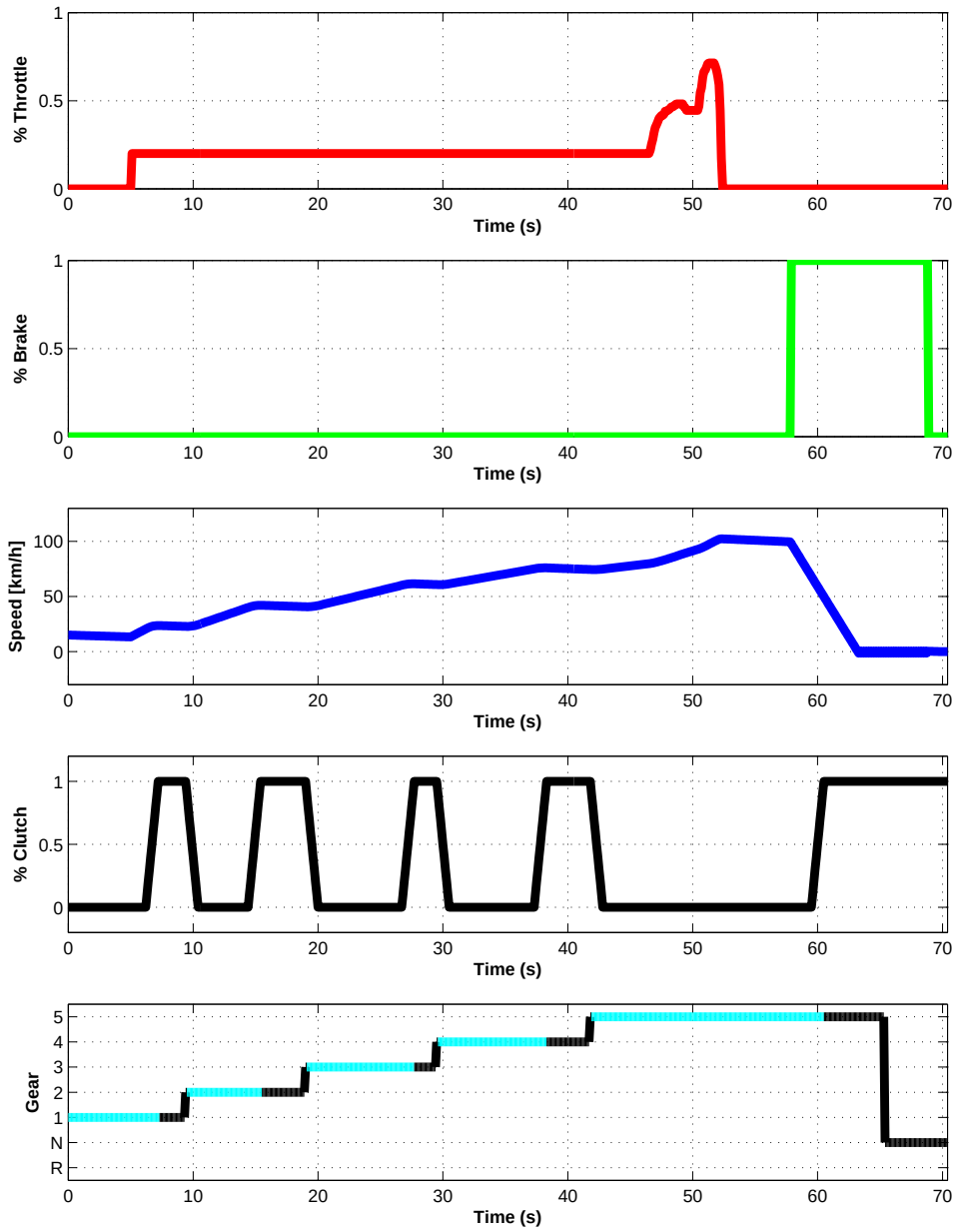


Figure 4.1: Graphs representative of Test number 1. From the top to the bottom, the graphs represent the Throttle Percentage, the Brake Percentage, the Vehicle Speed, the Clutch Percentage and the Gear Numbers. The black line on the "Gear" graph represents the moments at which the gearbox is disengaged from the engine.

The behaviour of the system while the car is stopped can be watched from the second 63 until the end. This behaviour is comprised of two distinct measures, to prevent the engine from stalling:

- The clutch is pressed;
- The Neutral is set on the gear selector mechanism device.

4.2.2.2 Test 2 - Start-up situation

Another situation tested using the Hardware-in-the-Loop simulator was the start-up situation, which consists of the interruption of the vehicle's stationary state. This test is depicted in Figure 4.2. The stationary state can evolve in two distinct movement directions, since the vehicle can move both in the forward direction by inserting the first gear, or in the backward direction, by inserting a reverse.

This test is performed using all the default initial conditions of the simulator, except for the initial speed, that is set to zero. The description of this test is the following:

1. The simulation starts with the car at 0 km/h, and the neutral inserted. Although the clutch pedal starts on a deactivated position, the simulator starts pressing the pedal automatically;
2. At second 4, the throttle pedal is pressed until it reaches more than 30% of its total stroke;
3. The car is randomly accelerated and some gear changes take place;
4. At second 36, the car is made to move in the backward direction;
5. At second 45, the brake is activated;
6. At second 49 the car is immobilized again.

At the beginning of the simulation, the neutral was already selected on the gear selector mechanism, but the clutch was not pressed. Pressing the clutch is always the first operation executed by the simulator. Soon after detecting more than 20% of throttle pressed, the car starts moving in the forward direction. Note that for this first start-up situation, the "reverse" push button on the graphical user interface is not pressed.

While the vehicle is running in normal operation, from second 7 to second 21, the gear changes happen sequentially. At second 21 the first brake interval on the simulation starts taking place, until the system reached the stationary state at second 27. Similarly to what happened during test number 1, the gear selector mechanism was sent to a neutral gear, and the clutch is completely actuated, for security reasons and to keep the engine from stalling.

At this point of the simulation, and in order to change the direction of the vehicle movement, the "Reverse" button is pressed. As soon as the throttle pedal is pressed more than 20%, the gear selector mechanism is set to the reverse gear. The clutch is then deactivated and the vehicle starts moving in reverse.

To end the simulation, at second 45 the brake pedal is pressed again, and as the vehicle looses speed the standard safety measure are taken, namely the complete pressing of the clutch pedal followed by the imposition of a neutral gear on the gear selector mechanism.

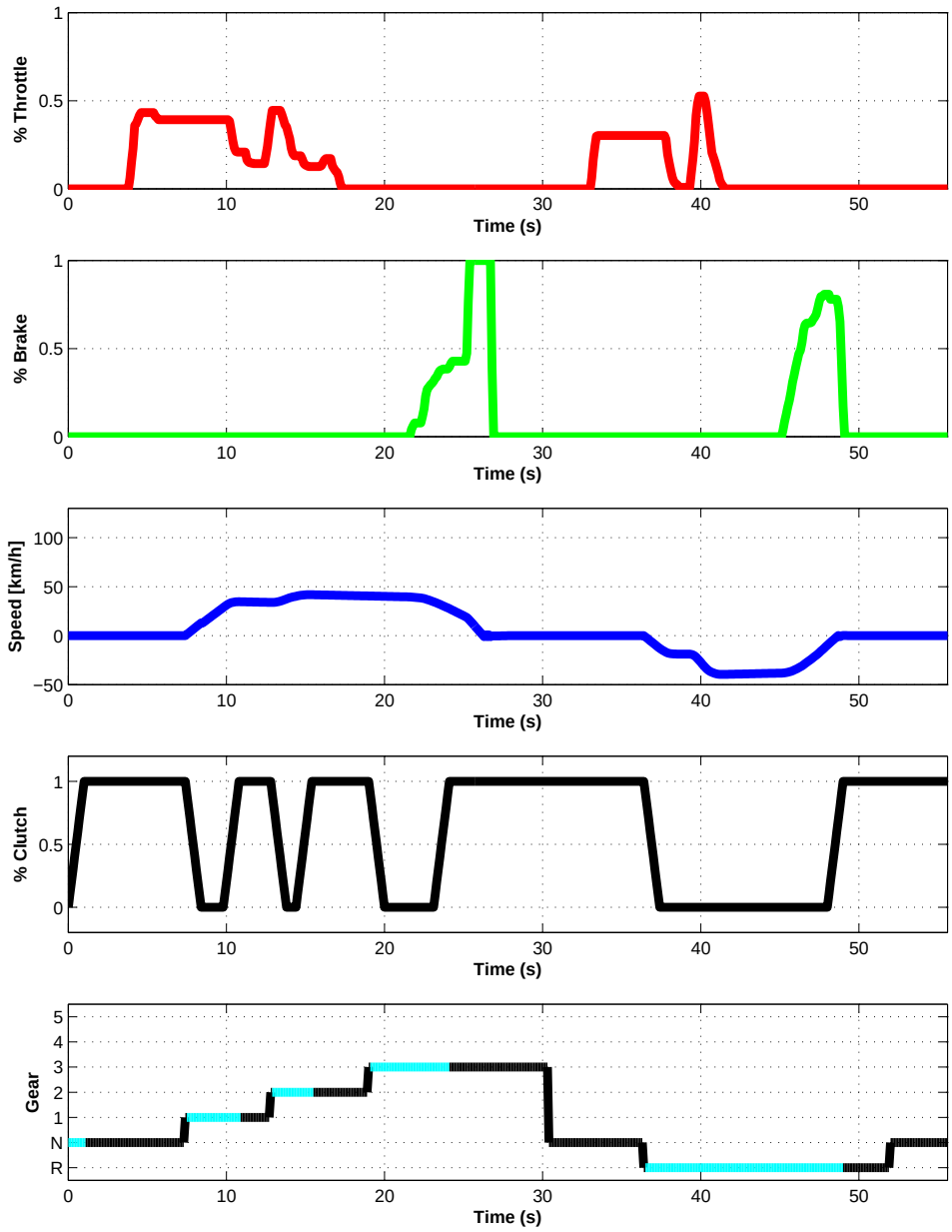


Figure 4.2: Graphs representative of Test number 2. From the top to the bottom, the graphs represent the Throttle Percentage, the Brake Percentage, the Vehicle Speed, the Clutch Percentage and the Gear Numbers. The black line on the "Gear" graph represents the moments at which the gearbox is disengaged from the engine.

4.2.2.3 Test 3 - Steep road

The last situation explored and described in this work is the behaviour of the gear selector mechanism when the vehicle is climbing a steep road using only its inertia. The test is depicted in Figure 4.3. During this test, the user does not accelerate, and the only forces acting on the vehicle are the gravity force, the rolling resistance force and the drag force. This is the reason why the first graph depicted is not the throttle percentage, like the previous tests. The only input from the driving pedals is the brake action, at the end of the test.

Similarly to what happened in the previous test, all the default initial conditions of the simulator are kept, except for the initial speed, that is set to 120 km/h. The description of this test is the following:

1. The simulation starts with the car at 120 km/h, and the fifth gear inserted. At this point the vehicle is also on a flat road, as depicted in the first graph;
2. The vehicle looses some speed for approximately 15 seconds;
3. At second 16, the angle of the street suddenly changes from 0° to 6°, which means that the vehicle starts climbing the road, and consequently, loosing speed;
4. At second 43, the car stops and starts moving backwards;
5. At second 51, the user finally presses the brake pedal completely, to stop the vehicle;
6. At second 54 the car is immobilized, and remains like this until the end of the test.

At the beginning of this simulation, the vehicle was already running at 120 km/h on a flat ground, the fifth gear was already selected on the gear selector mechanism and the clutch was not activated. During the first fifteen seconds of the simulation, the vehicle looses some speed. Because of the high speed of the car, this loss of speed is mainly due to the drag force, with some contribution of the rolling resistance force.

At second 16, the road angle suddenly changes from the flat ground, or 0°, to 6°. This sudden change makes the vehicle loose speed in a fast way, because the x axis gravity force component increases. At second 26 the speed of the vehicle justifies the insertion of a fourth gear. Although the fourth gear is inserted on the gear selector mechanism, at second 30, the speed is evaluated again, prior from releasing the clutch, and another gear shift occurs, and a third gear is inserted soon after second 32.

The car continues loosing speed and at second 34.5 a new gear shift is initiated from the third gear to the second gear, and completed at second 38.8. The car continues running on a second gear, until the speed is too low, and the default process of activating the clutch and inserting the Neutral. Note that, due to the Dijkstra's algorithm, the mechanism was able to skip the first gear.

At second 43 the car stops completely, and since the gravity force continues acting, the vehicle starts moving backwards. Note that the simulator does not insert the Reverse in this situation, since no input from the user is present. At second 51 the user finally presses the brake pedal completely, in order to stop the vehicle, which happens soon after second 54. The Neutral remains inserted in the gear selector mechanism, and the clutch remains pressed.

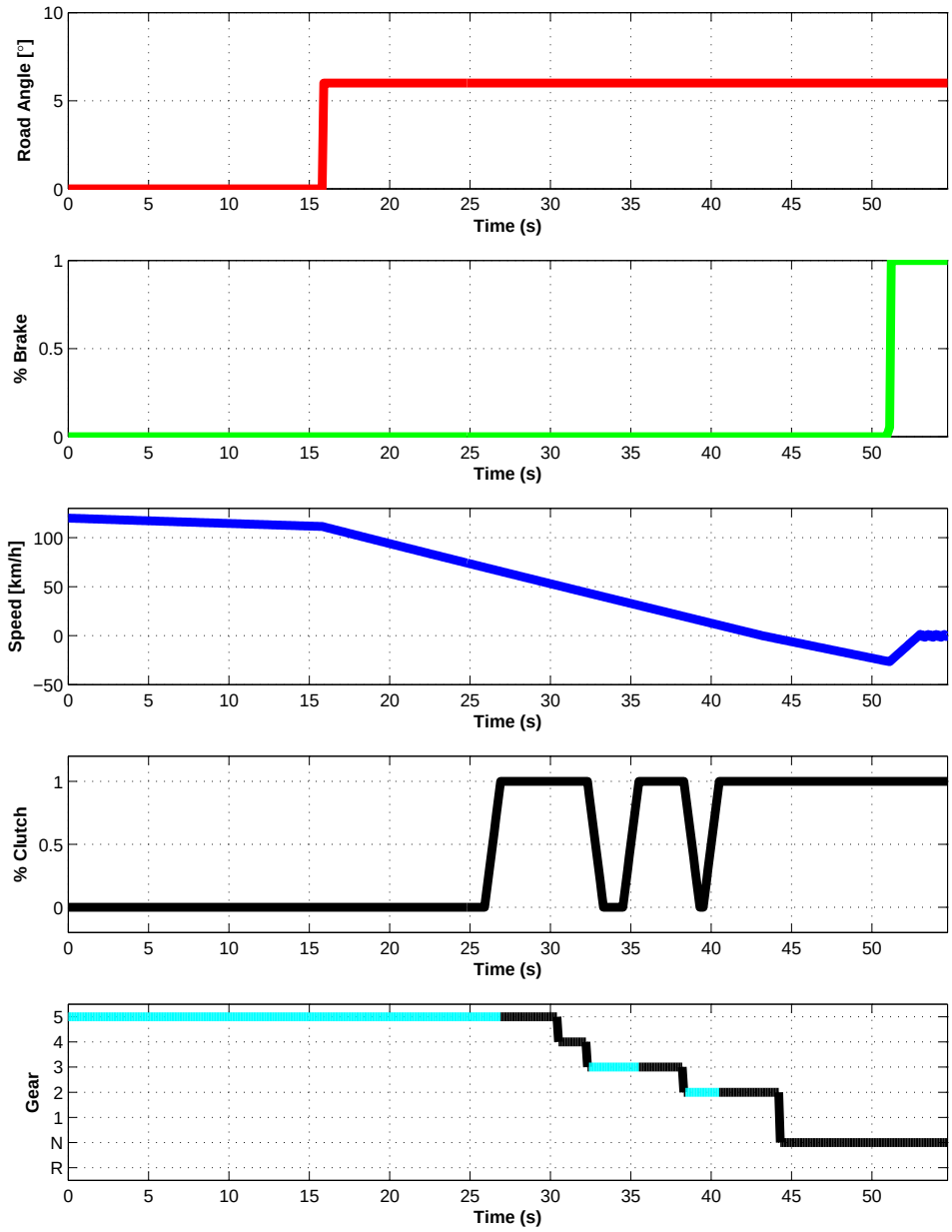


Figure 4.3: Graphs representative of Test number 3. From the top to the bottom, the graphs represent the Road Angle, the Brake Percentage, the Vehicle Speed, the Clutch Percentage and the Gear Numbers. The black line on the "Gear" graph represents the moments at which the gearbox is disengaged from the engine.

Chapter 5

Conclusions

During the present Chapter, a final and conclusive analysis of the developed hardware and software solutions proposed and executed during this work will be made. The objectives that were fully accomplished will be presented, and the ones which did not match the initial expectations will also be exposed. Some proposals about possible future tasks and new work fronts will also be proposed.

5.1 Conclusions

One of the main objectives of this work was to project and build a robust and reliable control board to drive both DC motors, as well as performing all the auxiliary tasks, like the potentiometer's reading system, the digital decoders, the 7-segment display and the manual or automatic selector switch. This objective has been fully accomplished, since the circuit has proven to be robust enough to withstand the laboratory tests, along with all the demanding firmware code development process. Note that only one printed circuit board version was manufactured and used throughout the whole development process, which is precisely the same one that is still driving the gear selector mechanism on the present day. There were no amendments made on the original design, whose drawing can be observed in the Annexes, and no electronic components had to be replaced due to damage during the execution of this work.

The *Arduino*-based control system also turned out to be a good option. Although its frequency limitations, and the higher rigidity of the system when compared to the Microchip PIC-based systems for example. The easy code development functionalities and the huge community developing new and more effective libraries, provided by the *Arduino*, have proven to be very helpful during the execution of this project.

In what concerns to the firmware programming itself, the Dijkstra's algorithm was successfully implemented on the *Arduino* firmware with the expected results, and the initial objective of a less rigid, non-sequential gear selector system could be reached. This firmware programming showed to be suited for the mechanism to successfully reach all the necessary gear positions.

The communication process between the firmware and the control software, which uses an Ethernet cable and the TCP/IP protocol, has proven to be very stable as initially expected. The messaging system used both to control the gear selector mechanism and to read its current state is very complete, and this characteristic has been achieved without the need do sacrifice the simplicity of the message sent and received by the device.

Other software modules were also developed and tested to aid the use of the gear selector mechanism. The calibration software created has shown to be particularly useful, allowing the recalibration of the system to new conditions without editing the *Arduino* code directly by hand. The existence of such calibration software is important within projects involving many people, like the AtlasCar project, since other people in the project do not need to know or understand everything about the gear selector mechanism's programming to use it and calibrate it.

The ROS Node, also developed during this work, allows the communication with the gear selector mechanism using the mew ROS architecture, which is currently the one present in the AtlasCar control PC. This Node was created to facilitate the future mounting of the device in the Atlascar vehicle, and its integration with the already developed software.

The simplified Hardware-in-the-Loop simulator, written in Matlab, also allowed the further testing of the gear selector mechanism. This simulation was useful to anticipate some particular situations or problems that may occur when the mechanism is mounted on the AtlasCar vehicle, and correct them.

The AtlasCar gear selector mechanism's power transmission system, using the belts and pulleys, may not be the most adequate to mount it directly on the AtlasCar vehicle. Although the DC motors' planetary gearboxes easily develop the necessary torque in order to move the gear lever, the belts may have not enough grip to drive the lever to its correct place. The original idea of using belts to transmit power was thought to allow the system to skip pulley teeth if the mechanism jammed, or if the controller failed for some reason. The problem is that the belts may fail when exposed to smaller tensions than the ones required to move the gear lever, which was noticed in the laboratory. The use of a more consistent power transmission system, like a chain and sprocket, would probably solve the problem.

5.2 Future Work

The most important future tasks that still need to be performed in the AtlasCar are listed below. These tasks would guarantee the proper operation of the gear selector mechanism on the AtlasCar, and would create the necessary conditions for the development of new and more daring manoeuvres:

- Mounting of the system in the AtlasCar vehicle's gear lever, and correction of possible problems that will inevitably appear during that process. The replacement of the belt and pulley system by an alternative system, like the chain and sprocket referred in the previous Section, may be necessary;

- Possible corrections and modifications on the code that controls the DC motors, to ensure the correct positioning of the gear lever. These corrections can only be executed after mounting the gear selector mechanism on the vehicle;

- Implementation of a closed loop speed control on the vehicle, once the mechanism is working properly;

- Implementation of certain low-level manoeuvres involving the gearbox, the clutch and the throttle, needed for a particular set of precision driving situations. Some

examples are the parking manoeuvre or the start of movement in a steep slope, where the biting point is needed to prevent the vehicle from rolling backwards.

6 References

[1] *ATLAS project.* (May 2012) Obtained from: atlas.web.ua.pt

[2] Tiago Rocha. *Piloto Automático para Controlo e Manobras de Navegação AtlasCar.* Master's thesis, Universidade de Aveiro, 2011.

[3] power train. [Art]. Encyclopædia Britannica Online. (June 2012) Obtained from: http://www.britannica.com

[4] *Oscar Gearbox.* (June 2012) Obtained from: oscarsautomatictransmission.com

[5] Robert Bosch. *Automotive Handbook (5th ed.).* SAE Automotive Society of Engineers, 2003.

[6] Marc Ross. *Fuel efficiency and the physics of automobiles.* Contemporary Physics, 1997.

[7] *About Cars.* (May 2012) Obtained from: cars.about.com

[8] Vitor B. Sabbagh, Elias J. R. Freitas, Guilherme M. M. Castro, Michelle M. Santos,Maurício F. Baleeiro, Tiago M. da Silva, Paulo Iscold, Leonardo A. B. Torres and Guilherme A. S. Pereira. *Desenvolvimento de um Sistema de Controle para um Carro de Passeio Autônomo.* XVIII Congresso Brasileiro de Automática, 2010.

[9] Michael A. Kluger and Denis M. Long. *An Overview of Current Automatic, Manual and Continuously Variable Transmission Efficiencies and Their Projected Future Improvements.* SAE Automotive Society of Engineers, 1999.

[10] *Siemens.* (May 2012) Obtained from: www.automation.siemens.com

[11] ST Microelectronics. VNH3SP30-E datasheet. Datasheet. ST Microelectronics.

[12] NXP Semiconductors. 4555B 1-of-4 decoder/demultiplexer. Datasheet. NXP Semiconductors.

[13] Vishay Spectrol. 7/8"(22.2 mm) Multiturn Wirewound 533: 3 Turns. Datasheet. Vishay Spectrol.

[14] *RS Components.* (June 2012) Obtained from: pt.rs-online.com

[15] Vishay Telefunken. Standard 7-Segment Display 13 mm. Datasheet. Vishay Telefunken.

[16] Fairchild Semiconductor. DM7447A BCD to 7-Segment Decoders/Drivers. Datasheet. Fairchild Semiconductor.

[17] *Arduino.* (May 2012) Obtained from: arduino.cc

[18] A. Krause. Foundations of GTK+ Development, 1st ed. 2007. Corr. 2nd printing ed. Apress, 2007.

[19] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler and Andrew Ng. *ROS: an open-source Robot Operating System.* ICRA Workshop on Open Source Software, 2009.

[20] Alex Serrarens, Marc Dassen and Maarten Steinbuch. Simulation and Control of an Automotive Dry Clutch. American Control Conference, 2004.

[21] Ian Millington. Game Physics Engine Development - How to build a commercial-grade physics engine for your game, Second Edition. Morgan Kaufmann Publishers, 2010.

[22] G. K. Batchelor. *An Introduction to Fluid Dynamics.* Cambridge University Press, 2000.

[23] National Academy of Sciences. Transportation Research Board Tires and Passenger Vehicle Fuel Economy: Informing Consumers, Improving Performance - Special Report 286. National Academy of Sciences, 2006.

[24] S. K. Clark and R. N. Dodge *A Handbook for the Rolling Resistance of Pneumatic Tires.* Industrial Development Division - Institute of Science and Technology - University of Michigan, 1979.

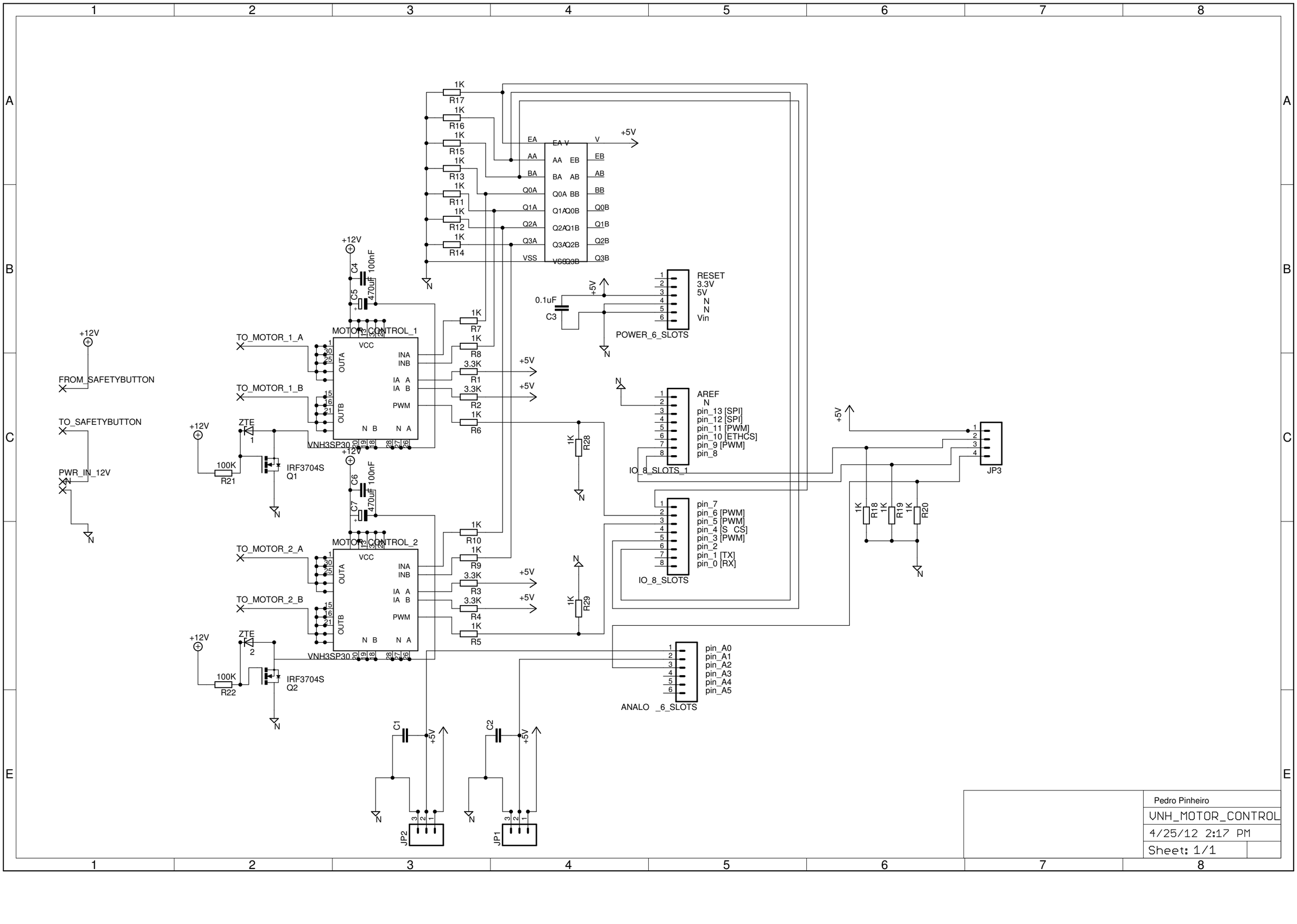
[25] Juan R. Pimentel and Michael T. Loeffler. *A Real-Time Engine Simulator Using Multiple Microcomputers.* IEEE Transactions On Industrial Electronics, 1983.

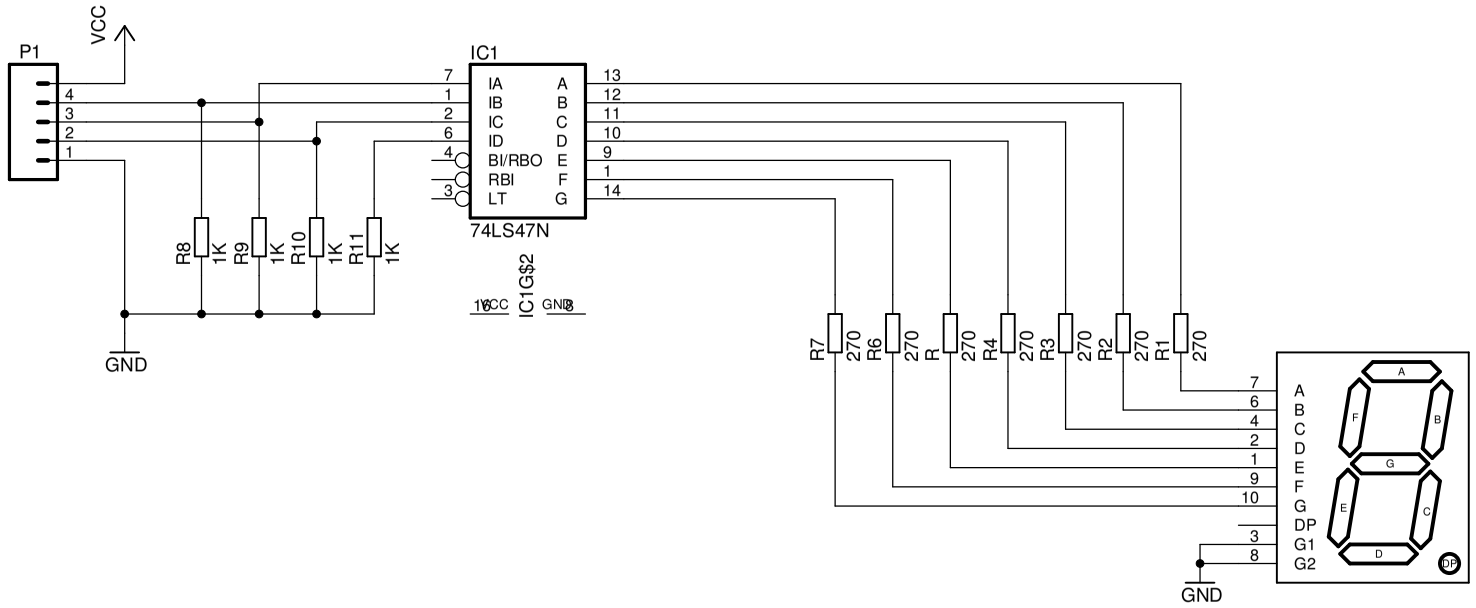
[26] ALLPAR. http://www.allpar.com/ Retrieved on May, 2012.

[27] (May 2012) Obtained from: http://www.mathworks.com/products/stateflow/demos.html?file=/products/demos/shipping/simulink/sldemo_autotrans.html

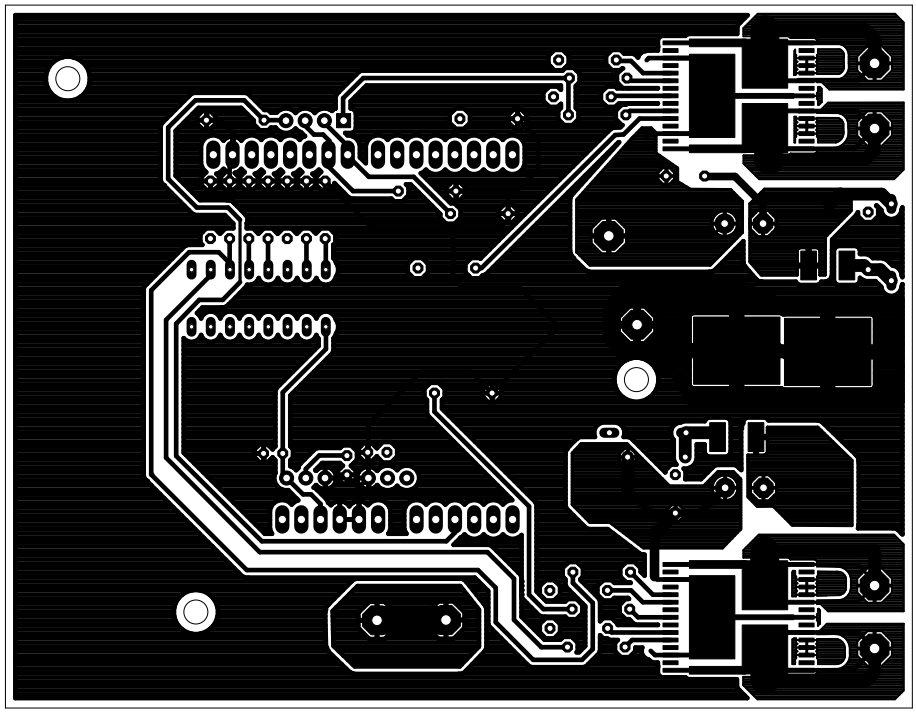
[28] Daekyun Kim, Huei Peng, Shushan Bai and Joel M. Maguire. *Control of Integrated Powertrain With Electronic Throttle and Automatic Transmission.* IEEE Transactions on Control Systems Technology, 2007.

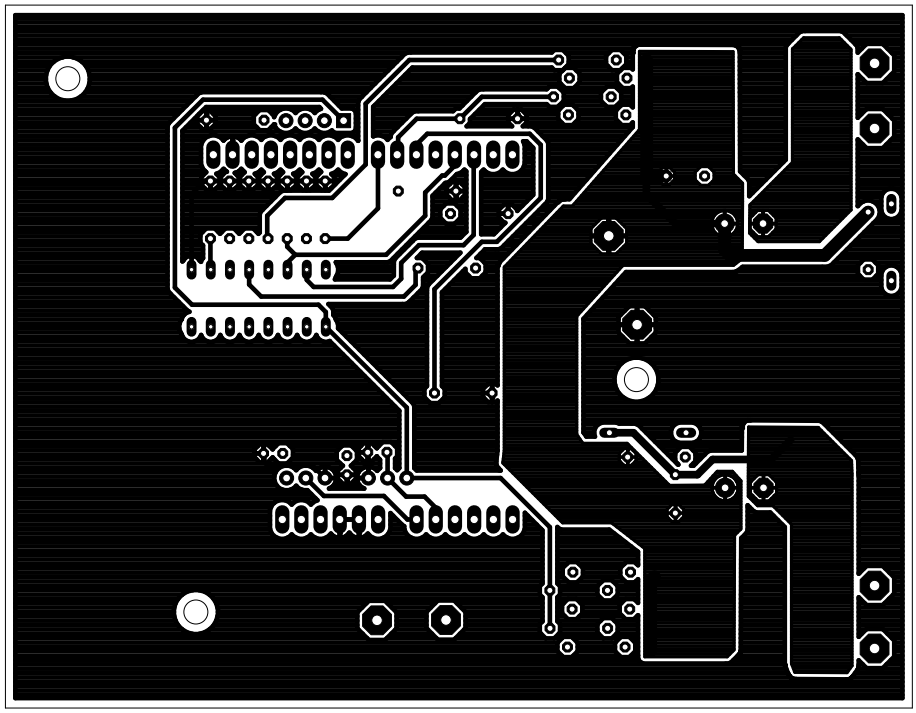
7 Annexes

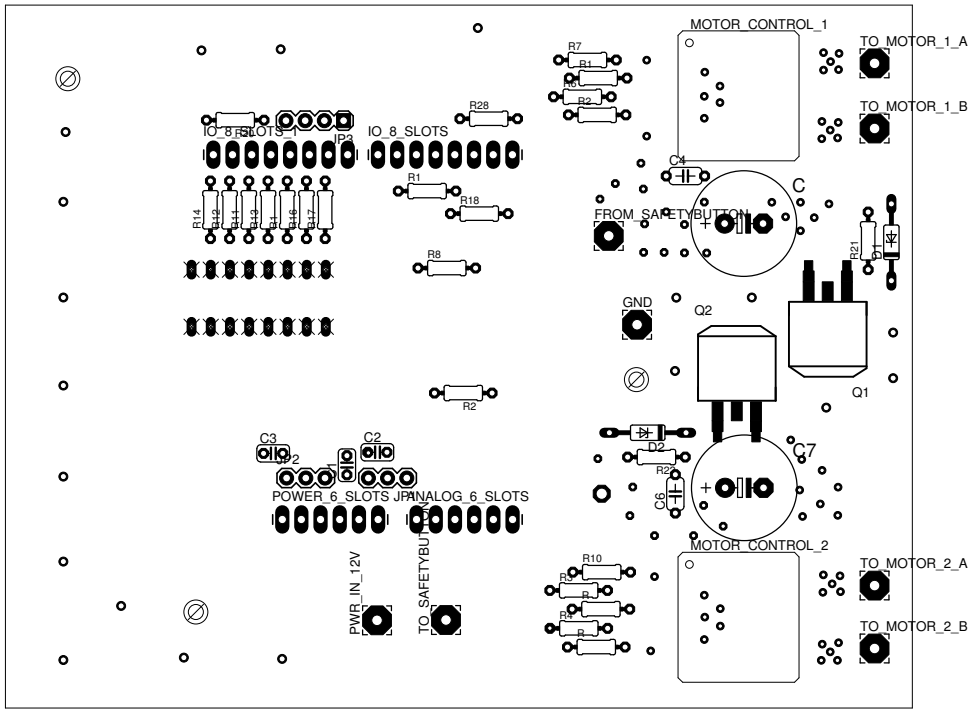

1K R17
1K R16
1K R15
1K R13
1K R11
1K R12
1K R14
EA
AA
BA
Q0A
Q1A
Q2A
Q3A
VSS
EA V
AA EB
BA AB
Q0A BB
Q1A Q0B
Q2A Q1B
Q3A Q2B
VSS Q3B
V
EB
AB
BB
Q0B
Q1B
Q2B
Q3B
+5V
N
+12V
C4 100nF
C5 470uF
MOTOR_CONTROL_1
VCC
OUTA
OUTB
INA
INB
IA A
IA B
PWM
N B
N A
VNH3SP30
1K R7
1K R8
3.3K R1
3.3K R2
1K R6
TO_MOTOR_1_A
TO_MOTOR_1_B
FROM_SAFETYBUTTON
TO_SAFETYBUTTON
PWR_IN_12V
ZTE
1
100K R21
IRF3704S Q1
C6 100nF
C7 470uF
MOTOR_CONTROL_2
TO_MOTOR_2_A
TO_MOTOR_2_B
1K R10
1K R9
3.3K R3
3.3K R4
1K R5
ZTE
2
100K R22
IRF3704S Q2
0.1uF C3
RESET
3.3V
5V
N
N
Vin
POWER_6_SLOTS
AREF
N
pin_13 [SPI]
pin_12 [SPI]
pin_11 [PWM]
pin_10 [ETHCS]
pin_9 [PWM]
pin_8
IO_8_SLOTS_1
1K R28
1K R29
pin_7
pin_6 [PWM]
pin_5 [PWM]
pin_4 [S_CS]
pin_3 [PWM]
pin_2
pin_1 [TX]
pin_0 [RX]
IO_8_SLOTS
JP3
1K R18
1K R19
1K R20
pin_A0
pin_A1
pin_A2
pin_A3
pin_A4
pin_A5
ANALO _6_SLOTS
C1
C2
JP2
JP1
Pedro Pinheiro
VNH_MOTOR_CONTROL
4/25/12 2:17 PM
Sheet: 1/1



7_seg	
6/29/12 11:35 AM	
Sheet: 1/1	






MOTOR_CONTROL_1
TO_MOTOR_1_A
TO_MOTOR_1_B
R7
R1
R6
R2
R28
IO_8_SLOTS_1
JP3
IO_8_SLOTS
C4
C
R1
R18
R14
R12
R11
R13
R16
R17
FROM_SAFETYBUTTON
R21
D1
R8
GND
Q2
Q1
R2
C3
C2
JP2
D2
R22
C7
C6
POWER_6_SLOTS JP1
ANALOG_6_SLOTS
MOTOR_CONTROL_2
R10
TO_MOTOR_2_A
TO_MOTOR_2_B
PWR_IN_12V
TO_SAFETYBUTTON