Benemérita Universidad Autónoma de Puebla

Faultad de Ciencias de la Electrónica

Maestría en Ingeniería Electrónica, opción instrumentación electrónica

Design and Implementation of a Re-configurable Driving Simulator Prototype.

Author:
Eng. Iván Cañedo Farfán

Advisor: Dr. Roberto Carlos Ambrosio Lázaro

Co-Advisor: Dr. José Fermi Guerrero Castellanos

December, 2021

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program "Maestría en Ingeniería Electrónica, opción instrumentación electronica".

Author: Eng. Iván Cañedo Farfán, ivan.canedo@alumno.buap.mx

Advisor: Dr. Roberto Carlos Ambrosio Lázaro

Co-Advisor: Dr. José Fermi Guerrero Castellanos

Benemérita Universidad Autónoma de Puebla
Faultad de Ciencias de la Electrónica
Av. San Claudio y 18 Sur Edif. FCE1 Col. San Manuel, Ciudad Universitaria, Puebla, Pue.
CP 72570

December, 2021

To my family for all the emotional support.
To my dogs for the unconditional love.
To my supervisors for their invaluable contribution on this work.
To Alexandra Elbakyan for her contribution to science.
To the content creators that helped me retain my sanity.
To my PC gaming master race for hard carrying this project.
To the void...

Acknowledgements

- I wish to extend my special thanks to CONACYT for the financial support provided during the development of this project.
- I wish to show my appreciation to the Faculty of Electronic Sciences (FCE) of the Benemérita Universidad Autónoma de Puebla for the support and guidance throughout this project.
- I would like to express my gratitude to my primary supervisor, Roberto Carlos Ambrosio Lázaro, who guided me throughout this project.
- I would like to express my gratitude to my secondary supervisor, José Fermi Guerrero Castellanos, who guided me throughout this project.

Agradecimientos

- Agradezco a el CONACYT por el apoyo económico brindado durante la realización de este proyecto.
- Deseo mostrar mi agradecimiento a la Facultad de Ciencias de la Electrónica (FCE) de la Benemérita Universidad Autónoma de Puebla por el apoyo y orientación a lo largo de este proyecto.
- Me gustaría expresar mi gratitud a mi asesor, Roberto Carlos Ambrosio Lázaro, que me ha guiado a lo largo de este proyecto.
- Me gustaría expresar mi gratitud a mi co asesor, José Fermi Guerrero Castellanos, que me guió a lo largo de este proyecto.

Abstract

The present work aims to develop an implement a Re-configurable Driving Simulator Prototype. Driving simulators are tools that allow to perform various tasks, like training new drivers or performing tests in a controlled environment that would otherwise be dangerous or expensive to perform on a real vehicle. the concept of a re-configurable driving simulator is centered in the fact that all the components of a driving simulator must be easily interconnected and replaced, along that, new features must be easily added and connected with the existing components.

The implemented system consist of a scale 3DOF parallel manipulator, a 3D simulated environment, a moderator software and input devices (steering wheel, pedals and a button panel). To control the scale platform's position, the moderator software performs a classic filter-based motion cueing algorithm (MCA) to translate the IMU signals from the simulated vehicle to a platform's desired position reference signal. After that a algebraic inverse kinematics calculates the actuators desired position form the platform's position reference. Finally, a embedded control uses a control strategy (ADRC) con control the position of three rotatory electrical actuators. The embedded control control runs on a 32bit ARM MCU programed using the MicroPython interpreter.

The 3D simulated environment is generated using a open source project called CARLA Simulator, this software runs with a TCP server-client architecture and communicates with the moderator software (coded in LabVIEW) thru that protocol. The moderator software communicates with the embedded control and the input devices thru a USB serial port.

To validate the system's functional behaviour, a simulation-based approach was performed. Testing and validation show a behaviour that matches the results found on the analyzed literature and the simulation results. All simulations were performed using MATLAB/Simulink

Keywords: Driving simulator, MCA, ADRC, ARM, Micropython, CARLA Simulator, reconfigurable, inverse kinematics, 3DOF PM, 3-RRS PM, forward kinematics, gear-motor.

Resumen

El presente trabajo pretende desarrollar e implementar un prototipo de simulador de manejo reconfigurable. Los simuladores de manejo son herramientas que permiten realizar diversas tareas, como la formación de nuevos conductores o la realización de pruebas en un entorno controlado que, de otro modo, serían peligrosas o costosas de realizar en un vehículo real. El concepto de un simulador de manejo reconfigurable se centra en el hecho de que todos los componentes de un simulador de manejo deben ser fácilmente interconectados y reemplazados, junto con eso, las nuevas características deben ser fácilmente añadidas y conectadas con los componentes existentes.

El sistema implementado consiste en un manipulador paralelo de 3DOF a escala, un entorno simulado en 3D, un software moderador y dispositivos de entrada (volante, pedales y una botonera). Para controlar la posición de la plataforma a escala, el software moderador realiza un algoritmo clásico de estimulación de movimiento (MCA) basado en filtros para traducir las señales de la IMU del vehículo simulado a una señal de referencia de posición deseada para la plataforma. A continuación, la cinemática inversa algebraica calcula la posición deseada de los actuadores a partir de la posición de referencia de la plataforma. Finalmente, un control embebido utiliza una estrategia de control (ADRC) para controlar la posición de tres actuadores eléctricos rotativos. El control embebido se ejecuta en un MCU ARM de 32 bits programado con el intérprete MicroPython.

El entorno simulado en 3D se genera utilizando un proyecto de código abierto llamado CARLA Simulator, este software se ejecuta con una arquitectura TCP servidor-cliente y se comunica con el software moderador (codificado en LabVIEW) a través de ese protocolo. El software moderador se comunica con el control embebido y los dispositivos de entrada a través de un puerto USB serial.

Para validar el comportamiento funcional del sistema, se realizó con un enfoque basado simulación. Las pruebas y la validación muestran un comportamiento que coincide con los resultados encontrados en la literatura analizada y los resultados de la simulación. Todas las

simulaciones se realizaron con MATLAB/Simulink

Palabras Calve: Simulador de manejo, MCA, ADRC, ARM, Micropython, CARLA Simulator, reconfigurable, cinemática inversa, 3DOF PM, 3-RRS PM, motorreductor.

Publications

1. Further results on modeling and control of a 3-DOF platform for driving simulator using rotatory actuators

   - **Type:** Paper
   - **Published on:** Recent Trends in Sustainable Engineering
   - **DOI:** `https://doi.org/10.1007/978-3-030-82064-0_6`
   - **First Place Award** on the Paper contest

2. Implementation of an ADRC for DC motor angular position in an ARM Cortex M4 MCU using MicroPython

   - **Type:** Poster
   - **Published on:** International Conference on Applied Science and Advanced Technology
   - **First Place Award** on the Poster contest

3. Plataforma virtual e interactiva para la conduccion de vehículo basado en modelo 3D de Unreal Engine mediante transmision de datos TCP/IP

   - **Type:** Paper
   - **Published on:** RVP-AI / ROC&C 2021

Contents

List of Figures

Glossary

actuator

A component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. In simple terms, it is a "mover".

automatic control strategy

Automatic control is the application of control theory for regulation of a processes without direct human intervention.

inverse kinematics

the mathematical process of calculating the variable joint parameters needed to place the end of a kinematic chain, such as a robot manipulator in a given position and orientation relative to the start of the chain.

motion platform

A dynamic system that provides motion to a human sitting on it. A parallel robotic system or parallel manipulator.

sate of the vehicle

A of the set of variables that are used to describe the mathematical "state"of a dynamical system. The state of a vehicle is its condition at a specific time

sensory display devices

A collection of devices or apparatus that have the goal of stimulating the human senses. E.g. A speaker simulates the additive system, etc.

tracking

On control theory, tracking refers to the capacity of a controller to follow a reference that varies rapidly over time. The opposite case is a regulation control where the controller has a steady reference.

vehicle control devices

A collection of devices or apparatus that control the behaviour of a vehicle. Typically include steering wheel, brake pedal, throttle pedal, clutch, lever, dashboard, etc.

vestibular organ

The structure composed of the utricle, saccule, and the three semicircular ducts of the membranous labyrinth of the inner ear.

workspace

The workspace of robot manipulator is defined as the set of points that can be reached by its end-effector.

sensor

A device that responds to a physical stimulus (such as heat, light, sound, pressure, magnetism, or a particular motion) and transmits a resulting impulse (as for measurement or operating a control).

List of Acronyms

ADRC	*Active Disturbance Rejection Control*
API	Application Programming Interface
CAD	computer-Aided Design
CNC	Computerized Numerical Control
DC	*Direct Current*
DoF	Degrees of Freedom
ESD	Electrostatic Discharge
ESO	*Extenden State Observer*
FPU	Floating Point Unit
FS	Full Speed
GD&T	Geometrical Dimensioning and Tolerances
GDP	Gross Domestic Product
IC	Integrated Circuit
IMU	Inertial Measurement Unit
IPN	Instituto Politécnico Nacional
MCA	Motion Cueing Algorithm
MCU	Microcontroller Unit
NADS	National Advanced Driving Simulator
OEMs	Original Equipment Manufacturers
OOP	Object Oriented Programming

OTG	On-The-Go
PCB	Printed Circuit Board
PID	Proportional–Integral–Derivative controller
PM	*Parallel Manipulator*
PSU	Power Supply Unit
PWM	Pulse Width Modulation
RRS	Revolute-Revolute-Spherical
SP	Spherical-Prismatic-Spherical
TCP	Transmission Control Protocol
UART	Universal Asynchronous Receiver-Transmitter
USART	Universal Synchronous Asynchronous Receiver-Transmitter
VI	Visual Instrument

Chapter 1

Introduction

During 2018 in Mexico there were 365,281 car accidents, where 337.266 were caused by the diver [1], this is equivalent to the 92% of all the registered accidents during that year, unfortunately there is no simple solution to that problem, nevertheless, driving simulator are tools that are tools widely used all around the world to develop strategies to reduce the number of car accidents [2].

Driving simulators are tools that allow to perform various tasks, like training new drivers or performing tests in a controlled environment that would otherwise be dangerous or expensive to perform on a real vehicle. Therefore, universities and industrial laboratories have been using driving simulators since the early 70s [3].

The principle of operation of a driving simulator is based on the virtual simulation of the driving activity, which is a very complex activity where the driver follows a trajectory using the different the control systems of the vehicle while being guided by a wide variety of data sources that serve as a feedback to the driver. Therefore, a driving simulator needs to provide the driver with the information that would be normally provided in the driving situation in a real-life context [4].

Categorize the driving simulators is complex since many author have proposed different classifications, in [3] the classification is performed depending of their area of use, in his text the author defines the areas of use as entertainment, training and research. On the other hand in [5] the research category is expanded to incorporate the categories of research in

psychology, engineering and medicine. Finally, in [4] the driving simulators are classified by its morphology where there are two main groups, static and dynamic simulators, this thesis will focus on dynamic simulators for research in engineering.

Nowadays, driving simulators have become more accessible since the computational capabilities have increased in power and decrees in price every year. Thus, many configurations have been developed, ranging from static (0-DOF) simulator to systems with 9-DOF or more. The minimum DOF needed to determine the driving simulator's validity is closely related to the type of task that the driving simulator will be used for [5]. Ergo, a compromise between fidelity of the simulation and the minimum DOF shall be made.

Each year thousands of research projects using driving simulators are published, those research projects use driving simulators for a wide range of applications e.g. driver training [6], vehicle evaluation [7], road design and vehicle dynamics simulation [8].

1.1 Justification

The development and manufacture of driving simulators has been divided into two main areas, commercial and industrial application.

Industrial application systems are designed, built and used by transnational companies, by personnel of the same company or by subcontracted companies [9], these simulators are the most complex and sophisticated, however, due to their private nature, the characteristics and technology used in them are an industrial secret and only the results of some researches performed in those systems are published [10, 11, 12].

Commercial systems are usually designed and produced by, mostly, national companies, focused on a local market and with two very well-defined approaches, the first one is the price and the second one is the characteristics of the software used to perform the driving simulation.

In both cases, there is little or no access to the technical specifications of the simulators, nor are they systems that are intended to be modified or improved, therefore, despite being a relatively common product, the knowledge about the design of driving simulators is quite scarce.

In the research field, the most common works on driving simulators are mainly directed towards software, several universities have made their own simulators based on graphic engines such as Unity, Unreal Engine, etc. In their development they have left aside the design of the motion platform and use as input devices commercial components designed for video games [13, 14, 15, 16, 17].

Being that said it's possible to conclude that, in the research field, there are no significant works that propose the design and construction of a dynamic driving simulator that can be known and studied by members of the academy.

In Mexico, the automotive industry generates 3.6% of GDP and ranks sixth in the world as a producer of automobiles [18], despite this, technological development and innovation are not among the country's strengths. Our country is selected by Original Equipment Manufacturers (OEMs) for its low operating costs (labor), high quality workmanship and privileged geographic position. In other words, Mexico's contribution to the added value is low; conducting research and development activities could lead our industry to a higher position in the value chain.

This research seeks to lay the foundations for the development of prototypes based on experimental development, taking advantage of existing knowledge obtained from research and/or practical experience, it will be possible to provide a development platform for the study of digital electronic communication, control systems, virtual instrumentation, mathematical modeling, system dynamics and thus, in the future, be able to use the platform to develop and validate new technologies.

1.2 Objectives

Design and implement a motion platform for a three-degree-of-freedom re-configurable driving simulator prototype.

1.2.1 Specific objectives

- To design the platform using CAD tools.
- To generate and implement the 3 Degrees of Freedom (DoF) control algorithm.
- To perform the signal conditioning for the following devices: steering wheel, pedals and lever.
- To implement the communication between sensors, input devices and *actuator*s.
- To perform system testing.

1.2.2 Methodology

The proposed methodology is represented a block diagram in Fig. 1.1. In this diagram, the relationship between the different activities is shown and the color of each activity represents the specific objective that rules over that specific activity. The proposed methodology is not

linear due to the complex morphology of the system in question. Therefore, the activities have a complex correlation between them, where some of them need other activities to be partially completed before they can be addressed. For that reason, there is not a unique starting point and most of the activities were performed on parallel.

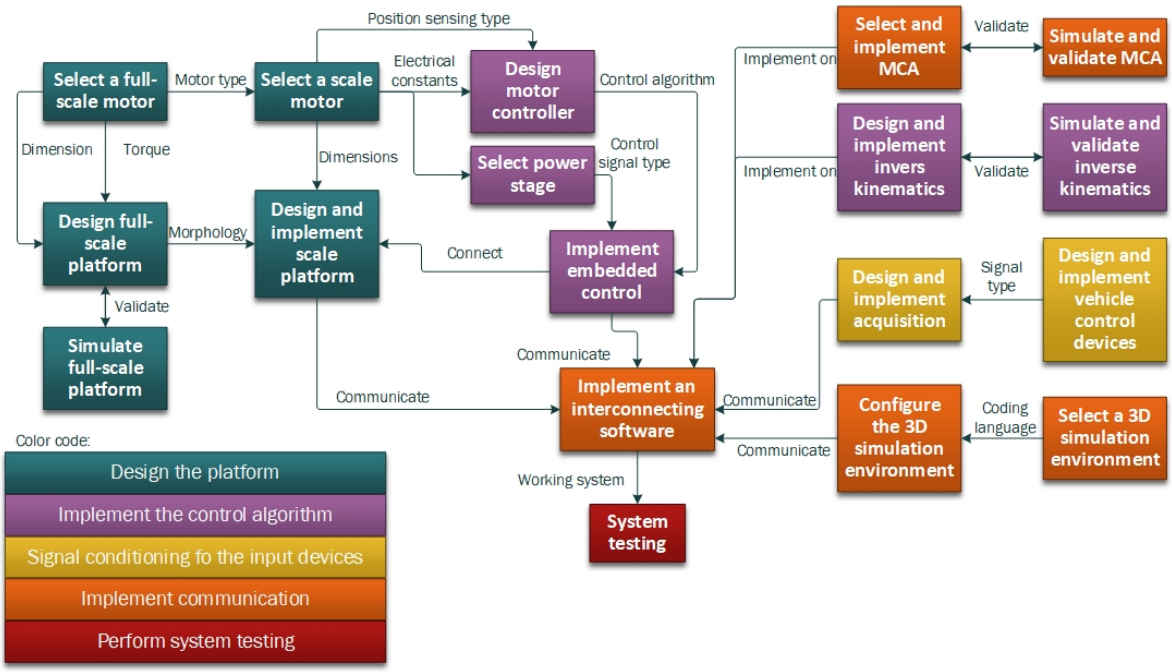

Figure 1.1: Proposed Methodology block diagram

Chapter 2

State of the Art

In this chapter the previous works on the field of driving simulators will be addressed alongside with the theoretical concepts that will be used in this research. First, a brief history of the driving simulators will be documented concluding with the actual state of the driving simulators, next, the need for motion in driving simulator will be addressed alongside with the different morphologies found in motion-based driving simulators.

After concluding the section about the driving simulators per se, the state of the art of the different theoretical concepts used in this research will be presented. Such concepts are the following, the 3D simulated environment, the Motion Cueing Algorithm (MCA), the modeling of the platform, and finally the motor's angular position control. The organization of this subjects have the goal of guiding the reading from the highest abstraction layer of the system to the lowest abstraction layer. Finally, a small word about re-configurability in driving simulators is presented.

2.1 Driving Simulators

The state of the art for driving simulators is bast and complex since the history and evolution of such systems is inherently related with the progression of many fields of technology and many different industries. The nowadays called driving simulators were born from the flying simulators therefore they are mostly driven by the needs of automotive and aerospace

industries, nevertheless, the technological improvements of the computational and graphical capabilities of the digital computers have dictated the development of the driving simulators.

To start diving deeper into the specifics of the driving simulators first it´s necessary to know the functional elements of the system, in [5] the system is defined as a performance measurement device using the human-in-the-loop approach. In other words, the system feeds the user (human operator) with sensorial information and allows the user to interact with the virtual environment changing the outcome of the process, this change can be measured and compared to evaluate the performance any given variable of interest.

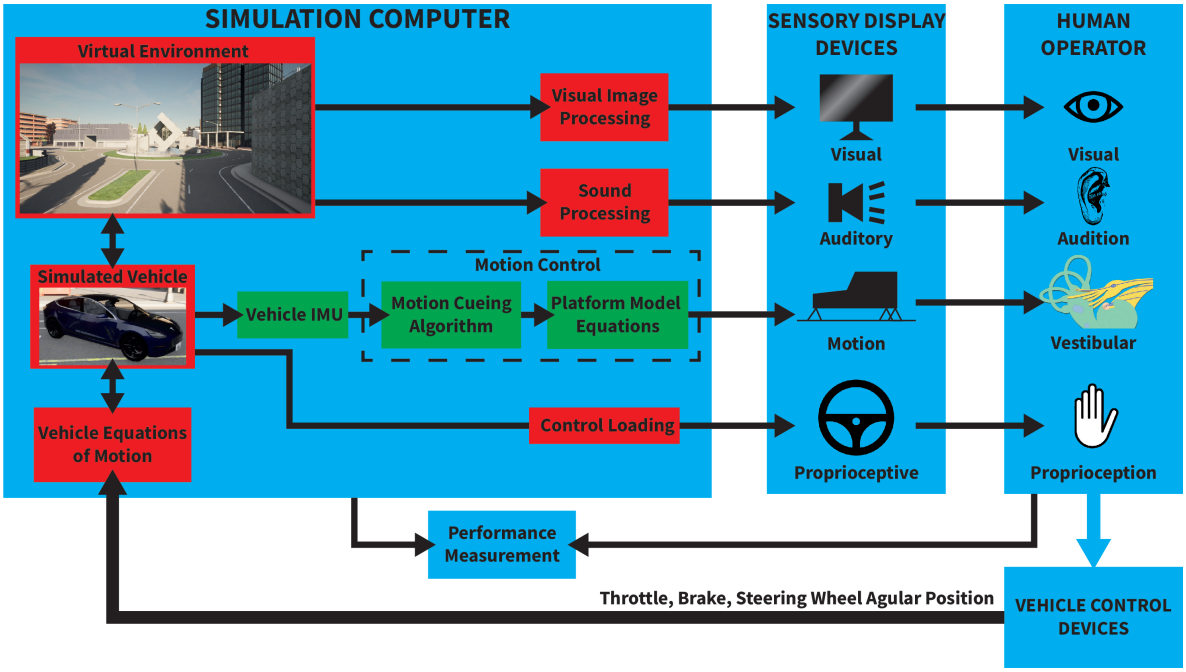

Figure 2.1: Functional blocks of a driving simulator

The functional elements of a driving simulator are shown in a block diagram in Fig. 2.1. In the highest abstraction layer, a driving simulator is made up by a computer, *sensory display devices*, *vehicle control devices* and a human operator. The sensorial organs of the human operator are stimulated by the sensory display devices providing a simulated driving situation, with this information the operator can use the vehicle control devices to interact with the virtual world created in the computer.

The computer of a driving simulator must perform a vast number of processes, the central process is the creation of a virtual environment where virtual actors can interact with the world and within each other, such actors can be pedestrians, vehicles, spectators, etc.

The next process the computer must perform is the creation of a virtual vehicle controlled by some equations of motion. Those equations take as an input the *sate of the vehicle* and

the state of the vehicle control devices, e.g., throttle pedal, steering wheel, bake, etc.

Also, the computer must process the output to the sensory display devices, while the visual and sound processing are complex systems themselves, the low-level behavior of that processing is out of the scope of this research. The control loading is the feedback force that the driver senses in some vehicle control devices, mainly in the steering wheel, the exact nature of this feature in a low abstraction layer is left as a future work.

The motion control processing is one of the central topics of this research and will be described with more detail in subsequent chapters, for now only the most basic description will be provided. The motion control has the goal of controlling a *motion platform* that stimulates the *vestibular organ* of the operator. The motion of the platform recreates the acceleration and velocity experienced by the virtual vehicle within the constraints of the *workspace* of the motion platform. To do so, the accelerations and angular velocities experimented for the vehicle must be calculated as if a Inertial Measurement Unit (IMU) where attached to the vehicle. To be mor precise, ideally the signal from the IMU should be generated as if the measurements were taken in the position where the head of the driver should go.

Once the inertial variables are calculated the system must calculate a desired platform position, since is not possible to recreate the forces experimented by the vehicle in a simulator, a Motion Cueing Algorithm (MCA) is necessary, this algorithm takes as an input the IMU´s output and calculates the desired position of a platform, the operation principle of this algorithm is related to de modeling of the vestibular organ and the otoliths. Over the years, many approaches to solve this problem have been developed but the most recognized are the classical, the adaptative and the optimal MCAs.

Finally, with the desired position of the platform, the platform´s actuators position must be calculated, this task is performed by the kinematic or dynamic modeling of the platform.

2.1.1 Driving Simulator-based Research

As stated in the introduction, the diving simulators are tools used in many research projects, up next the state of the art of the driving simulator-based research is addressed. Currently many governments and companies invest heavily in the development of driving simulators, there are even international conferences such as the Driving Simulator Conference (Europe), which brings together specialists from the academic and industrial community, while the Driving Assessment Conference (USA) is a conference linked to The University of Iowa where the focus is much more academic.

There are private companies that offer testing services in driving simulators to governments or companies, there are also private research centers that do research in driving simulators, for example in [19] the Virginia Tech Transportation Institute presents the development of a driving simulator to validate augmented reality technologies in the automobile, another

example is [20] where in Changan US R&D Center, Inc. A driving simulator was developed to test how new technologies would be accepted by drivers, in this work it is mentioned that high fidelity driving simulators require considerable investment and are not very flexible in adapting new systems, so they decided to develop their own simulator to meet their specific needs.

About driving simulators in universities, in general, the studies are carried out in simulators of reduced capacities because few universities have the means to build simulators of large capacities, the most notable exception is the University of Iowa that has the National Advanced Driving Simulator (NADS) [21], however, being this a special case, it is considered an exception. In 2015 at Clemson University, South Carolina (USA), a static simulator was developed with the purpose of training young drivers [13], in another case in 2011 at Aalborg University (Denmark) [16], a static simulator was developed seeking to do it with the lowest possible cost, in 2017 at the National Formosa University, Taiwan a static simulator for heavy vehicles was developed. A national example is the development of a tractor-trailer simulator using a 3D environment made by the Instituto Politécnico Nacional (IPN) in 2012 [15].

2.1.2 Need for Motion

Considering that the main characteristic of a driving simulator is to evaluate the performance of a human operator in a minimum risk environment, so that the operator's behavior is truly a reflection of his behavior in a real environment, it is necessary to consider the theory of risk homeostasis [22], The mechanism that governs risk homeostasis is represented in Fig. 2.2. Applying this theory to driving simulators, for a simulation to provide valid data, the simulated environment must be sufficiently realistic to generate responses that emulate the responses of a driver in real conditions [23].

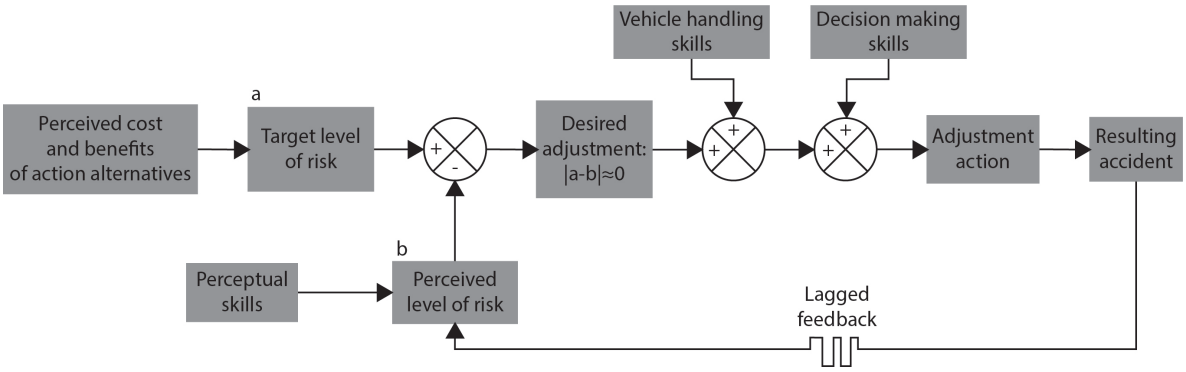

Figure 2.2: Homeostasis block diagram

The homeostasis mechanism is similar to an automatic control system where the perceived level of risk is the feedback, and the reference is the target level of risk. The reference level of

risk is a complex social, economic and psychological factor intrinsic to the driver and situation. The perception, decision making, and vehicle handling skills are other factors intrinsic to the driver and suck skills are a common field of study for driving simulators. The only component in this mechanism that is related to the driving simulator is the perceived level of risk, since the driver can't perceive something that it's not being fed to his/her perceptual organs, the driving simulator must provide as much information to the driver to make the homeostasis mechanism work in as chose to a real situation as possible.

Other factor that is impacted by the motion cueing system is the simulator sickness, this phenomenon have been discussed since the earliest days of simulator development [5]. [24] proposed that motion sickness was the result of perceptual conflict between different sensory systems. A simulator can provide unlimited visual acceleration feedback however, the motion cueing system is limited, therefore, a conflict between the visual and kinesthetic information exists. This problem is exacerbated if the driving simulator has a fixed base (static).

2.1.3 Platform Configuration

Recently, there has been an increasing interest in research for driving simulators with different configurations for automotive applications. The most widely accepted are the configuration based on the so-called Stewart platform, this system is a closed-loop mechanism made up of two platforms linked by independent serial kinematic legs or chains. In this configuration one platform is fixed to the ground and the other one moves. This platform was introduced by D. Stewart In 1965 [25] nevertheless, currently more degrees of freedom have been added or removed from the conventional 6-DOF Stewart platform.

While some important research institutes have developed driving simulators with 8 or more DOF, like the NADS in Iowa, USA [21], or PSA Peugeot-Citroën advanced driving simulator in France [11], other research institutes have developed static simulators [13] or simulators with 2-3 DOF [26]. Meanwhile, in recent years many companies started commercializing motion platforms for diving simulators ranging from 1 to 6 DOF at a low cost and with the added benefit of not requiring any special set up like rails in the ground or a secondary electrical line. The validity of the commercial platform-based driving simulators is still to be determined [27] and while they are affordable and easy to set up, they are not designed to be modified or used with custom software, therefore this kind of driving simulators lack some of the necessary features for research.

2.2 3D Simulated Environment

The 3D simulated environment main feature is to provide the visual feedback, and is the most compelling cueing system. Historically, this component of a driving simulator has been the

prime driver of development. The reason for that is that the evolution of 3D graphics has grown exponentially allowing more realistic visual feedback.

Nevertheless, visual cueing is not the only feature of a 3D simulated environment, some of the most critical features of a 3D simulates environment are:

- Scenario generation
  - Road profile
  - Environmental conditions
  - Scenario mapping
  - Illumination and visibility
- Traffic control
  - Traffic
  - Traffic control devices
  - Pedestrians
- Vehicle dynamics
- Common reference frame for different actors
- Audio generation

Ideally all these features must be running in real time to provide a responsive system. A few decades ago, it was almost impossible to run such a system in real time and with a good graphic fidelity, however, nowadays a not so sophisticated PC can do de job with flying colors. On the other hand, while the computational capabilities are no longer the impediment, generating a software that can perform all those tasks is an enormous challenge. As stated in section 1.1 many academic and research centers around the world have developed their own 3D simulated environment. Unfortunately, the requirements needed to create a valid 3D simulated environment are so complex and time consuming that it's almost impossible for a single person to program all the assets and features.

Commercial use software for driving simulators exist and the market for it is very competitive. Many commercially available software includes not just the features listed before but extended capabilities. As any other private software, any modification to the software is prohibited, discouraged, impossible or comes with an extra fee, therefore, those kinds of 3D simulated environment software are not suited for many research fields.

2.2.1 CARLA Driving Simulator

CARLA simulator is "an open-source simulator for autonomous driving research. CARLA has been developed from the ground up to support development, trained and validation of autonomous urban driving systems. In addition to open-source code and protocols, CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely. The simulation platform supports flexible specifications of sensor suites and environmental conditions" [28].

CARLA despite being designed for autonomous driving research is relevant for this work for the next reasons. CARLA offers all but one (audio generation) of the necessary features of a 3D simulation environment for driving simulators. CARLA is open-source software, which means that it's possible to recompile the software to add or modify any feature. CARLA is supported by a development team that constantly adds new features and assets alongside with a community of researchers and enthusiasts. CARLA works in Windows which means that can be used alongside other software tools like LabVIEW, MATLAB, etc.

2.3 Motion Cueing Algorithm (MCA)

"In order to make a compromise between the faithfulness of the cuing and the physical limits of the platform, several platform command strategies have been proposed. These are structured around three principles: the cueing of transitory motions, titling the platform for slow motions and, finally, returning to the neutral position when the speed of the virtual vehicle is constant. The basic idea is to perform a frequency separation, using special filters, linear accelerations and angular speeds."[4].

2.3.1 The Human Motion Perception System

It might not be evident why does a tilting motion can be equivalent to a linear acceleration, however, it's possible due the imperfections of the vestibular organ. The vestibular organ is situated in the internal ear. It's considered to be a gravitation-inertial sensor that is composed by the otoliths (linear acceleration sensor) and the semi-circular channels (angular speed sensor), nevertheless, the vestibular organ is incapable of sensing certain motions if they are blow certain threshold.

The anatomy of the vestibular organ is complex, and its study is outside of the reach of this work, nevertheless the mathematical description of the organ is relevant since it can help to understand the motion cueing algorithm.

The modeling of the semi-circular channels is a lowpass filter.[29] The transfer function of the filter is shown in (2.1), where α is the acceleration of the head, θ_e is the angular motion

of the endolymph (part of the semi-circular channels) and τ_1, τ_2 are time constants with $\tau_1 >> \tau_2$, this expression was improved experimentally in many subsequent researches. A more accurate transfer function for the scope of this research is shown in (2.2) where τ_a is an adaptation operator, a is a constant, $K_{csc} = a$, $\hat{\omega}$ is the speed of the rotation felt and ω is the rotation speed of the head [30].

$$\frac{\theta_e(s)}{\alpha(s)} = \frac{\tau_1\tau_2}{(1+\tau_1 s)(1+\tau_2 s)} \tag{2.1}$$

$$\frac{\hat{\omega}}{\omega} = K_{csc}\frac{\tau_a s}{(1+\tau_1 s)(1+\tau_a s)} \tag{2.2}$$

With:

- $K_{csc} = 5.73$
- $\tau_a = 80$
- $\tau_1 = 5.7$
- $\tau_2 = 0.005$

2.3.2 Motion cueing algorithm's general conception

With a little more understanding of the vestibular organ lets dive into the motion cueing algorithm, the main goal of the algorithm is to stimulate the vestibular organ with an angular velocity ω and a force f in a way that the $\hat{\omega}$ and $\hat{f}$ are equal to the angular velocity and force experimented in the simulated vehicle.

To archive the goal of producing the desired sensation to the driving simulator operator many different solutions have been presented, unfortunately, a systematic solution doesn't exist. Most of the existing solutions use the concept of frequency separation to take advantage of the imperfections in the vestibular system. The velocity and acceleration signals experimented by the vehicle are separated in two frequencies, high-frequency (HF) and low-frequency (LF).

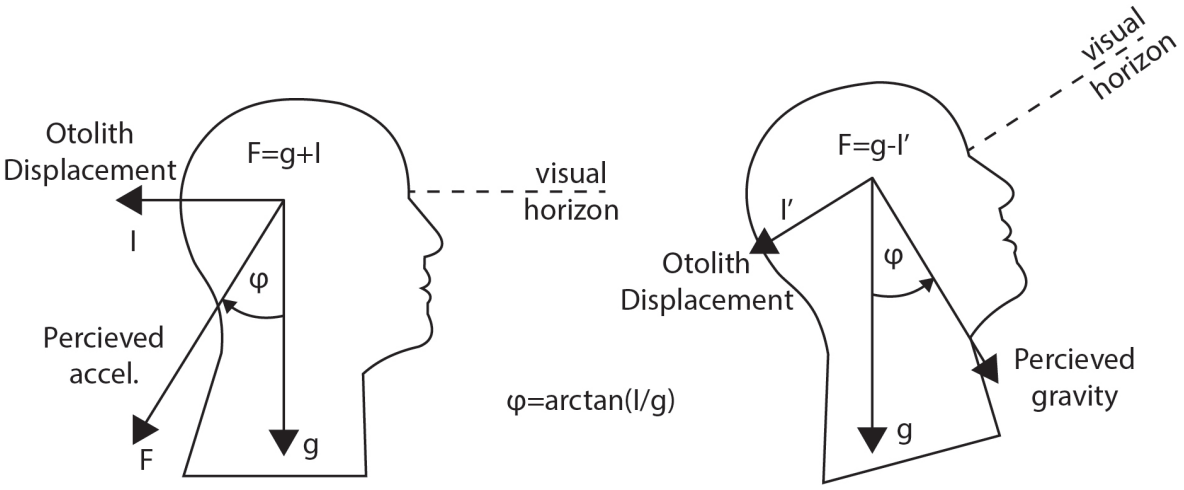

Figure 2.3: Tilt coordination principle of operation

HF components of the acceleration are called transitory and are reproduced by a linear motion of the platform, LF components, or sustained accelerations, are reproduced by tilting the platform to gain a component of the gravity vector, as shown in Fig. 2.3. The tilt will be interpreted as a acceleration as long it doesn't pass a certain threshold, this technique is called tilt-coordination [4].

Finally, when the acceleration is null, the platform must return to a "neutral position", the velocity of the return must be slow enough so the vestibular system doesn't perceive it as a negative acceleration, failing to do so will cause a "false cue". This concept is called "washout".

Using those concepts many MCAs have been developed over the years, however, there are three main branches of MCA developing, the classic, the optimal and the adaptative. Almost every simulator uses a variation of any of those approaches, being the classic the most widely implemented due to its simplicity and acceptable performance.

2.4 Platform Modeling

For the motion cueing device used to provide the diving simulator operator with motion feedback there are mainly two types of devices, a serial robotic system and a *Parallel Manipulator* (PM). Both the serial and parallel systems have their tradeoffs, but the most widely accepted configurations are the PMs, which can be categorized by the type of joint on the kinematic chains that move the platform. The most popular configuration is the Spherical-Prismatic-Spherical (SP) configuration where the kinematic chains are conformed by two

spherical passive joint and an actuated prismatic joint, this configuration is the best documented so far, but it's not the only configuration that exist, other popular configurations are the Revolute-Revolute-Spherical (RRS), RSS and RRR.

To control the position of a moving platform two components are needed. The model of the platform and the control of the actuators, those components have the goal of tracking a reference, this reference is a signal that contains the desired position of the platform. Since the platform taken in consideration is meant to be used in a driving simulator, the reference signal is a time variant not deterministic and aperiodic signal with low and high frequency components. Therefore, the validation of the tracking system can be more complex than the validation of a reference tracking system with a well-known reference signal.

The parallel manipulator problem is a very old one and it has been approached by countless authors and everyone cast light over it from a different angle but, in the end, it narrows down to a mathematical model to calculate the desired actuator position that puts the manipulator in the desired position.

There are, mainly, three subjects of study regarding parallel manipulators: 1. the inverse model, 2. the forward model and 3. the workspace calculation. When the platform uses an inverse model, the reference is the desired platform position, and the output is the actuator's desired position. In a forward model where the reference is the desired actuator position, and the output is the platform position. The workspace calculation is to determine all the possible positions that the platform can reach regarding the mechanical constrains of the specific configuration.

2.4.1 6DoF Parallel Manipulators

Some relevant works about the modeling of parallel manipulators addressed the 6-DoF, for example: [31] is a comprehensive and detailed document regarding *inverse kinematics* and dynamic modeling of the 6-SP parallel manipulator, this work considered the actuator dynamics and performs a simulation and experiment to validate the model, similarly in [32] the model developed in [31] is tested using Proportional–Integral–Derivative controller (PID) and SMC control for the actuators, and their performance measured. Regarding the forward kinematic problem, in [33] the problem is addressed using a dual quaternion and D-H method, this model can be used for a 6-R and a 2RP3R configuration, in the same topic, in [34], the research solves the problem using optimization algorithms.

2.4.2 3DOF Parallel Manipulators

Regarding the 3-DoF parallel manipulators, the attention of parallel manipulators with less than 6-DoF has increased in the past decades due to its excellent relationship between performance and price, application of 3-DoF parallel manipulators include orientation applications like, TV satellite antenna or sun tracker orientation [35], micromachining [36], laser cutting [37], and even weapon road testing [38].

Diving into the actual solution of the platform model, like its 6-DoF counterpart, the most common configuration of the 3-DoF parallel manipulators are based on actuated prismatic joint, but the main difference is that instead of using an SP confirmation, the most common configuration is a RPS (revolute, prismatic, spherical) [39, 40]. Thus, regarding the modeling of the platform, there are two approaches to the problem, the inverse kinematics and the dynamics modeling, in reference [35] a comparison between the kinematics and dynamics of a RRS and a RSR configuration is performed taking in consideration the complexity of the model, the workspace and singularities of every model; in [41] the velocity an acceleration analysis was performed for the RRS configuration, in [42] the inverse kinematics is solved using a conformal geometric algebra approach and the algorithm was validated with a small scale experimental platform, the reference [37] used the approach for the modeling of the platform taking in consideration the flexibility of the links using dynamic modeling, finally, in [43] the problem of the forward kinematic problem is addressed and its determined that the systems have 16 possible solutions for every platform position.

2.5 DC Motor Position Control

Medium-scale motion-based commercial driving simulators using a 3-DoF parallel manipulator with an RRS configuration and DC gearmotors as actuators have recently become popular. The advantage of a more affordable price than its 6-9 DoF counterpart, makes it more attractive for research in the field of driving simulators, nevertheless, real performance of those systems along with modeling and control strategies used are not openly available.

As shown in section 2.4, regarding 3-DoF RRS parallel manipulator, most of the published works are about the modeling of the platform. Nevertheless, the model of the motor actuating the revolute joint is even rarely mentioned, even though it´s a fundamental piece when it comes to the performance of the overall system and if the inertia and external forces disturbing the motor are taken into consideration no popular control strategy like PID will be able to track a position reference and compensate for external disturbances.

Regarding to the control, the selected strategy must be cable of perform the position tracking of a reference position signal and compensate the external disturbances, in other hand the system must have a saturation that maintains the system in the workspace.

2.5.1 Active Disturbance Rejection Control (ADRC)

The selected control strategy is the *Active Disturbance Rejection Control* (ADRC) since it has the feature of compensating the internal and external disturbances, this control strategy was first introduced in [44], since then it has been implemented in multiple reported projects [46][47], the main principle behind this strategy is to reduce all perturbations as a single term that is compensated in the control law, to estimate the disturbance an *Extenden State Observer* (ESO) is implemented, this observer estimates the states of the system and the perturbation in base of the control signal and the output of the system.

2.6 Re-configurability

In [45] the concept of a re-configurable driving simulator was introduced, the concept is centered in the fact that all the components of a driving simulator must be easily interconnected and replaced, along that, new features must be easily added and connected with the existing components. The proposal presented in that paper defines that the architecture of a driving simulator is composed by hardware components, software components and resources.

"The concept of the communication topology is that all components will be connected to the so-called moderator software via an input signal bus and an output signal bus instead of connecting the system components directly." [45]. This work also defines a process that a re-configurable driving simulator must follow, the process sequence is shown in Fig. 2.4.

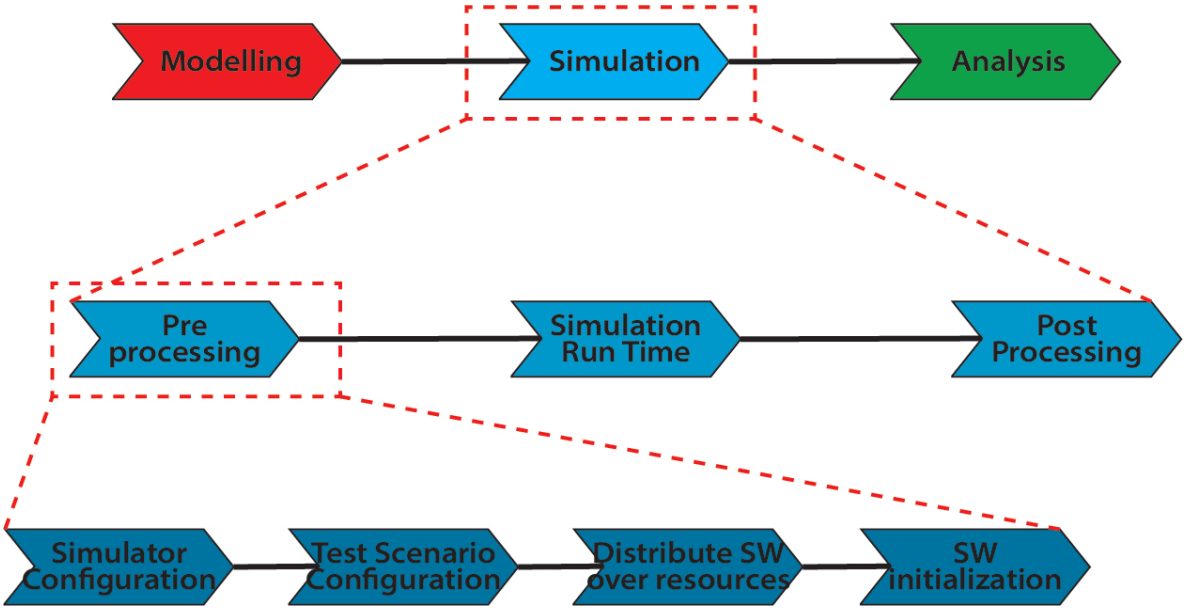

Figure 2.4: Re-configurable driving simulator process sequence

Where:

- **Modeling** is the process where all the simulator is configured and all the components and connections are defined
- **Simulation** is the process where the driving simulation takes place
  - **Pre-processing** is the process where the simulation is initialized
    - **Simulator Configuration** reads the configuration created in the modeling process
    - **Test Scenario Configuration** applies the configuration to the components
    - **Distribute SW over resources** manage the computational resources available
    - **SW initialization** starts the software applications
  - **Simulation run-time** is the process where the simulation takes place
  - **Post processing** is the process that wraps everything up
- **Analysis** is the process where the data obtained in the driving simulation process is saved.

Chapter 3

Proposed System

In this chapter the proposed system is described alongside with the design process and mathematical calculations. This chapter has the goal of describing the proposed system in the highest abstraction layer, the details for every system will be addressed in chapter 4. The design constrains and tradeoffs are discussed in detail to justify all the decisions taken in the design process.

The system is divided in different subsystems to simplify the description of the overall system, the first subsystem is the parallel manipulator design, in this section the mechanical design of a 3-RRS PM is documented. The next subsystem is the control strategy for the actuators of the platform. After that the motion control algorithms are presented. The following subsystem is the necessary software to run the driving simulator. Finally, the electronic design is addressed where the required electronic hardware is discussed.

3.1 Parallel Manipulator Design

The PM subsystem is described first because it the one with leas codependency with the other subsystems and is the one that dictates a lot of constraints for the following components of the driving simulator.

As discussed in section 2.1.3, the motion-based driving simulators with a 3-RRS parallel manipulator are popular in commercial applications, nevertheless, they are not suited for many

fields of academic research. This kind of manipulator for motion-based driving simulators offer a lot of advantages that are attractive to small and medium-size research facilities. Thus, a 3-RRS PM is selected for this research.

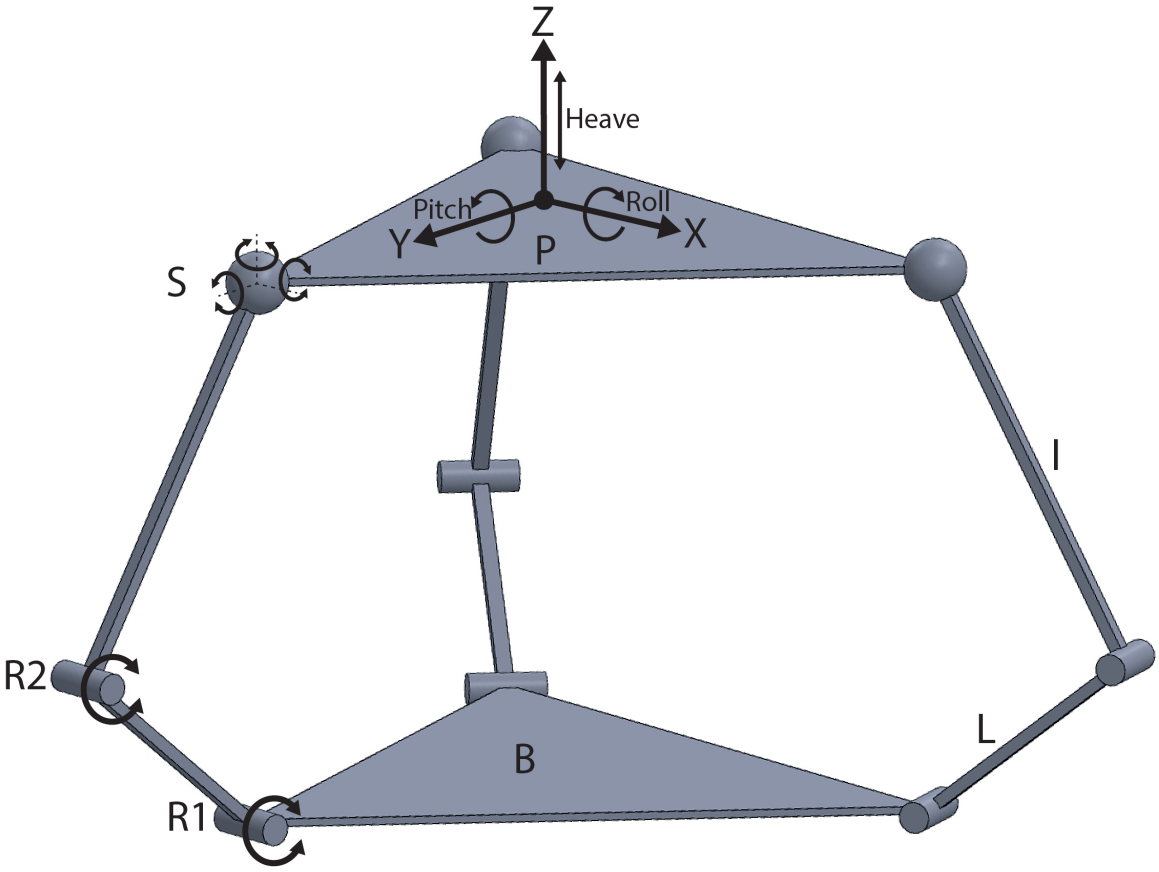

Figure 3.1: 3-RRS PM diagram and DoF

A 3-RRS PM is a parallel robot with 3 degrees of freedom (Roll, Pitch and Heave). The morphology of the PM is depicted in Fig. 3.1. Mechanically speaking the system is made by a fixed base B, a moving platform P and three kinematic chains. The kinematic chains have three joints, two revolute joints (R1, R2) and one spherical joint (S). It has two passive joints (R1, S) and an active joint (R1) that is actuated by an electric motor.

3.1.1 Modeling of the Platform

As discussed in section 2.4, it's necessary to model the platform and many authors have proposed a wide variety of solutions. The selected approach to this problem is the inverse

kinematics because is one of the most studied approach and the solution is entirely geometrical which makes it the easiest to understand, modify and implement.

The geometrical representation of the platform is shown in Fig. 3.2. It is composed by two platform and tree "legs", the coordinate system B-XYZ is attached to the center of the base platform and it is considered to remain static, in the center of the moving platform the coordinate system P-xyz is attached. B_i (i=1,2,3) are the connecting points to the base platform and similarly P_i are the connecting points of the moving platform, R_i are the union points for the passive revolute joints, L and l are the length of the links between $B_i - R_i$ and $R_i - P_i$ respectively, ℓ is the distance between B_i and P_i, the angle λ is the angular position of the actuated revolute joint B_i with respect of the vertical.

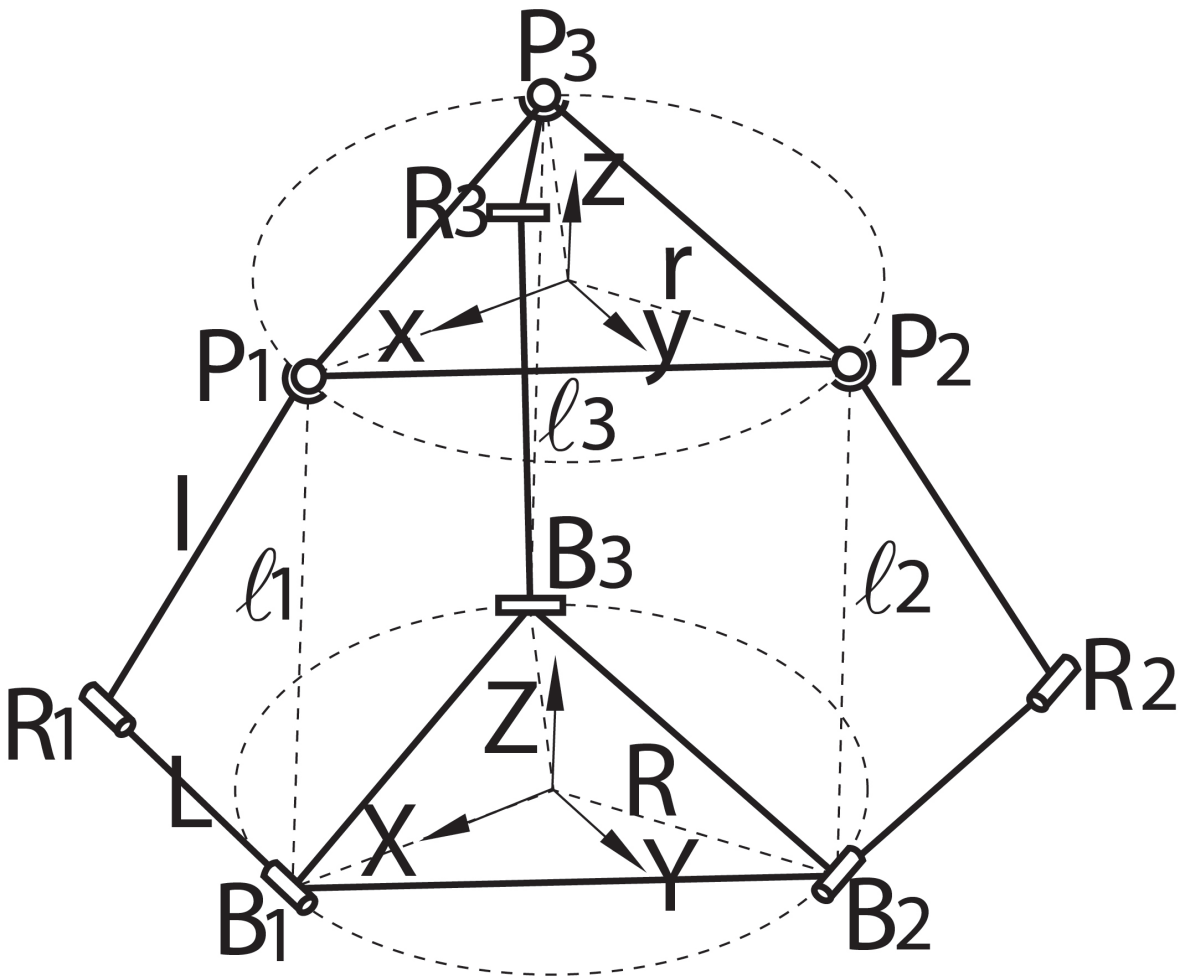

Figure 3.2: Schematic diagram of the 3-RRS parallel manipulator.

Inverse Kinematics

The inverse kinematics of a 6-SP PM is defined in [31], from this modeling is possible to modify the equations and propose the inverse kinematics of a 3-RRS PM, this is possible thanks to the geometrical similarities of both systems, nevertheless since the actuator is different a geometrical modeling of the revolute actuator is proposed.

The first step is to define the position of the base and platform union joints. Since the separation between every point in the base and platform is 120º, in (3.1) and (3.2) are defined the base and platform´s union points position.

$$B_i = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} R\cos a_i \\ R\sin a_i \\ 0 \end{bmatrix} \tag{3.1}$$

$$P_i = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} r\cos a_i \\ r\sin a_i \\ 0 \end{bmatrix} \tag{3.2}$$

With:

$$a_1 = 0^\circ, a_2 = 120^\circ, a_3 = 240^\circ \quad \text{and} \quad i = (1,2,3) \tag{3.3}$$

The next step is to calculate the new position of the platform, in order to do this, a translation vector T and a rotation matrix R are defined as a function of the three degrees of freedom of the platform, the translation vector is shown in (3.4) where the translation in the X and Y axis must be 0 and the translation on the Z axis ($\triangle Z$) is defined as the sum of the desired longitudinal translation and the platform height H, the height is set arbitrarily choosing a desired "home" position, on the other hand the rotation matrix is shown in (3.5) as a function of the desired orientation of the platform, where the rotation around the X axis an Y axis (roll, pitch) are ϕ and θ, the rotation around the Z axis is not taken in consideration since $\psi = 0$.

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ H + \triangle Z \end{bmatrix} \tag{3.4}$$

$$R = R_x(\phi)R_y(\theta)R_z(0) = \begin{bmatrix} \cos\theta & \sin\phi\sin\theta & \cos\phi\sin\theta \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\sin\theta \end{bmatrix} \tag{3.5}$$

Next, the desired position of the platform points P_d is calculated using (3.6) with i=(1,2,3)

$$P_{di} = RP_i + T - B_i \tag{3.6}$$

Once the desired P_d is obtained, the next step is to calculate the angle λ, thus, three vertical planes are defined such as the points of P are on those planes. A graphical representation of this concept is shown in Fig. 3.3.

With the new planes defined, its possible to model the kinematic legs, as shown in Fig. 3.4. Using this geometric representation is possible to calculate the angle lambda. That angle is obtained by calculating the angles α, β an the line ℓ using (3.7) to (3.10).

$$\ell_i = \|P_{di}\| \tag{3.7}$$

$$\alpha_i = \arccos \frac{L^2 + \ell_i^2 - l^2}{2L\ell_i} \tag{3.8}$$

$$\beta_i = \arccos \frac{P_{dzi}}{\ell_i} \tag{3.9}$$

$$\lambda_i = \alpha_i + \beta_i \tag{3.10}$$

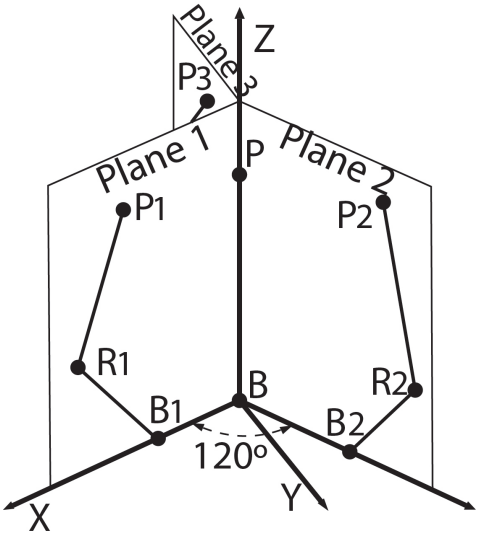

Figure 3.3: Proposed vertical planes.

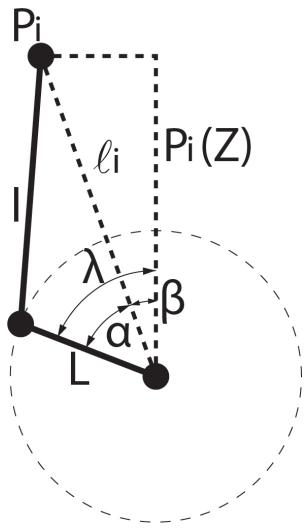

Figure 3.4: Kinematic legs geometry.

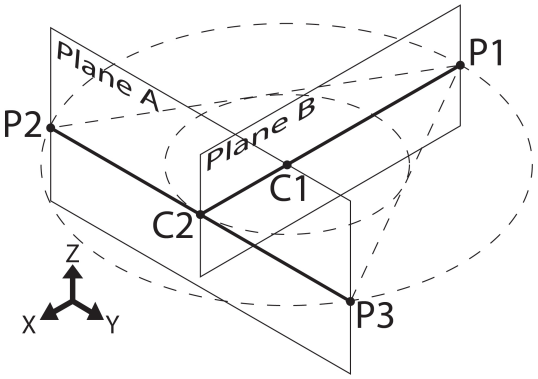

Figure 3.5: Proposed geometry of the platform using points P_i.

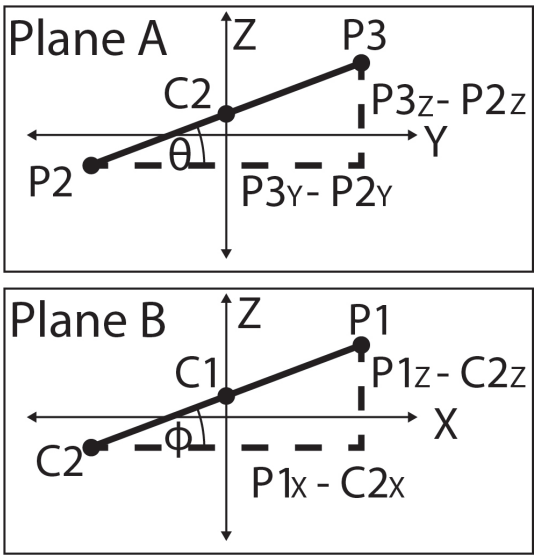

Figure 3.6: Frontal view of the panes generated from P_i.

Validation of the Model and Forward Kinematics

To validate the modeling and inverse kinematics of the platform is necessary to measure the difference between the platform's reference position and the actual position of the platform. To do so it's necessary to calculate the forward kinematics. The problem of the forward kinematics it's more complex than the inverse kinematics therefore a simulation-based solution is proposed. In chapter 5, section 5.1 the output of the simulation that validates the modeling and the control strategy will be presented.

For the simulation-based forward kinematics, in this approach the position of the union points of the platform (P_{ri}) are assumed to be known, the proposed geometry for the platform its shown in Fig. 3.5, the Planes A and B defined in Fig. 3.5 are shown in Fig. 3.6, points *P2* and *P3*, defined in plane (A), are used to calculate the platform roll, then (3.11) is used to calculate the pitch of the platform (ϕ); (3.12) to calculate point *C2* on plane A; (3.13) together with *C2* on plane B, to calculate the roll of platform (θ); and (3.14) is used to calculate heave ($\triangle Z$).

$$\phi = \arctan \frac{P3_Z - P2_Z}{P3_Y - P2_Y} \tag{3.11}$$

$$C2 = \frac{P2_Z + P3_Z}{2} \tag{3.12}$$

$$\theta = \arctan \frac{P1_Z - C2_Z}{P1_x - C2_x} \tag{3.13}$$

$$\begin{aligned} m &= \frac{P1_X - C2_X}{P1_Z - C2_Z} \\ \triangle Z &= P1_Z - mP1_X \end{aligned} \tag{3.14}$$

3.1.2 Scale and Full-Scale model

To implement a motion-based driving simulator a full-scale model is needed. The design of every mechanical component for a full-scale PM is proposed, nevertheless the manufacturing and implementation of such system is left as future work.

A scale model is proposed to test the driving simulator. While the main propose of a motion-based driving simulator is to implement a human-in-the loop system and to do so is impossible with a scale model, such model will allow to test the system within the time and resources constraints of this project.

Unfortunately, a scale model will not be able to test the mechanical needs of a full-scale model, this means that it's not possible to test that the mechanical design is able to withstand the mechanical stress, strains, deformations, etc. Therefore, to test the mechanical design of the full-scale model a finite element simulation is proposed. The system must be able to withstand the weight of an 80Kg person plus equipment.

The material selection for the full-scale system must meet the next requirements: The material must be of commercial use, can be purchased nationally, can be machinable with chip breaking machining and must be of the lowest price possible.

The mechanical design must meet the following requirements: all pieces must be manufacturable with CNC-type machines or with manual machining, must use commercial screws, nuts, washers, etc. it must not depend on tight Geometrical Dimensioning and Tolerances (GD&T) and must be possible to assemble the system with basic tools.

The scale model must try to replicate as close as possible the full-scale model, the model must be printable in 3D and the actuators must be of the same type of the full-scale model.

3.1.3 Selection of the Mechanical Dimensions

The mechanical dimension establishes the workspace of the PM and the toque needed in the actuators to move a load, in that note, the link L is acting as a crank attached to the motor that actuates the joint B_i.

To calculate the amount of weight that a motor with certain torque will be able to lift, a static analysis is proposed because, although the simulator is a dynamic machine, the

dynamics of the platform takes into consideration the inertia and acceleration of the moving masses and this requires deterministic knowledge of unknown magnitudes such as the weight and morphology of all the peripherals connected to the platform including the operator, therefore a static analysis allows to make an approximation of the operating ranges of the system.

To perform the static analysis of the amount of weight, defined as W, that each motor will be able to lift (3.15) is used where: τ is the torque of the motor, L is the crank length and g is the gravity.

$$W = \frac{\tau}{gL} \tag{3.15}$$

This is the weight that each leg could lift statically, if we assume that the weight is divided equally in the three legs then the total weight that can carry the platform is multiplied by three, however, the total weight that the platform can carry dynamically is approximately $\frac{2}{3}$ of what is obtained in the static analysis.

3.2 Control Design

To control the angular position of the motor that actuates the active revolute joint (R1), it's necessary to use an *automatic control strategy*, such strategy must be able to track the position reference. The nature of the angular position reference signal will be addressed in subsequent sections. Alongside with *tracking*, the control strategy must be capable of compensating the external disturbances and the stationary state error.

The motor is expected to experiment strong external disturbances due to the load on the driving simulator, ergo, a classic PID control might not be enough to reach the expected performance. An *Active Disturbance Rejection Control* (ADRC) strategy is proposed.

The last design criteria is the type of motor that will be controlled, a ferrite permanent magnet DC gearmotor is proposed. The selection process was based on availability, affordability, torque and speed. The reference values for the torque and speed were obtained from existing commercial driving simulators. Ideally the motors for the full-scale model must be at least able to provide 30Nm and 50rpm. For the scale model the motor should be of the same type so the mathematical model can be shared between the two motors.

3.2.1 DC Motor Model

The DC motor can be modeled in many ways, e.g. as a system of first or second order, nevertheless, in this research the motor is represented as a perturbed double integrator, and (3.16) is proposed. In this equation the rotor acceleration is calculated as a function of the

armature voltage and it takes into consideration the endogen perturbations as the factor including angular velocity, and the exogen perturbations as the factor that containing the torque applied to the motor. Once the motor's acceleration is calculated, it is integrated two times to obtain the position.

$$\ddot{q} = \left(\frac{K_a K_b}{J_m R} + \frac{B_m}{J_m}\right)\dot{q} + \frac{K_a}{J_m r R}V - \frac{\tau}{J_m r^2} \tag{3.16}$$

Where:

- q = angular position
- $\dot{q}$ = angular velocity
- $\ddot{q}$ = angular acceleration
- K_a = motor-torque constant
- K_b = counter electromotive constant
- J_m = rotor inertia
- R = armature resistance
- B_m = viscous resistance
- τ = torque applied to the motor
- r = gear relation

3.2.2 ADRC

The selected control strategy is the *Active Disturbance Rejection Control* (ADRC) since it has the feature of compensating internal and external disturbances. This control strategy was first introduced in [44], since then it has been implemented in multiple reported projects [46, 47, 48], the main principle behind this strategy is to reduce all perturbations to a single term that is compensated in the control law. To estimate the disturbance, an *Extenden State Observer* (ESO) is implemented, this observer estimates the states of the system and the perturbation based on the control signal and the output of the system, Fig. 3.7 shows the block diagram of the ADRC.

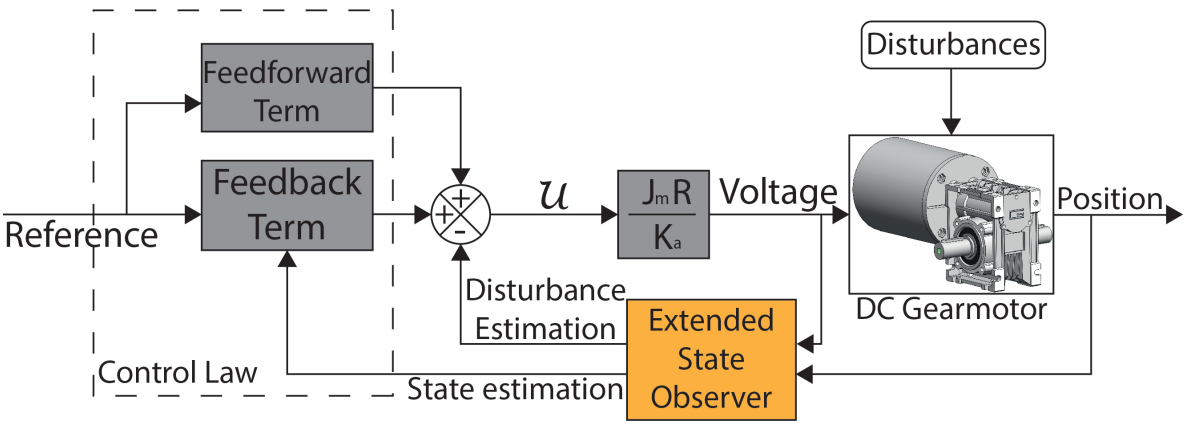

Figure 3.7: ADRC block diagram

To design the control, (3.16) is modified to group all disturbances in a single term ζ, the resulting expression is shown in (3.17), then, making $\ddot{q} = u$, where u is the control signal, and leaving only v on the left side of the equation as shown in (3.18); defining the system states as (3.19), the next step is to define a control law, the proposed control law is stated in (3.20) where x_d represents the motor's desired position, this control law is for tracking the reference, and it relays on the first and second derivative of the reference signal, therefore, the reference signal must be derivable at least two times. If the signal is not derivable, 3.21 is the control law for regulation.

$$\ddot{q} = \frac{K_a}{J_m r R} V + \zeta \tag{3.17}$$

$$V = \frac{J_m r R}{K_a}(u - \zeta) \tag{3.18}$$

$$\begin{aligned} x_1 &= q & \dot{x_1} &= x_2 \\ x_2 &= \dot{q} & \dot{x_2} &= u + \zeta \end{aligned} \tag{3.19}$$

$$u = \ddot{x_d} - K_1(x_1 - x_d) - K_2(x_2 - \dot{x_d}) - z_1 \tag{3.20}$$

$$u = K_1(x_1 - x_d) - K_2 x_2 - z_1 \tag{3.21}$$

Where:

- K_i = control gains $i = (1, 2)$
- z_1 = estimated disturbance ($z_1 \approx \zeta$)

ESO

The ESO is a key feature of the ADRC since it can estimate the system disturbance z_1, and the states of the system $\hat{x_i}$ with $i = (1, 2)$, the equations that define the observer are shown in (3.22), these equations use the constants o_i with $(i = (0 - 3))$, these constant values must be carefully calculated since they rule over the speed of the observer, as a rule of thumb, the ESO must be faster than the controlled system but the exact values needed to tune the system require a fair amount of trial and error, nevertheless there is a useful methodology to calculate those values, a fourth-order Hurwitz polynomial is defined as shown in (3.23). Then the roots of that equation are used as the constants o_i. This leaves only the natural frequency ω_n and the damping factor ξ to be defined as shown in (3.24).

$$\begin{aligned} \dot{\hat{x_1}} &= \hat{x_2} + o_3(x_1 - \hat{x_1}) \\ \dot{\hat{x_2}} &= u + z_1 + o_2(x_1 - \hat{x_1}) \\ \dot{z_1} &= z_2 + o_1(x_1 - \hat{x_1}) \\ \dot{z_2} &= o_0(x_1 - \hat{x_1}) \end{aligned} \tag{3.22}$$

$$(s^2 + 2\omega_n\xi s + \omega_n^2)(s^2 + 2\omega_n\xi s + \omega_n^2) = 0 \tag{3.23}$$

$$\begin{aligned} o_3 &= 4\omega_n\xi \\ o_2 &= 2\omega_n^2(1 + 2\xi^2) \\ o_1 &= 4\omega_n^3\xi \\ o_0 &= \omega_n^4 \end{aligned} \tag{3.24}$$

3.2.3 Discretization of the ADRC

The ADRC described in section 3.2.2 is defined in continuous time, nevertheless to implement the ADRC in a digital system its necessary to define the control and the ESO in discrete time. The discrete control theory is very powerful but it comes with a lot of trade-offs, therefore, a discretization technique of the continuous-time defined control and ESO is proposed. To perform the discretization (3.25) is used where h is the sampling rate of the digital system, this equation is useful to calculate an approximation of the derivative of a signal.

$$\frac{d\hat{x}}{dt} \approx \frac{\hat{x}(t_{k+1}) - \hat{x}(t_k)}{h} \tag{3.25}$$

After using the approximation of (3.25) in (3.22), the resulting expression is (3.26), this way it's possible to calculate the ESO in a digital system, its important to note that the sampling rate h must be low enough to make the approximation of the derivative valid. The exact value of the sampling rate must be at least $\frac{1}{10}$ of the time that takes for the plant to reach 93% of the reference for a unitary step input.

$$\begin{aligned} e(t_k) &= x_1(t_k) - \hat{x_1}(t_k) \\ \hat{x_1}(t_{k+1}) &= \hat{x_1}(t_k) + h[\hat{x_2}(t_k) + o_3 e(t_k)] \\ \hat{x_2}(t_{k+1}) &= \hat{x_2}(t_k) + h[u + \hat{z_1}(t_k) + o_2 e(t_k)] \\ \hat{z_1}(t_{k+1}) &= \hat{z_1}(t_k) + h[\hat{z_2}(t_k) + o_1 e(t_k)] \\ \hat{z_2}(t_{k+1}) &= \hat{z_2}(t_k) + h[o_0 e(t_k)] \end{aligned} \tag{3.26}$$

For the regulation control law the discretization process is very simple since it doesn't contain any derivative, therefore (3.27) is obtained. For the tracking control law, the first two derivatives of the input are needed, to calculate both derivatives in a embedded system can be complex, therefore $\ddot{x_d}(t_{k+1})$ and $\dot{x_d}(t_{k+1})$ will be calculated on the PC and passed to the embedded system, in that way (3.28) can be defined.

$$u(t_{k+1}) = K_1[x_1(t_{k+1}) - x_d] - K_2[x_2(t_{k+1})] - z_1(t_{k+1}) \tag{3.27}$$

$$u(t_{k+1}) = \ddot{x_d}(t_{k+1}) - K_1[x_1(t_{k+1}] - x_d(t_{k+1})) - K_2[x_2(t_{k+1}) - \dot{x_d}(t_{k+1})] - z_1(t_{k+1}) \tag{3.28}$$

3.3 Motion Control

The motion control is made up by the inverse kinematics of the platform, discussed in section 3.1.1 inverse kinematics, and the MCA. The propose of the MCA, as discussed in section 2.3, has the goal of creating the desired position of the platform from the virtual vehicle's IMU. There are two steps to implement an MCA in a driving simulator, the design of the MCA itself and the tunning thereof.

To design an MCA the first step is to select the scope of maneuvers that are intended to be simulated. The next step is to select an MCA from the literature in the field of the motion cueing. From the literature many MCAs can be found, each one with its tradeoffs, and everyone was developed to address a specific problem or improve the performance of a past work. The last step is to modify the selected MCA to fit the mechanical constraints of the motion platform that will be used in the driving simulator. The Tuning of an MCA

consist in modifying the gains that modify the behavior of the motion feedback. This process is normally performed on-line with an expert simulator operator and a expert in the driving simulator. This process is heavily time consuming and totally subjective since it doesn't exist an objective stopping point, instead, the process ends when any of the experts involved in the tuning gets tired of the time runs out [49].

The proposed MCA is a modification of the classic MCA, the full version of the classic approach takes into consideration a motion platform with 6-DOF. Therefore, the proposed MCA is a trimmed version of the classical filter-based MCA.

3.3.1 Proposed MCA

The classic MCA has been the benchmark for any new development since the early days of flight simulators. Since this research have the goal of developing a platform to develop any required feature for a driving simulator, the choice of implementing the classic MCA is evident. The design and implementation of other types of MCA will be future work.

The proposed MCA is shown in Fig. 3.8 and the center of the algorithm are the filters that separate the low and high frequency components of the vehicle's IMU. There are two low-pass filters and three high pass filters, the filters are modeled as transfer functions of first and second order, such transfer functions have two constants (ξ, ω_n) that define the behavior of the filter.

The low pass filtered acceleration signal goes thru the tilt coordination process, this part has the goal of transforming a linear acceleration in a perceived acceleration by tilting the platform. To do so the concept shown in Fig. 2.3 MCA general conception is applied. Finally, the rate limiter is a saturation function that limits the output of this component to maintain the platform in its workspace.

The high pass filtered angular velocity signal is integrated to obtain an angular position, this position signal is summed to the saturated tilt coordination signal to obtain the desired rotation of the platform. The gains 1 to 5 are used to tune the MCA where a gain on 1 represents that the reference signal is stimulated as it is in the platform, therefore, the max value of the tuning gains is should always be 1. For the accelerations in the Z axis, the signal goes thru a high-pass filter and its integrated two times. This transforms a linear acceleration in a linear position.

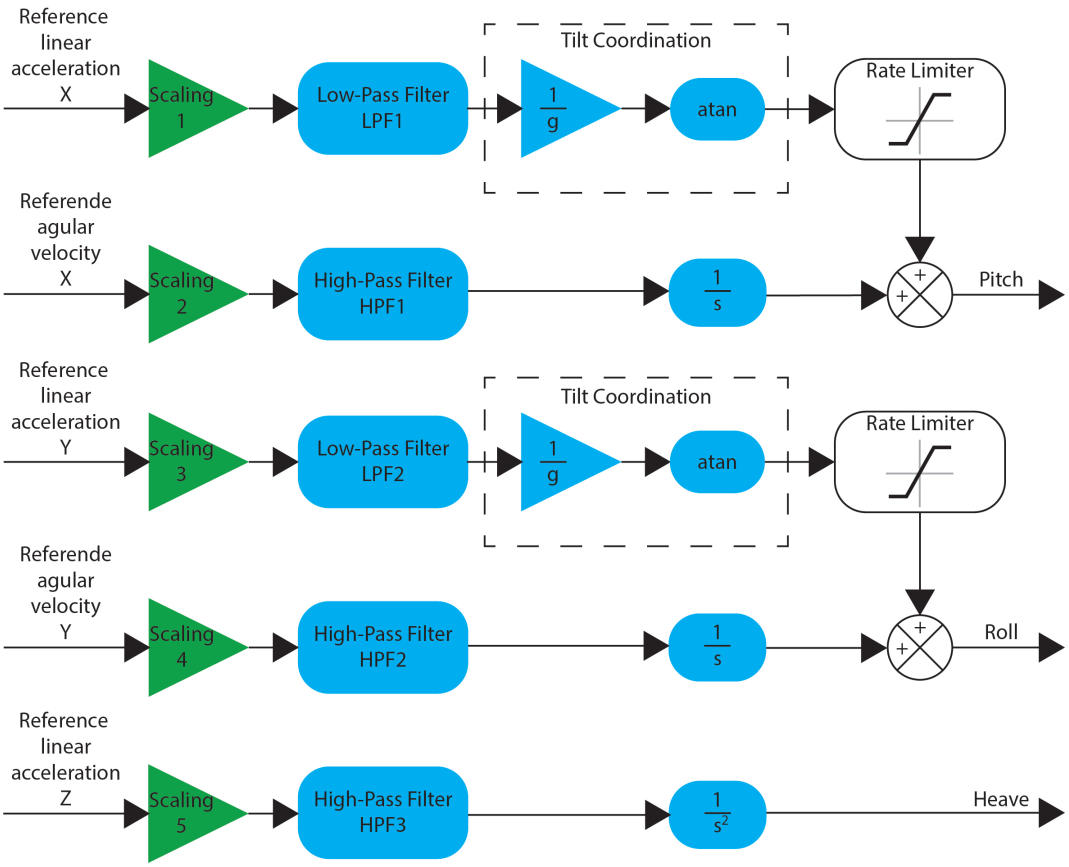

Figure 3.8: Proposed MCA block diagram

The proposed transfer functions of every filter alongside with the value of the constants (ξ, ω_n) are shown it table 3.1, the values of the constants are taken from [50]. The optimization of the filters is left as future work.

Filter	**Order**	ω_n	ξ	**TF**
LPF 1	2	5.0	1.0	$\frac{\omega_n^2}{s^2+2\xi\omega_n s+\omega_n^2}$
LPF 2		8.0		
HPF 1 and 2	1	1.0		$\frac{s}{s+\omega_n}$
HPF 3	2	4.0		$\frac{s^2}{s^2+2\xi\omega_n s+\omega_n^2}$

Table 3.1: MCA's High-Pass and Low-Pass parameters

3.4 Software

The software required to implement the driving simulator is the most critical part of the development, the software must execute fast enough to be considered real time and it must

perform various tasks communicated with each other. The block diagram of all the software components is shown in Fig. 3.9.

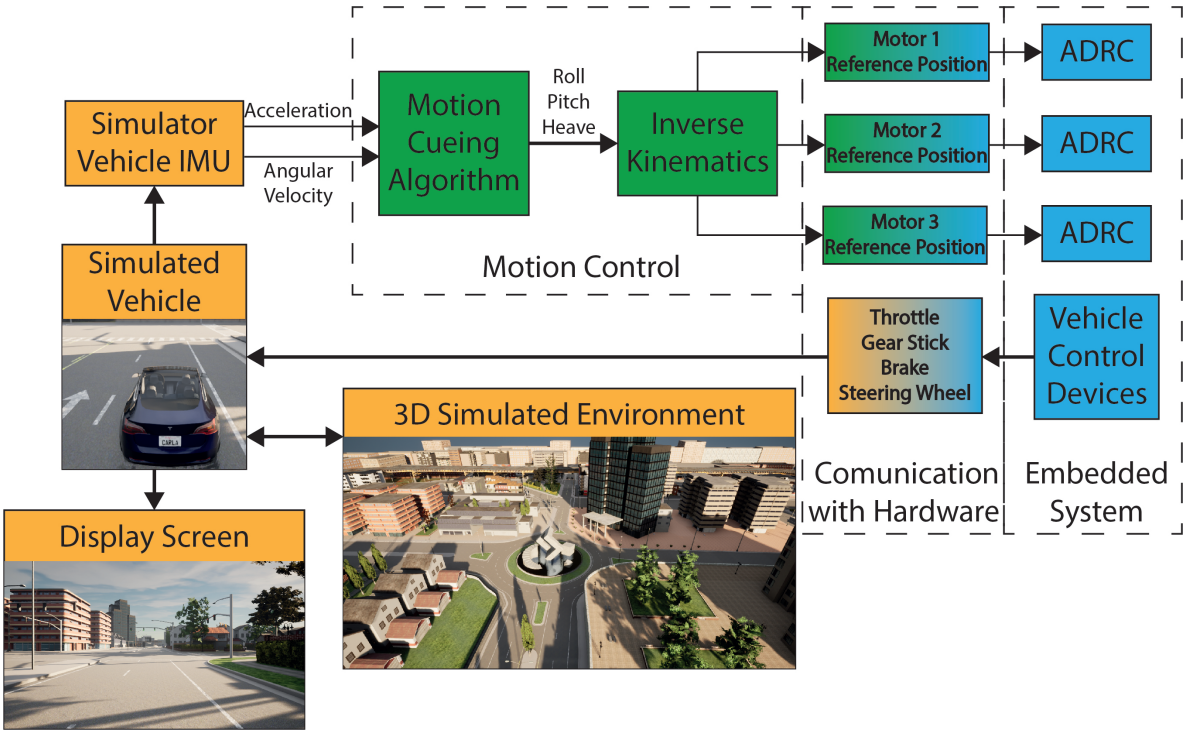

Figure 3.9: Software block diagram

The orange and green blocks in Fig. 3.9 represent the task that must be performed in the PC, and the blue blocks represents the task that must be performed in a separated electronic hardware, capable to control the motors and acquire the vehicle control device's signals. The green block represent the task required to perform the motion control calculation, the orange block represent the task associated with the simulated environment.

The tasks associated with the simulated environment will be described in section 3.4.1, the task associated with the motion control were described in section 3.3, the tasks process to communicate the PC with the external hardware will be detailed in section 3.4.2, finally, the tasks that must be performed in a external embedded system will be described in section 3.4.3

3.4.1 CARLA Simulator

In this proposal the CARLA Simulator will be used to perform all the tasks related with the 3D simulated environment as shown in Fig. 3.9, but first it's necessary to understand how the CARLA Simulator works. This open source software works with a client-server architecture,

the server contains the 3D simulated environment were "actors" interact with the simulated world with a common reference frame, on the other hand, the client is a piece of software that uses a Python API to communicate with the server using the Transmission Control Protocol (TCP) protocol. The CARLA server can be modified and recompiled downloading the source code, however, in this proposal the server will be left unmodified and the development will be centred in the client.

The client have many responsibilities but it can be narrowed down to two main concepts, the communication with the CARLA server and the communication with the next piece of software that will be executing the motion control and the communication with the hardware. To be more specific, the CARLA client needs to connect with LabVIEW, the proposed methodology to do so is to use, once again, the TCP protocol where the LabVIEW visual instrument will be the client and the CARLA client will be the server, to illustrate the communication between those three pieces of software,and the external hardware, in Fig. 3.10 a block diagram of the interconnection of the parts is shown.

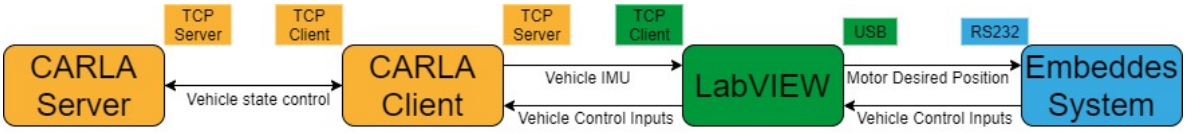

Figure 3.10: Software connection block diagram

On top of managing the communication, the CARLA client must perform numerous tasks related with the control of the CARLA server, since the CARLA server is a passive piece of software that only executes the instructions provided to it with the Python API. A detailed description of those task will be presented in chapter 4, for now the tasks will be enumerated up next:

- World initialization
- Vehicle creation
- Weather management
- Sensors creation/management
- Camera creation
- Display rendering
- Help Text/HUD generation

3.4.2 Communication with Hardware

The proposed communication with the external electronic hardware will be performed by LabVIEW since this graphical programming environment is perfectly suited for developing programs that incorporate external hardware.

One of the main characteristics of this software must be implementation of a fast an reliable communication protocol. The available communication channels in a common PC include WiFi, Ethernet, Bluetooth and USB. Since the wireless connectivity is not a desired feature, the remaining options are Ethernet and USB, from those two field buses Ethernet would be the ideal options since it provides a robust and fast communication, nevertheless, the embedded systems with a hardware Ethernet controller are not as common or affordable as a USB solution, another problem with the Ethernet is that the developing time to implement of a embedded system with such protocol is exponentially bigger than a USB-based solution.

The USB communication used to implement a virtual serial port and, thus, a serial protocol communication, is one of the worst options available, since its slow and not very robust, nevertheless it's, by far, the quickest and easiest protocol to implement, on the other with the the proper flow control, this protocol can meet the needs of this project. Therefore, to meet the time and budget constrains, the USB/serial protocol is proposed to implement the communication with the external hardware.

3.4.3 Low-Level Software

The low level software refers to the program saved in the memory of the embedded system, to develop a low level program there are many options depending of the manufacturer of the device. While assembler programming will always provide the most control over the device, developing complex algorithms in assembler can be really challenging and time consuming, on the other hand, some devices can be programmed with graphical programming languages, this approach reduces enormously the development time but takes away many resources of the device. Therefore, a compromise must be made between the control and the ease of use at the moment of selecting a programming language.

The use of MicroPython is proposed since it offers access to most of the devices capabilities with a comprehensive syntax that speeds the development time when compared with other text based programming languages. "MicroPython is a full Python compiler and runtime that runs on the bare-metal. In addition to implementing a selection of core Python libraries, MicroPython includes modules such as 'machine' for accessing low-level hardware." [51]. MicroPython is implemented for a short amount of devices compared with other similar options, the list of supported devices include some STMicroelectronics Microcontroller Unit (MCU) and Espressif ESP MCU.

The responsibilities of the low-level software is to manage the peripheries of the embedded device, calculate the control, handle the serial communication with the PC, condition the acquired signals of the vehicle control devices and keep a stable performance

3.5 Electronics

As stated in 3.2.3, the control strategy for the motor's angular position requires a critically constant and small sampling and updating period, to achieve such a thing in a multi task, multi user operative system as Windows it's almost impossible. To implement such control strategy, its necessary to perform floating point operations in a short time, therefore it´s necessary to have a processing unit with a Floating Point Unit (FPU). In the market many devices with a FPU can be found, nevertheless, The devices best suited for this task with the lowest price are the ARM-based MCU.

To implement a embedded system as described before it's necessary to use a development board or a custom made board that contains the selected MCU, in that note, for the scale model driving simulator the use of a commercial development board is proposed, in the other hand, for the full scale system the design of a custom board is proposed, unfortunately, due to time constraints the fabrication of the designed board will be left as future work.

The process to select the specific MCU that will be used starts by selecting the manufacturer, in this case the options narrow down to Espressif and STMicroelectronics since they are the only supported manufacturers by MicroPython. With those two options in mind, there's no real question, the optimal option is STMicroelectronics due to the enormous quantity of MCUs available that one can be choose from.

The next step is to select the family of MCUs, the proposed family are the "High Performance" family, and within that family the STM32F4 Series is poposed sinche it's the series with the best price-performance relationship of the high performance family. The STM32F4 Series is based on ARM Cortex M4 where "The STM32F4 series consists of eight compatible product lines of digital signal controllers (DSC), a perfect symbiosis of the real-time control capabilities of an MCU and the signal processing performance of a digital signal processor (DSP)" [52]. This in theory makes this series a perfect fit to the needs of this project.

The final step is to select a MCU is to actually do exactly that, select a specific device, the selection must be made between the boards available for purchase in Mexico that also have a MicroPython port ready to use. With those restrictions in mid, the selected device is the STM32F407VGT6 and the development board "Ophyra" from the company Intesc.

Finally, since the electronic system's responsibility is not only the control of the platform, a interconnection architecture between the different electronic components must be proposed, the complete system must communicate the PC to the embedded control and the driver's

control devices. in the following subsections the proposed architecture for the scale and full-scale systems will be addressed.

3.5.1 Electronics for the Full-Scale System

The full-scale proposed full scale system must address issues that are not present in a scale model, for example, if the communication between any of the components fails in the full-scale system, an accident with possible human injury can occur and in the worst case scenery this could lead to human loss. To address this problem is important to implement security features in the platform controller and to communicate all the components with a Fieldbus to assure that the communication is not the source of an error.

There are many Fieldbus protocols used in industrial robots like PROFIBUS, MODBUS, INTERBUS, PROFIsafe, CAN BUS or even Ethernet. Every Bus protocol have different features that make them best suited for a specific application, therefore, its necessary to select the protocol that suits the best this project. When it comes down to select a Bus protocol, there are many thins to consider, like bandwidth, speed, security features, maximum distance, etc.

Another important factor to consider is the implementation requirements, to use a Fieldbus two thins are necessary a Transceiver and a controller. a transceiver can be purchased as an IC and the controller can either be acquired as a speared Integrated Circuit (IC) or be embedded directly into a MCU. This comes with availability and cost related issues, specially if it's desired to have the controller embedded in the MCU. This narrows the options of available protocols to very few candidates. Between the few commercially available options the CAN BUS is the protocol that is most widely implemented by ICs manufacturers.

The CAN BUS protocol is old, slow (compared with other protocols) and it can carry only 8bits per message, nevertheless is very reliable, widely used in the automotive industry and has a maximum length of 250m (depending on the frequency). Those features make this protocol obvious choice for this project, specially because many automotive spare parts communicate naively thru CAN BUS. When choosing between a discrete IC controller and a embedded one, the best choice is always an embedded controller, the reason for that is the fact that a controller must be managed by an external logic. Taking this fact into consideration, a embedded controller is directly connected to the CPU and share the register memory, buses, DMAs etc. While a discrete controller must communicate with a MCU or CPU using another communication protocol like SPI. This extra communication delays the system operation and makes the communication susceptible to external disturbances because SPI has no failure recovery or noise robustness.

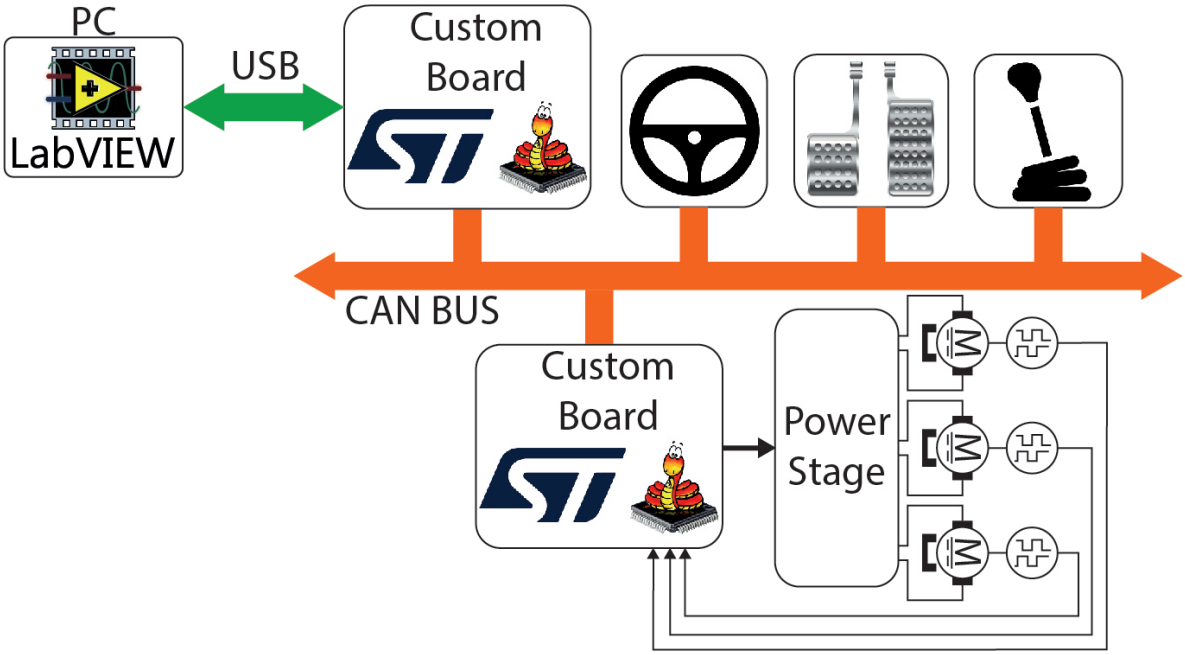

Figure 3.11: Proposed Electronic architecture for the Full-Scale system

Finally, since the selected MCU have a embedded CAN BUS controller, the proposed architecture of the system is depicted in Fig. 3.11. In this diagram it can be seen that two custom boards are used, one manages the communication between all the devices and the PC, the other one executes the control of the platform's motors. The vehicle control devices are automotive spare parts that communicate thru the CAN BUS, for example the angular position of the steering wheel is sensed with a steering sensor, the throttle and the bake are automotive pedals and so on. Even if the Full-Scale model System, the foundation for a future implementation is presented.

3.5.2 Electronics for the Scale System

Unlike the Full-Scale system, the scale version can disregard the Fieldbus and the vehicle control devices can be implemented with simple components like potentiometers and switches. Therefore, the electronic system is reduced to a acquisition system that reads the value of the vehicle control devices and the embedded control system, the acquisition system will be detailed in the next subsection and the proposed system that controls the platform's motors is depicted in Fig. 3.12.

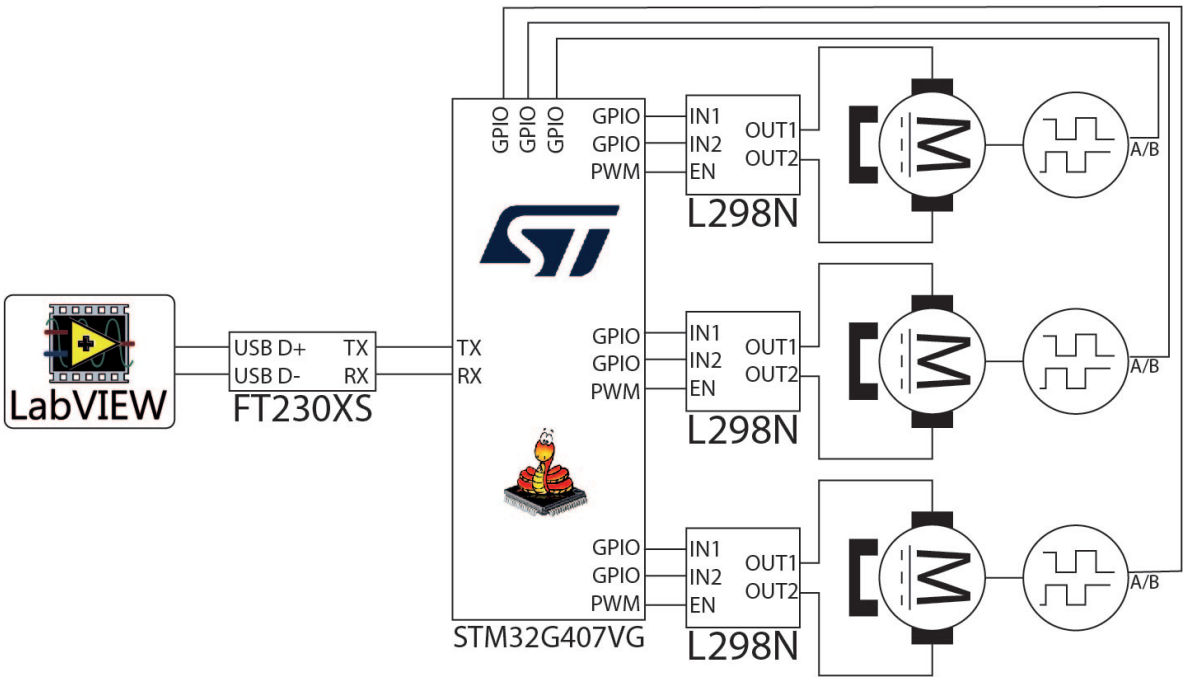

Figure 3.12: Proposed Electronic architecture for the Scale system's platform controller

In Fig. 3.12 it can be seen that the MCU communicates with the PC thru a USB Full Speed to Basic Universal Asynchronous Receiver-Transmitter (UART) IC from FTDI, this component is included in the selected commercial board. The proposed power stage is the H-Bridge L298N that can be acquired as a ready to use module that includes all the components needed to use the device, this power stage drives the scale platform's motors. The proposed methodology to read the motor's angular position is an incremental quadrature encoder. The encoder signals are read by the MCU to calculate the control output.

3.5.3 Vehicle Control Devices

The proposed vehicle control devices for the sale model are the steering wheel, the throttle pedal, the brake pedal, a reverse selector and a hand brake selector. The selected devices represents the variables that the CARLA Simulator take as an input for a vehicle. The proposes way to implement the steering wheel, the throttle pedal and the brake pedal is using potentiometers. On the other hand the reverse selector and the hand brake are proposed as a switch or push button. Therefore, the acquisition system must read 3 analog signals, two digital signals and send that information to the PC thru USB. Since the control devices are not expected to have high frequency components a high sample rate can be used. With this

design criteria established it is possible to implement the system using a slow MCU-based device as an Arduino Nano.

The steering ratio of a passenger car goes from 12:1 to 20:1, meaning that the steering wheel has 2.7 to 3.2 turns lock-to-lock. For this implementation, 3 turns was chosen for the steering wheel, this is an arbitrary middle value that is not based in any further research, nevertheless, as a proof of concept, any value between 2.7 and 3.2 could be used.

Chapter 4

Implementation

In this chapter all the details about the implementation are discussed. This chapter has the goal of documenting the implementation process of the proposed system described in Chapter 3. The organization of this chapter has the focus on the tools used to develop every subsystem needed to implement a re-configurable driving simulator. The components that where designed but not implemented like the full-scale platform and the custom board are also discussed in this chapter. The simulations performed to validate the design before implementation are also described in this chapter. All the resources needed for this implementation can be downloaded from https://github.com/IvanCanedo/Re-configurableDrivingSimulatorPrototype.git or https://1drv.ms/u/s!AognwflcNk1qgbMkKV8goaVP7kUmBQ?e=IQjAq5

4.1 Simulation

To validate the behaviour of the proposed system, a simulation was performed. The simulation was performed using MATLAB/Simulink, the basis of the simulation is a 3D platform that uses the Simscape complement, the 3D model is shown in Fig. 4.1 in the block called "platform", in this model it is possible to visualize the platform's movement, measure the position of points P_i and measure the position and acceleration of points B_i defined in 2.4. Those latter points are actuated using the output of the control strategy.

This simulation measures the performance of the ADRC and the overall performance of the system with the blocks called “Motor Control performance Analysis” and “Platform Performance Analysis”, the measurement is made by comparing the reference with the output of the system, the performance results will be discussed in section 5.1

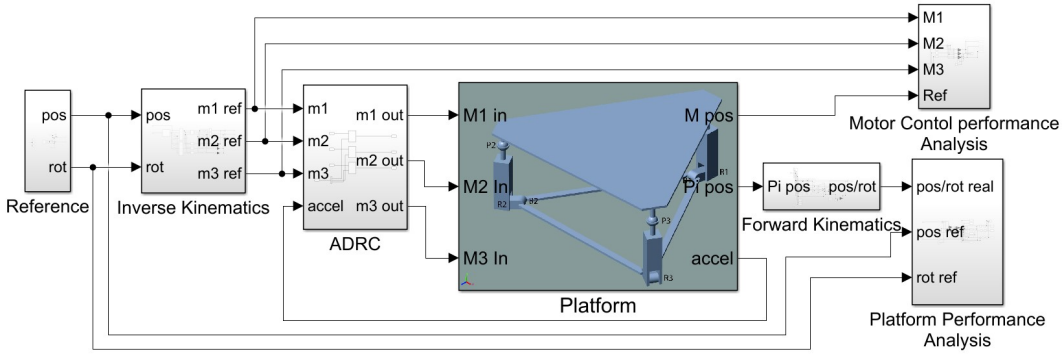

Figure 4.1: MATLAB/Simulink Program

4.1.1 MCA Simulation

To simulate the proposed MCA the IMU signals were obtained from CARLA Simulator and saved into a excel spreadsheet to be used by the Simulink program. A set of test IMU signals were obtained from the CARLA Simulator with the goal of testing the system in different driving conditions, like, acceleration and deceleration, a lane change, a right turn or even a collision.

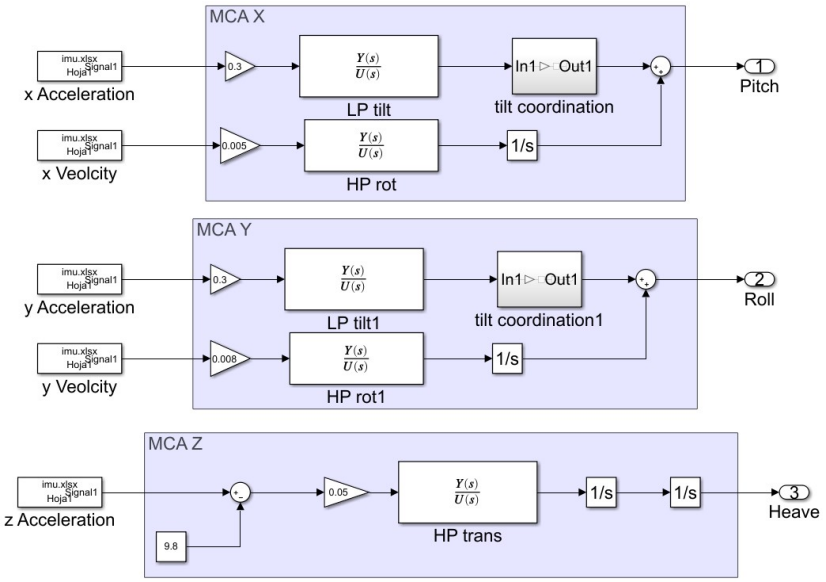

Figure 4.2: Section of the Simulink program that simulates the proposed MCA

Fig. 4.2 shows the section of the Simulink program used to simulate the MCA. in this figura it can be seen than the acceleration and angular velocity en the XYZ axis are accessed from a Excel file. The MCA's filters are defined using a MATLAB script that loads into the worksapce the value of the numerator and denominator of the transfer function, as defined in section 3.3.1 with the values of table 3.1. The gains before the filter were defined experimentally by trial and error using the best and worst case scenario, were the best case scenario is a acceleration and deceleration of the vehicle in a straight line and the worst case scenario is a collision.

4.1.2 Inverse Kinematics Simulation

To simulate the inverse kinematics it's necessary to use the equations defined in section 3.1.1, Fig. 4.3 shows the section of the Simulinks program that simulates the inverse kinematics, the block called "Rot Matrix" performs the calculations defined in (3.5), similarly, the block called "Post Computation" executes (3.6), finally the blocks called "pos2rad M(1,2,3)" execute (3.8) to (3.10). It's worth mentioning the fact that the derivative output of the inverse kinematics is calculated to implement the next step of the simulation that is the ADRC.

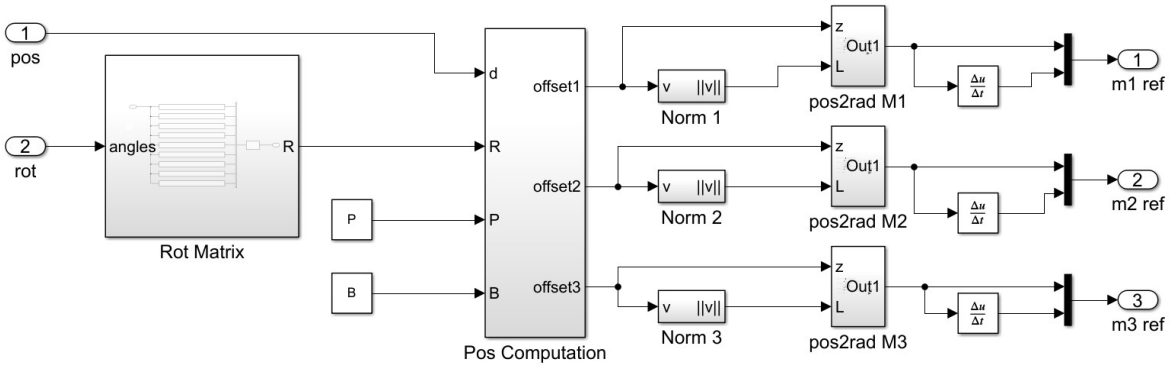

Figure 4.3: Section of the Simulink program that simulates the inverse kinematics of the platform

4.1.3 ADRC Simulation

To simulate the proposed ADRC the program shown in Fig. 4.4 was implemented. The fist step is to simulate the motor model as defined in section 3.2.1, this is done in the "Motor Model" block. The block named "ADRC" calculates (3.20) as defined in section 3.2.2, the "ESO" block performs (3.22), finally the external disturbance is obtained from the platform 3D model generated with the Simscape complement. It's worth mentioning that the ADRC is simulated in continuous-time and not discrete-time because the deisretization technique

defined in section 3.2.3 allows to tune the control in continuous-time and expect a similar behaviour when passed to the discrete-time domain.

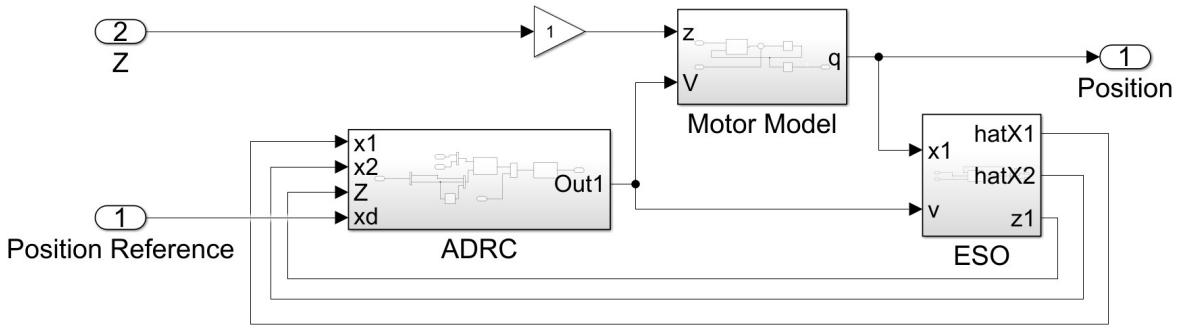

Figure 4.4: Section of the Simulink program that simulates the ADRC

4.2 Motion Platform

To design the PM that is used as the motion platform for the proposed driving simulator, the mechanical designs was built around the mechanical dimensions of the selected motors, the selection process for the motors was based in availability, price and performance. The design process followed the following steps in the order they are listed.

1. Selection of the motors with similar characteristics of motors from commercial-use driving simulators
2. Selection of the bearing an joints from a supplier catalogue
3. Mechanical design of the full-scale platform.
4. Perform a finite element simulation to validate the design
5. Reiterate mechanical design process until the finite element simulation's results are acceptable
6. Design a scale model as similar as possible to the full-scale model
7. Manufacture the scale model

4.2.1 Full-Scale Platform

The designed platform is depicted in Fig. 4.5, the design was implemented in SolidWorks 2019, the 3D models of the motor and the bearings were obtained from the manufacturers website. The technical drawings of all the other pieces will be provided on the GitHub repository

The selected motor for this implementation is the ECM180/050 from Transtecno, this motor is a permanent ferrite magnet, brushed, DC motor with a wormgear reductor, its specifications are the following.

- Torque of 39 Nm
- Speed 38 rpm
- Input power 250W
- Service factor 1.2
- Gear ratio 80
- Nominal Voltage 24V

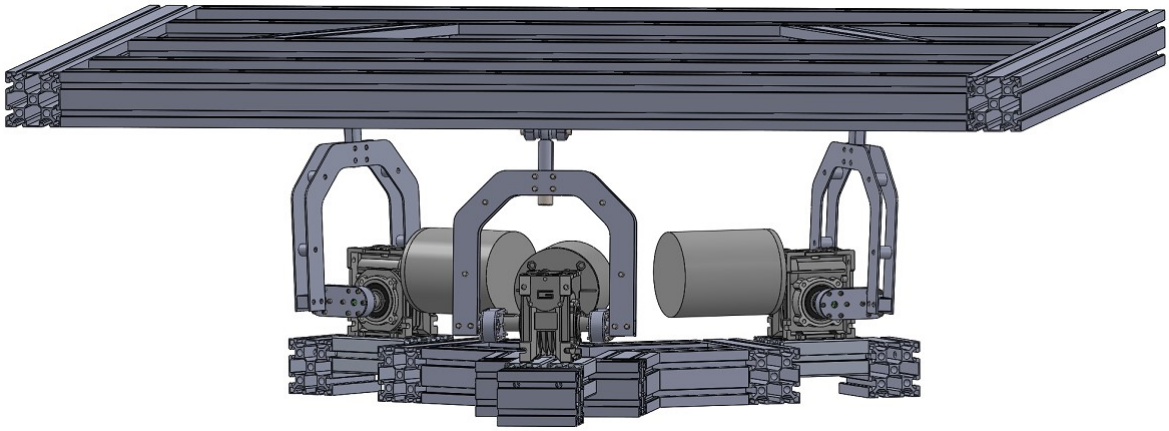

Figure 4.5: Full-Scale platform designed in SolidWorks

If the motor is taken as the active revolute joint of the 3-RRS PM, the passive revolute joint would be implemented as a bearing, to be more precise a self self-aligning ball bearing, this type of bearing was selected due to the misalignment that the bearing will be supporting, the selected product was the 2202 ENT9 self-aligning ball bearing from SKF. The fact that the design inflicts misalignment to the bearing is a not ideal since the change in load to the bearing will cause dynamic misalignment that is not technically supported for any type of bearing, therefore, the implementation of a solution for this issue is left as future work.

For the passive spherical joint a 2-piece stainless steel spherical joint was selected, the maximum miss-align angle of this kind of joints limit the maximum inclination that the platform can have, nevertheless the load that this kind of joints can support in comparison with their magnetic counterparts is the driving factor for its selection.

The base of the platform and the moving platform are designed to be manufactured from extruded aluminium rails, this decision was taken for two main reason: 1. to lower the weight of the moving platform 2. to make the reconfiguration of the platform easier. Extruded aluminium rails have a excellent weight to strength ratio and are really easy to work with, rails can be connected with screws and connectors using simple tools.

The crank is one of the most important design parameters, Fig. 4.6 shows the designed crank where the distance between center defines the amount of weight that the platform will be able to lift as discuses in section 3.1.3, also, this magnitude is the variable L defined in section 3.1.1, therefore it also rules over the maximum angle that the platform will be able to rotate. The proposed magnitude of the distance between centers is 70mm where, according with (3.15), would mean that every motor can lift up to 56.8Kg, meaning that the whole platform would be able to lift 113.6Kg approximately. Using Solidworks to validate the maximum angle that the platform would be able to rotate, it was determined that the maximum angle is 11° with 0 Heave.

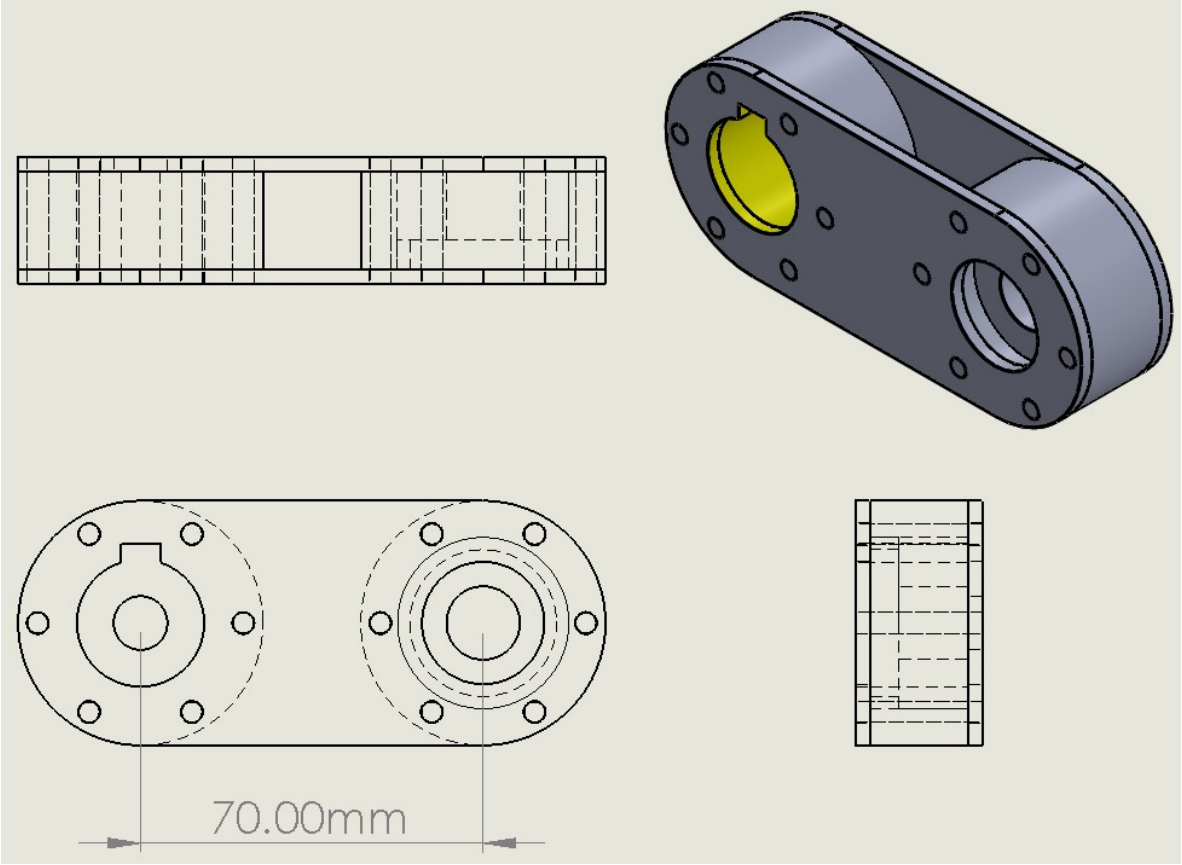

Figure 4.6: Technical drawing of the proposed crank

The selected materials to manufacture the pieces are the SAE 4140T and the ASTM A36, the selections was based on availability and price. The SAE 4140T is sold in bars, therefore it will be used to manufacture the pieces that net to be machined with chip braking tools. The ASTM is sold in steel plates and it will be used to manufacture the pieces that can need to be plasma cut.

4.2.2 Scale Platform

The designed scale platform is depicted in Fig. 4.7, this model was designed to bee as similar as possible to the full-scale model, the motor used is the DG01D-E from Sparkfun, that is a permanent magnet, brushed, DC gearmotor, the documentation for this motor is scarce, therefore the parametric uncertainty is considerable, the only information available is the following. Since the torque and other parameters of the motor doesn't matter in the scale motor a low price motor like this one was selected.

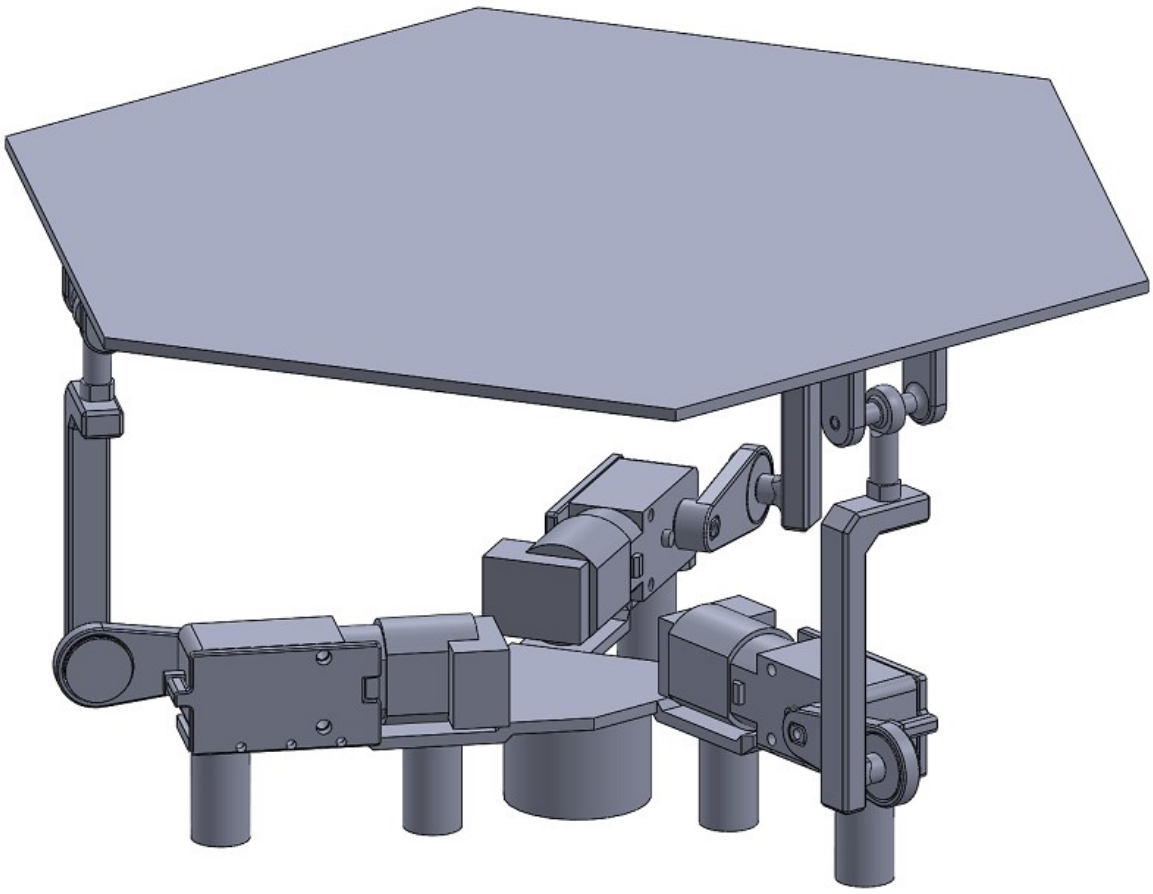

Figure 4.7: Caption

- Nominal Voltage of 9V
- Gear ratio 1:48
- Speed at 4.5V 90RPM
- Hall-effect incremental AB encoder

To emulate the passive joint of the full-scale model, a M3 2-piece stainless steel spherical joint was selected for the spherical joint, for the passive revolute joint, a 3D printer's 5x16x5 ball bearing spear part was used. The rest of the pieces that were used to implement the scale platform were 3D printed in white PLA. To assemble the model it was necessary to use 6 M3 screws, 9 M2 screws and nut and superglue. Finally, the moving platform was laser cut from 3mm acrylic. The implemented scale model will be depicted in section 5.3.2.

4.3 CARLA Client

To implement the CARLA client, the Python Application Programming Interface (API) provided by the CARLA team was used, this API is imported to a Python script using a .egg file that is added to the system path. The documentation of all the methods available are detailed in the CARLA documentation web site [53]. The CARLA team also provides examples and tutorials, the proposed client script is based on the manual control example that was licensed under the terms of the MIT license. Copyright (c) 2019 Computer Vision Center (CVC) at the Universitat Autonoma de Barcelona (UAB). All modifications to the original software will be provided and explained in Appendix B

The CARLA client is based in Object Oriented Programming (OOP) and it makes use of 11 classes defined in the script, Fig. 4.8 shows the behavioral flow diagram of the script when is executed in python 3.7. The requirements for this software to work are numpy version 1.18.4 or later, pygame and python version 3.0 or later.

In the flow diagram of the CARLA client it can be seen that, basically, the script initializes and creates all the necessary assets to start the 3D simulated environment and then stats to execute a loop that updates the simulated world in function of the inputs given. Finally, it exits if the exit command is entered with the keyboard, after witch, it destroys all the created assets an finishes the simulation, this is important since the created actors and sensors remain in the CARLA server if they are not explicitly destroyed.

Entering in more detail about the CARLA Simulator, the project includes several maps, weathers, lights, sensors and blueprints. The "world" can be described as the state of the server an can be modified by a client, the world contains the map, actors, weather, lights along with other features. The map, weather an light do what their names imply they do

and set those features of the server. On the other hand the actors are less intuitive, "Actors not only include vehicles and walkers, but also sensors, traffic signs, traffic lights, and the spectator." [53]. When an actor is created, it is selected from a library of blueprints that define they proprieties, they are called blueprints since two actors created with the same blueprint can be different from each other because every blueprint have parameters than can be defined by the user.

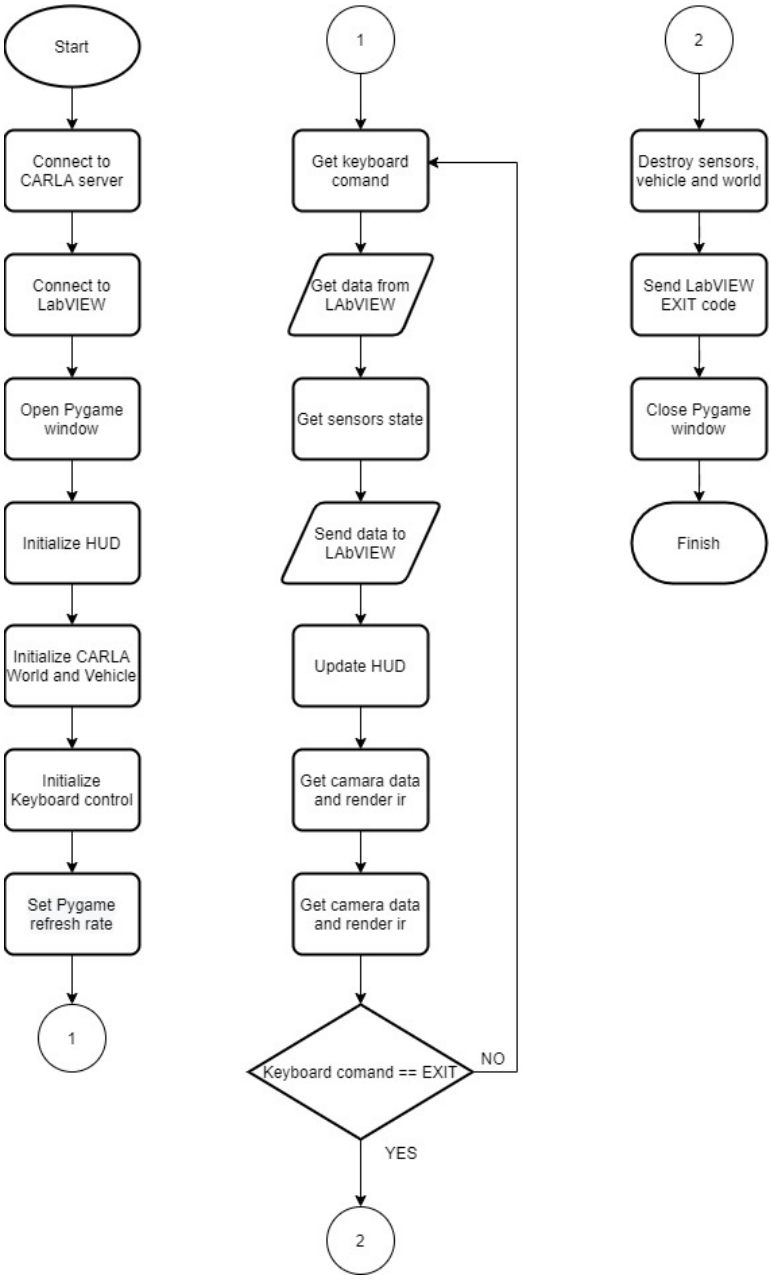

Figure 4.8: Flow Diagram of the CARLA client

The available types of actors at this moment are: Sensors, Spectator, Traffic signs and traffic lights, Vehicles and Walkers (pedestrians). Every kind of actor has at leas one type of blueprint associated to it. All the types of actors play a crucial roll in the performance of the simulation, nevertheless, the most crucial feature provided for an actor that is relevant for this work is the IMU sensor, this sensor comprises an accelerometer, a gyroscope, and a compass. To use this sensor, an actor of the class Carla.IMUMeasurement is created an attached to a vehicle actor.

4.4 LabVIEW

LabVIEW is not only the software that communicates with the external hardware, it is also the "moderator software" defined in section 2.6, this means that before the simulation starts this visual instrument must perform the "Modeling" and "Pre processing" steps as depicted in Fig. 2.4. The LabVIEW's built-in tools make the programming of such interface easy and fast, therefore, any new feature can be added to the program with a relatively small development time, thus fulfilling the main goal of a re-configurable driving simulator.

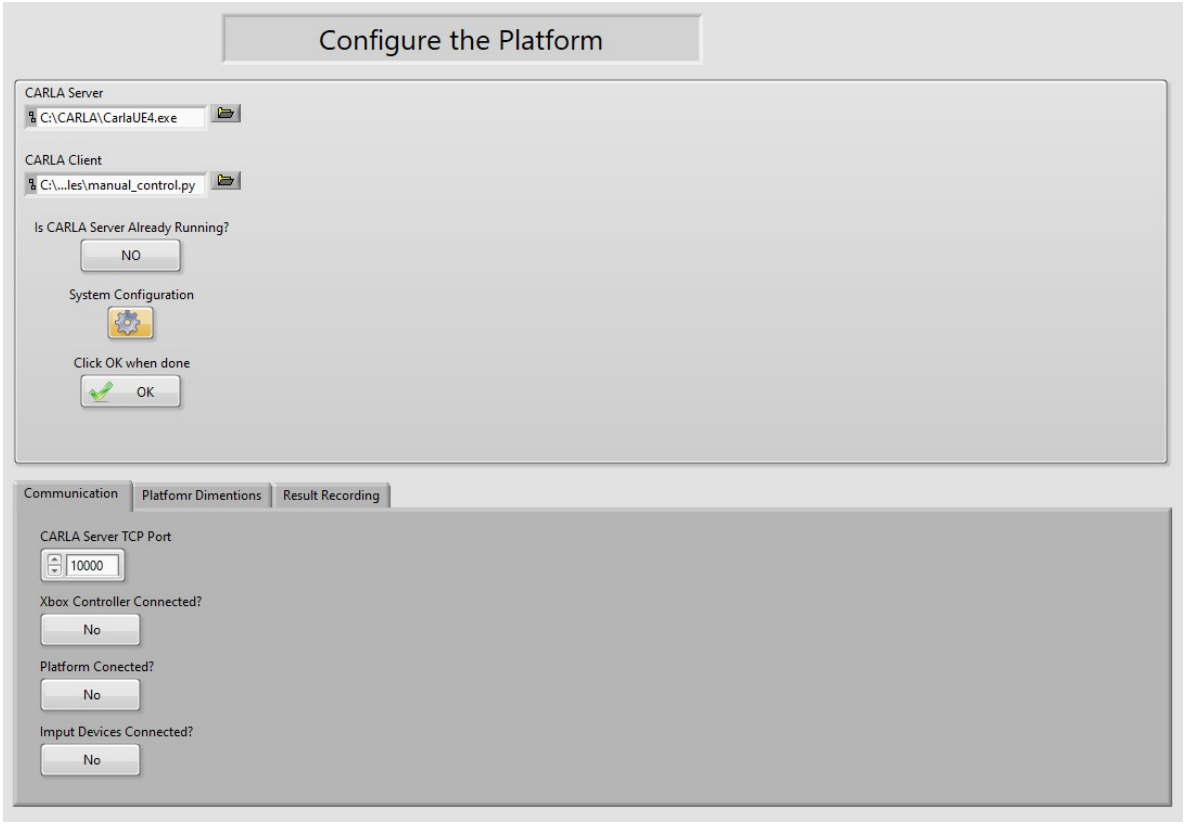

Figure 4.9: LabVIEW's Visual Instrument Modeling screen

To implement the Modeling process, a interface that allows the user to modify the simulation parameters was implemented, the displayed screen is shown in Fig. 4.9, in this screen the user can launch the simulator with the default values or press the System Configuration button to display the configuration interface, in this tab control the user is capable of select and configure the peripheral devices connected. If the motion platform is connected, after activating the device it's possible to move each motor individually to set the home position in case that the embedded system looses position for any reason or to calibrate the system, this feature is shown in Fig. 4.10.

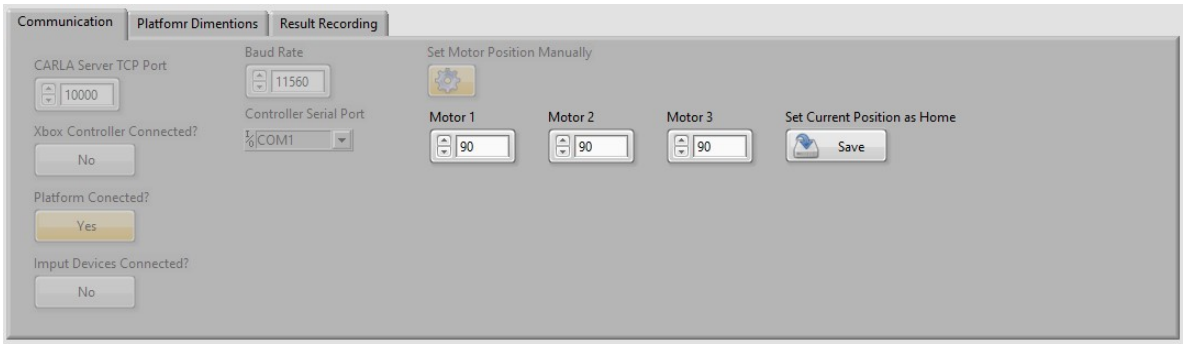

Figure 4.10: LabVIEW's Visual Instrument platform calibration feature

In the Modeling process of the LabVIEW Visual Instrument (VI) it's also possible to configure the platform dimensions that will be used to perform the inverse kinematics, this feature is present in the Platform Dimensions tab. Similarly in the Result recording tab is possible to activate and configure the recording of variables during the simulation run time. Finally when the Start Simulation button is activated, the program will check if every configuration is valid, if any issue is found an error log will be presented to the user detailing the source of the error.

When the Start simulation button is activated and the program detects no error the Modeling phase is finished and the pro processing phase starts, this part doesn't require any user interventions therefore the user will see a waiting screen that shows the status of the process. During this phase the visual instrument uses CMD commands to execute the CALRA server and client. According with the definitions of section 2.6, the moderator software should be able to configure the simulator test scenario. This is done in the Python code, therefore, even if the test scenario can be configured editing the Python code, the moderator software should be able to have some control over things like the maps, spawn points, weather etc. This feature is left as future work.

After the pre processing is done and the CARLA Simulator is running, the visual instrument enters the "Simulation Run Time" phase. In this moment the user see the default screen

shown in Fig. 4.11, in this windows its possible to control the vehicle using the LabVIEW controls displayed.

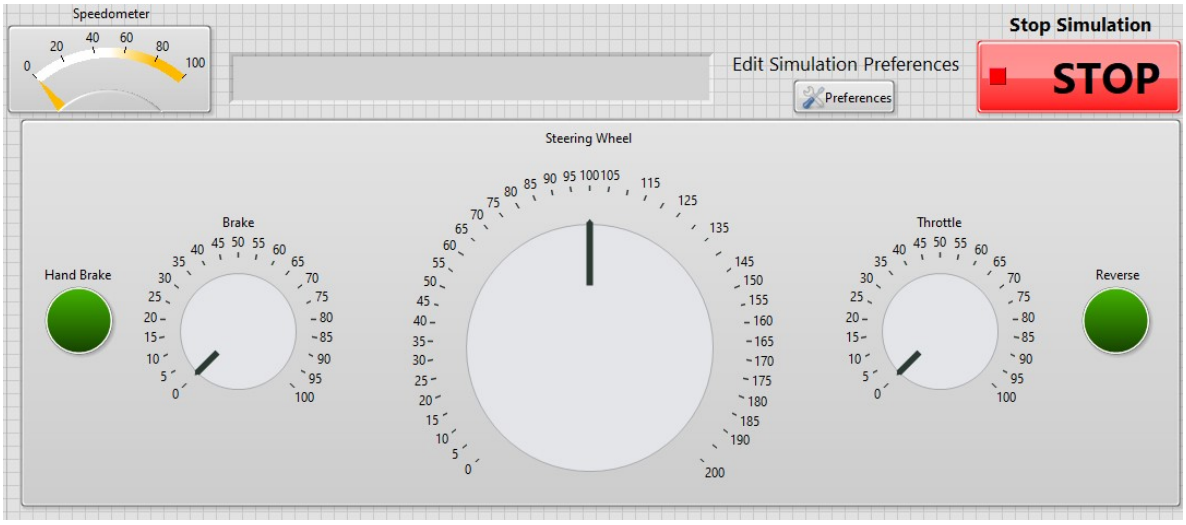

Figure 4.11: LabVIEW's Visual Instrument default Run Time screen

To activate the other input devices, its necessary to activate the Edit Simulation Preferences button. This button opens a tab control that allows to configure several simulation parameters, for example, in the Input Devices tab the input device can be selected, in the IMU tab the filtering of the IMU signals can be configured, and in the MCA tab it's possible to change the gains to tune the MCA as described in section 3.3. This feature is shown in Fig. 4.12. In this screen it's also posible to visualize in real time the varialbes or the simulation lie the IMU signals of the vehicle, the platform desired position and, if the platform is connected, the motor's angular position. Ideally in this phase the user should be able to modify the scenario by changing the weather, adding pedestrians, etc. This feature is left as future work.

Finally, when the simulation run time ends, the post processing phase executes, this phase has the responsibility to close the client window, destroy the created actors, organize the recorded data and save it in a file to be analysed, in this implementation the recorded data can be saved en several formats that can be chosen in the modelling phase, the default option is a excel spreadsheet.

The currently available features in the visual instrument are the bare minimum features a re-configurable driving simulator needs, nevertheless, the current program structure is designed to be base line for any future development. The LabVIEW program's block diagram window will be detailed and explained in Appendix C

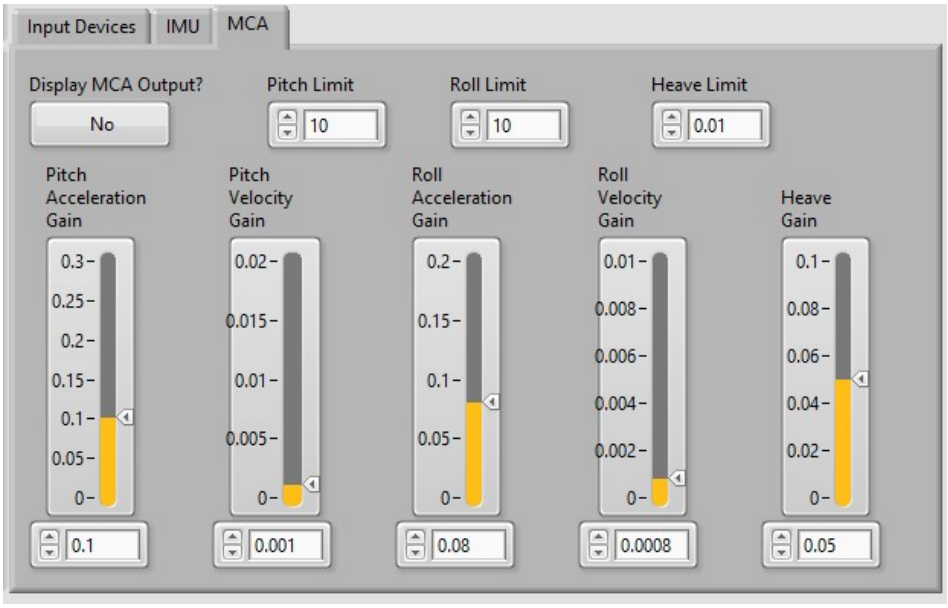

Figure 4.12: LabVIEW's Visual Instrument Edit Simulation Preferences tab control

4.5 MicroPython

To implement the control strategy it's necessary to measure the motor's position and control the motor's direction and speed. To do so, the position of the motor is measured using a AB encoder attached to the motor shaft, to control the direction and speed of the motor a H-bridge is used, where the enable pin is controlled with a Pulse Width Modulation (PWM) signal. The flowchart to implement is shown in Fig. 4.13.

The sampling rate is defined by a timer, it must be configured to generate an interruption at least $\frac{1}{10}$ of the time that the motor need to reach its nominal velocity, therefore, the time to calculate the control signal must smaller than $\frac{2}{3}$ of the sampling rate. The algorithm will be implemented in the STM32F407VGT6, as stated in sction 3.5, with a clock frequency of 168MHz for the CPU. The schematic diagram of the implementation is shown in Fig. 3.12. This implementation needs one timer to control the sampling rate, another timer to measure the encoder and a one more to generate the PWM signal for each motor, this means that the system must have at least 7 timers to control 3 motors.

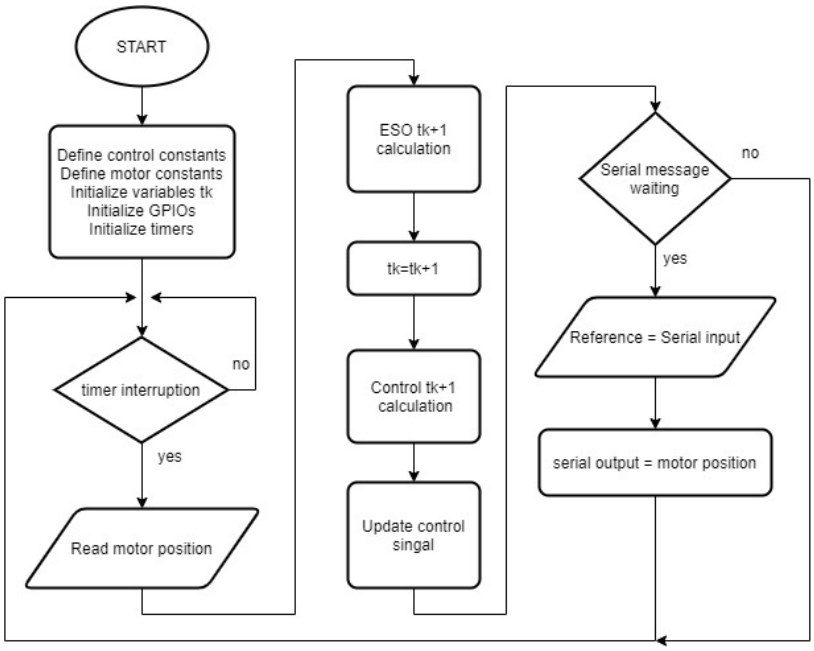

Figure 4.13: Flow diagram of the ARDC implementation in MicroPython

MicroPython offers a the pyb library that contains the classes and methods to control the hardware peripheries of the MCU, the pin class allows to create objects that can control the GPIOs, the class UART allows to manage the UART/USART communication ports, finally the class Timer allow to set timers, define the interrupt handlers and configure the timer's alternate functions, such as the AB encoder mode or PWM. The STM32F407VGT6 has 14 timers, this means that the maximum number of motors that can be controlled are 4, nevertheless, the encoder mode is only implemented in channels 1 and 2 of the timers, which means that the timers with less than 2 channels like the TIM10, 11, 13, 14, 6 and 7 can't be used to implement this feature. Therefore, due to pin availability, the maximum number of motors that can be controlled narrows down to 3.

Regarding the communication with the PC, the system must update the control's reference using the information received thru the UART port, to do so, the UART's buffer is set to a known length and when the program finishes updating the control signal, checks if there are a full message waiting in the buffer. This is a polling technique and it was chosen over a interrupt based technique for the following reasons. The control's sampling frequency is much faster than the communication rate and the control signal calculation is extremely time sensitive. Therefore its better to have the incoming data waiting for a short period of time than interrupting the control calculation process. The full code will be provided and explained in Appendix D.

4.6 Electronics

Along side the mechanical and software components, some electronic devices were needed, therefore, the following sub sections contain the implementation's description of the electronic components used in this development.

4.6.1 Input Devices

To control a vehicle in CARLA simulator three floating point variables are needed and two Boolean variables, the floating point variables are used to control the throttle, brake and steering wheel and the Boolean variables are used to set the reverse and handbrake on/off. This information is provided to the CARLA client thru the TCP protocol form LabVIEW, by default those variables are controlled by the native LabVIEW's visual controls as shown in Fig. 4.11. To implement the real world counterpart of the LabVIEW's knobs and switches it's necessary to implement an electronic device that acquires analog signals that represent the throttle, brake and steering wheel and digital signals that represent reverse and handbrake, finally it needs to send the information to LabVIEW to be sent to the CARLA client. An extra feature added is and emergency stop button that will end the simulation the moment its activated.

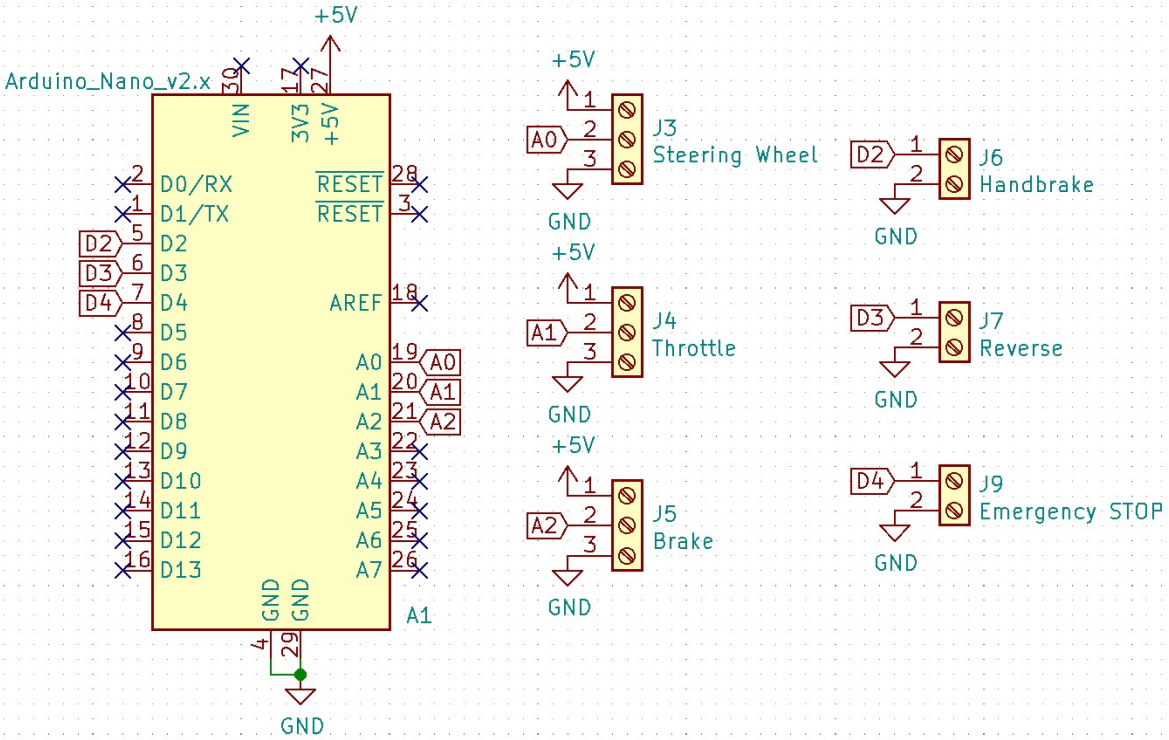

Figure 4.14: Schematic of the electronic capture card for the input devices.

The implemented system consist of an Arduino Nano, three potentiometers and three switches connected as depicted in the schematic diagram shown in Fig. 4.14, a Printed Circuit Board (PCB) was manufactured following this schematic. The selected analog devices are 10 turn 10KΩ linear potetiomiters model 3590S from Bourns. For the reverse and handbrake, regular two-positions switches were selected, finally, a big red push button was selected for the emergency stop.

For the steering wheel implementation it was necessary to implement a gear transmission between the steering wheel and the potentiometer. The implemented linear gear ratio is 4:10 yo maintain all components 3D printable. The potentiometer was soldered to a PCB to ease the connections to the wires and fixation to the base of the system. The full steering wheel system 3D model is shown in Fig. 4.15, in this model the white pieces were 3D printed in PLA, the blue pieces are the potentiometer PCB and component, the grey component is a ball bearing, and finally, the red piece was laser cut in MDF triply. The system set-up is meant to be used in a controlled laboratory-like environment since the materials are ripe for failing or breaking.

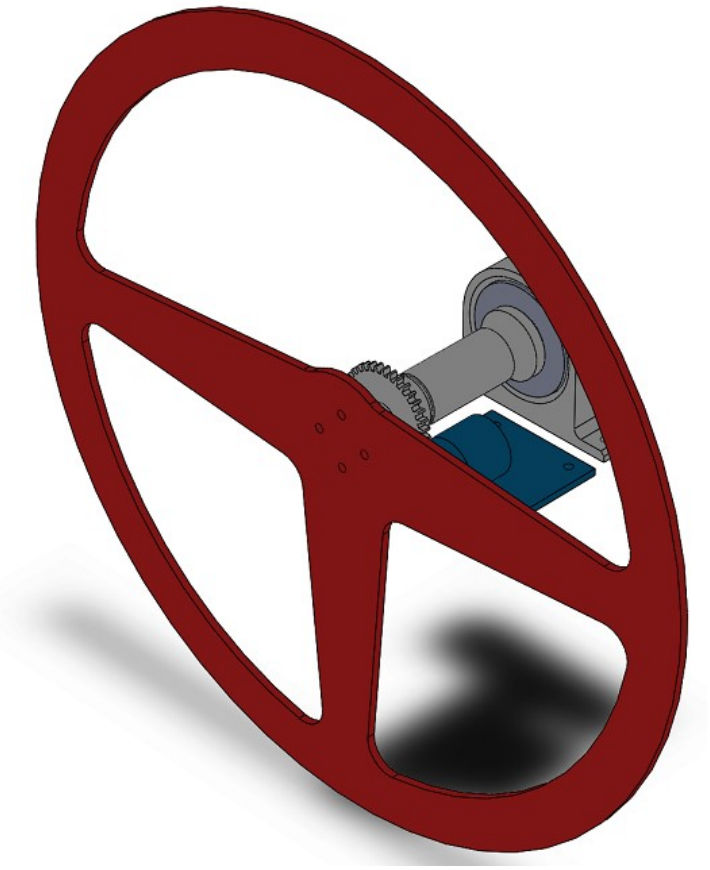

Figure 4.15: Steering wheel 3D model

Similarly to the steering wheel, the pedals were designed in a similar fashion, the same linear-gear transmission was used to transmit the pedal's rotation to the potentiometer. In Fig. 4.16 the pedal's 3D model is depicted, the color scheme in the illustration is the same as in the steering wheel illustration.

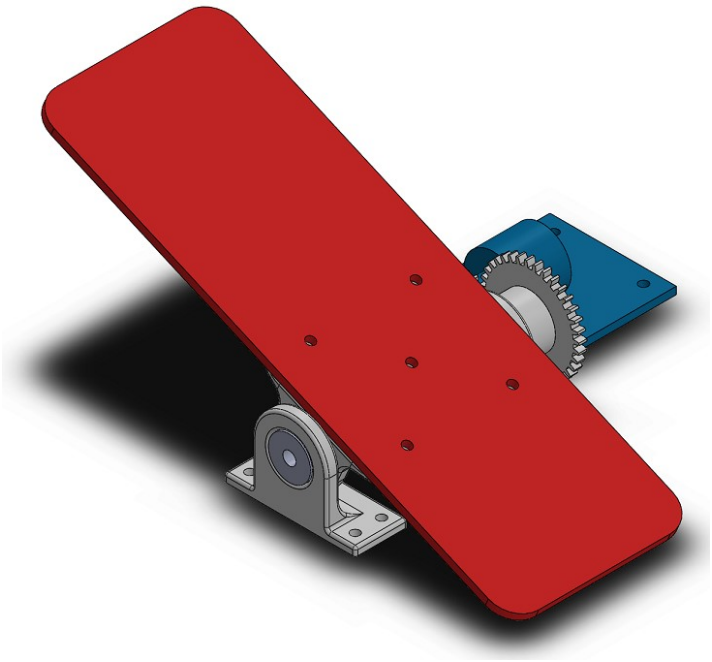

Figure 4.16: Pedal 3d model

4.6.2 Scale Platform Electronics

To implement the system described on section 3.5.2, it was necessary to select a commercial development board that fulfills the requirements. While the "STM32F4DISCOVERY Discovery kit" manufactured by STM satisfies the requirements, due to accessibly on the local market, the final selection was a development card of the name "Ophyra" manufactured by a company named "Intesc". Fig. 4.17 shows a stock photo of the Ophyra development card, it's specifications can be found on the manufacturer website.

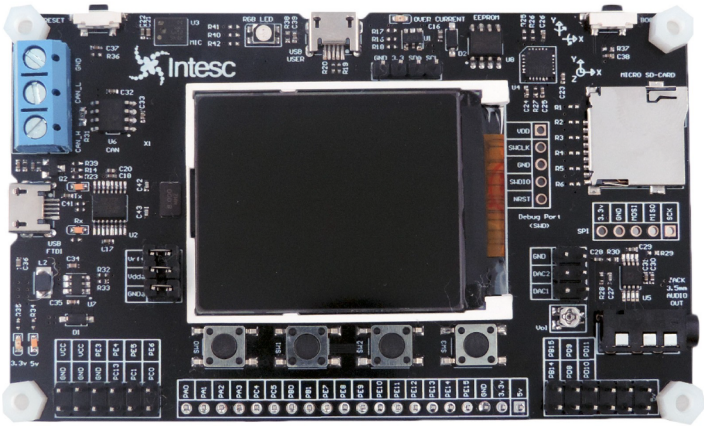

Figure 4.17: Ophyra by Intesc powered by STM32F407VGT6

Since the card's pins are male headers, it was necessary to design an manufacture a PCB to make the connections as shown on Fig. 3.12. Furthermore, Fig. 4.18 shows the schematic diagram of the PCB. Connectors J9, J7 and J8 are meant to connect to the Ophyra development board, unfortunately for this design, some of the connector of the card are facing upwards, thus, it's impossible to connect them directly to a PCB located below the Ophyra. To solve this issue the pins will be connected using wire jumpers to avoid the risk of damaging the Ophyra while disordering the male pin headers than are facing the wrong direction.

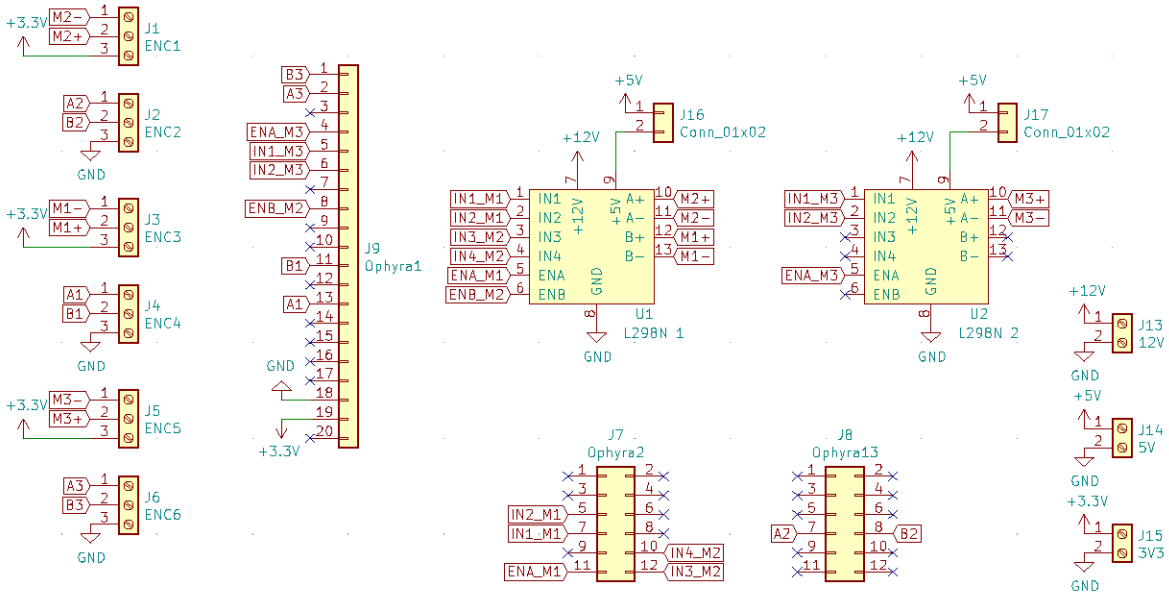

Figure 4.18: Schematic diagram of scale platform embedded control´s PCB

Fig 4.19 shows the layout of the PCB. The PCB is a two layer design due to the relatively low complexity of the design.

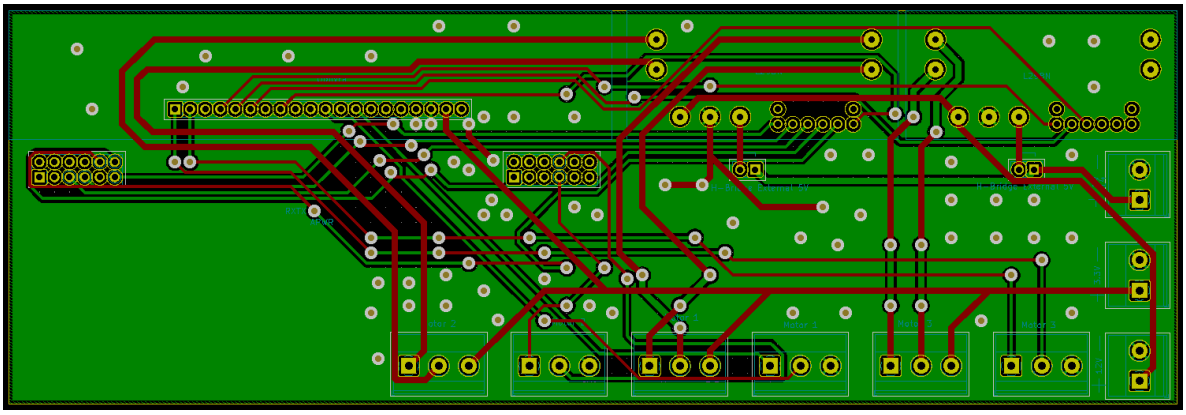

Figure 4.19: Layout of scale platform embedded control´s PCB

4.6.3 Custom Board

As part of the full-scale proposal a custom board was designed to fulfill the role shown in Fig. 3.11. The Schematic and PCB layout were designed in KiCad, all components were selected from the Digikey Catalog. The board's features as shown in the block diagram of Fig. 4.20. In this section the board's features will be detailed briefly, for a detailed explication refer to Appendix A.

The center piece of the custom board is the STM32F407GV, the manufacturer provides a series of datasheets, application notes and design references, upon reviewing those, all the necessary hardware has implemented. As an example, the MCU needs to be powered with 3.3V filtered by 10 decoupling capacitors of different values and requirements, the capacitors also need to be positioned as close as possible to the MCU's power pins. This IC also needs an extra filtering stage for the analog reference.

To program the MCU, the manufacturer offers tools that communicate with the IC thru the SWD protocol, but it also offer an embedded boot loader that makes possible the programming of the MCU thru some of the embedded communication protocols, like the USB On-The-Go (OTG) Full Speed (FS), USART and CAN. In the designed board the programming is possible using SWD, USB OTG FS and USART thru the implemented USB/USART converter.

Regarding the powering of this board, the design takes into consideration 3 possible options to power the board, the USB OTG connector, the USB/USART connector and a barrel jack for external +6V to +34V power source; to power the board it's possible to connect just 1 or all 3 of the power sources. Since this board is meant to work in a harsh environment, some protections were taken into consideration for the power stage of the design. The designed

protections are, reverse polarity, over current and Electrostatic Discharge (ESD) protection in the USB OTG data lines.

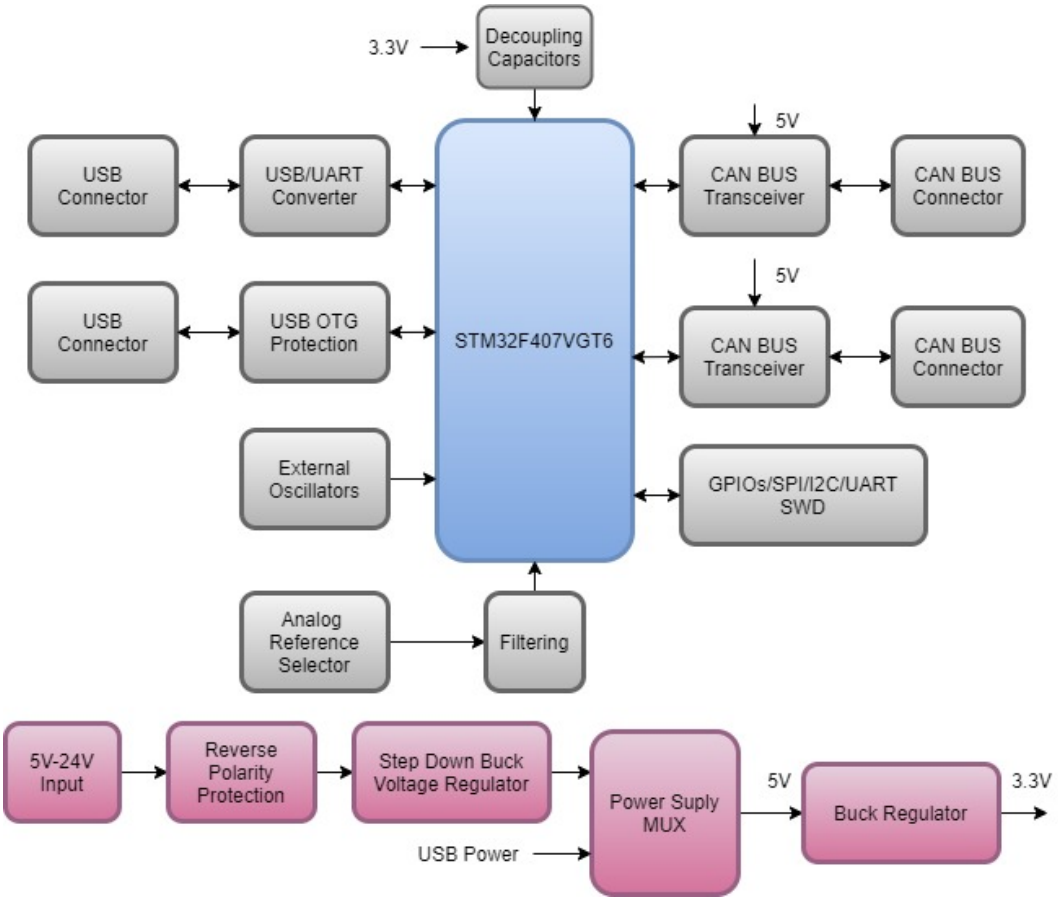

Figure 4.20: Custom board block diagram

One of the most critical components is the external oscillator, the selection and design of this component requires the calculation of variables like the load capacitance, the load resistance, the oscillator transconductance and a very delicate layout topology. The implemented external oscillators in this design is an 8MHz oscillator for the HSE (High Speed External) and a 23.768KHz oscillator for the LSE (Low Speed External).

The communication ports that are available in this design are, USB OTG FS, UART3 by a USB/USART converter and by male pin headers, UART2 and UART6 by male pin headers, SPI1 and SPI2 by male pin headers, I2C1 and I2C2 by male pin headers with a 4.7KΩ pullup resistance, SWD by male pin headers and finally CAN1 and CAN2 by terminal blocks. Regarding the CAN BUS, while the controller is embedded in the MCU, the transceiver is implemented in the board and a 120Ω can be connected to the CAN H and CAN L lines with a jumper. Finally in Fig. 4.21 it can be seen a 3D view of the designed custom board.

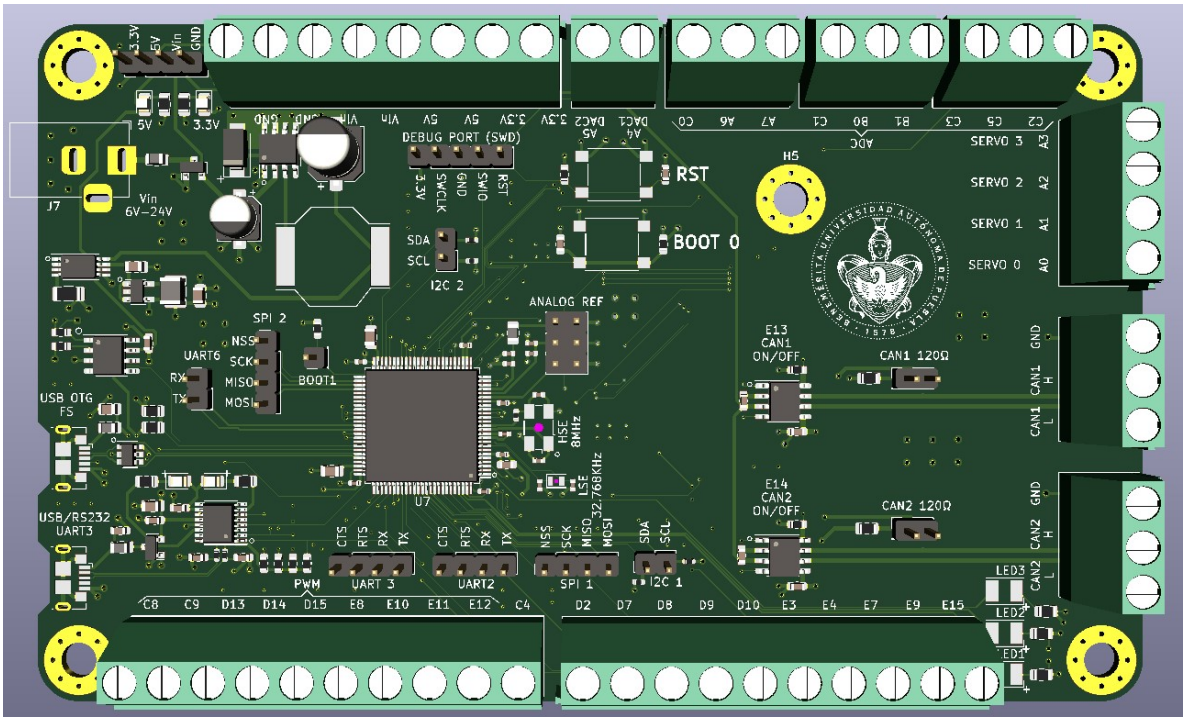

Figure 4.21: Custom board's PCB 3D View

Chapter 5

Results

In this chapter the obtained results will be presented. Starting from the simulation results where the obtained plots are shown and briefly discussed. The following results are the performance of the embedded ADRC implemented in MicroPython. Finally, the complete re-configurable driving simulator prototype will be presented. A more in depth discussion of the obtained results will be presented in chapter 6.

5.1 Simulation Results

The simulations that were performed to validate the proposal and implementation are presented in this section. First, the simulations that were preformed in MATLAB will be discussed in section 5.1.1, the results presented have the goal of validating the MCA, inverse kinematics, ADRC and the overall performance. After that, in section 5.1.2, the simulations results used to validate the mechanical design of the full-scale platform are presented.

5.1.1 MATLAB Results

The simulation was performed using MATLAB/Simulink, the basis of the simulation is a 3D platform that uses the Simscape complement, the 3D model is shown in Fig. 4.1 in the block called "platform", in this model it is possible to visualize the platform's movement, measure

the position of points P_i and measure the position and acceleration of points B_i. Those latter points are actuated using the output of the control strategy.

The overall structure of the simulation is shown in Fig. 4.1, the reference position of the platform is being generated from an Excel sheet that contains the reference position of a IMU signal obtained from the CARLA simulator. Two sets of IMU signals were used to perform the simulation, a statr ans stop maneuver, were the vehicle stats to accelerate and then stops using the brake. The other maneuver is a 90° right turn, where the vehicle stats to accelerate, then reduces the speed and finally, performs a right turn.

The control strategy was simulated using the motor model, the ADRC control and the ESO; at the same time the measurement of the acceleration in the joints is compared with the ideal acceleration of the motor and that difference is added as a disturbance. The overall performance of the system is being measured comparing the input reference position (roll, pitch and heave) with the measured position of the platform. The platform position is measured calculating the forward kinematics.

To visualize the systems performance, Fig. 5.1 shows the signals for the start and stop maneuver. This figure slows the IMU signals and the overall platform performance, were the signals labeled as "Real" are the measured platform position and the signals labeled as "Ref" are the output of the MCA. In that figure it can be seen that the platform tilts when the acceleration starts and slowly returns to the home position as the vehicle reaches constant velocity. When the vehicle stats braking, the platform tilts in the opposite direction and slowly returns to the home position once the vehicle is stationary. In this maneuver, the only degree of freedom used is the pitch, since the only forces experienced by the vehicle are in the X axis.

For a maneuver that requires more than one degree of freedom, a right turn was simulated. Fig. 5.2 shows the behaviour of the system, the IMU signals and platform position are plotted in the same way that in the start and stop maneuver. In a 90° turn, the vehicle experiences forces in more than one axis. Similarly to the run and stop, when the vehicle starts to accelerate, the platform tilts in the pitch direction ans slowly returns to the home position and when the vehicle decelerates, the platforms tilts in the opposite direction. In this maneuver, when the vehicle turns, the platform tilts in the roll direction and returns to the home position, nevertheless, it can be seen around the second 23 that the platform makes a transitory motion. This transitory motion obeys the angular velocity experienced when the vehicle stops turning and returns a linear movement. It can also be appreciated a big angular velocity experienced in the Z axis, this signals gets ignored since the proposed 3DoF configuration can't replicate it, to recreate an angular velocity in the Z axis the Yaw axis is needed.

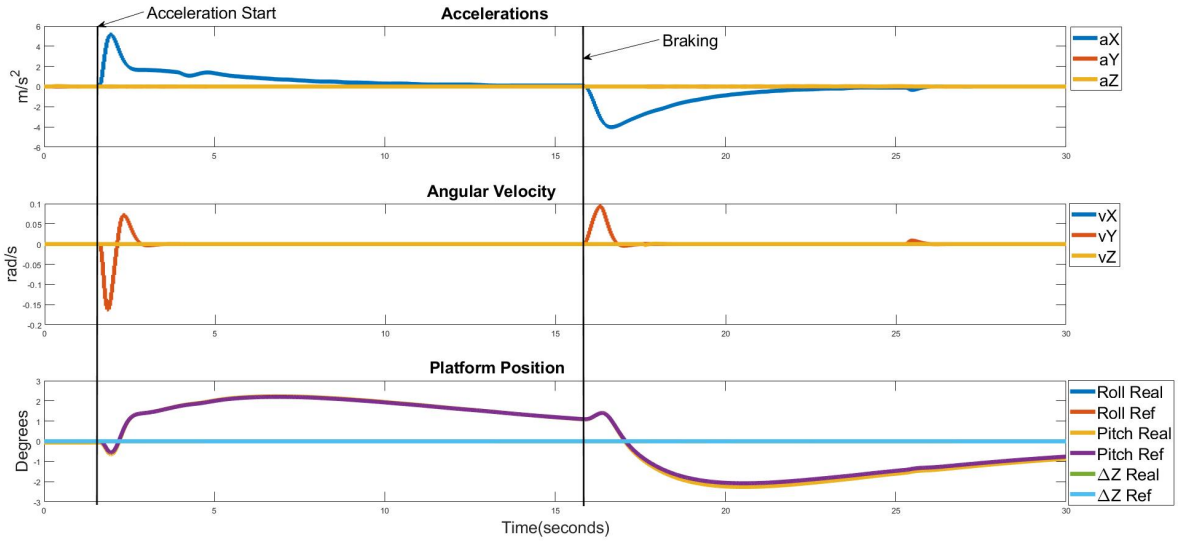

Figure 5.1: Start and Stop

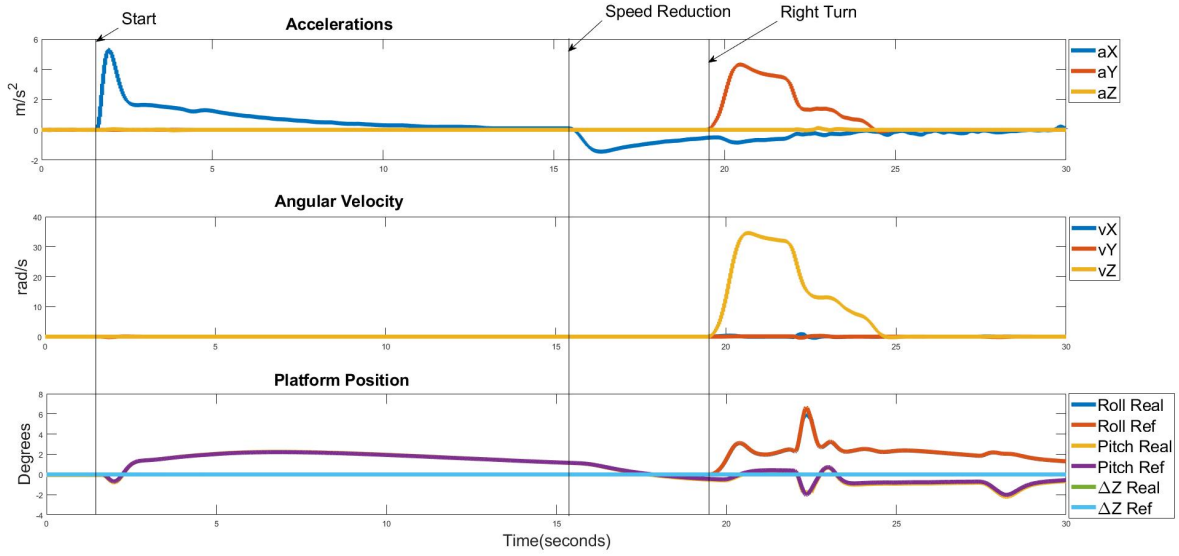

Figure 5.2: Right Turn

In Fig. 5.1 and Fig. 5.2 the platform's reference and actual position is shown, nevertheless, it's difficult to appreciate the performance of the platform with just that information. Therefore, to visualize the performance of the system quantitatively, the performance index ISE (integral squared error) is calculated The ISE index is used to compare the performance of the system with other proposals. The proposed system is compared with the ISE index

of the system changing the ADRC with a PID controller, Fig. 5.3 shows the evolution of both indexes.. To obtain the PID performance, the controller was tuned with the PID Tuner provided by MATLAB/Simulink and the disturbance of the system was multiplied by several orders of magnitude with the goal of simulating a loaded platform. Therefore, while the inverse kinematics also plays an important role in the performance of the platform, the main source of error comes from the control strategy and the disturbances affecting the system.

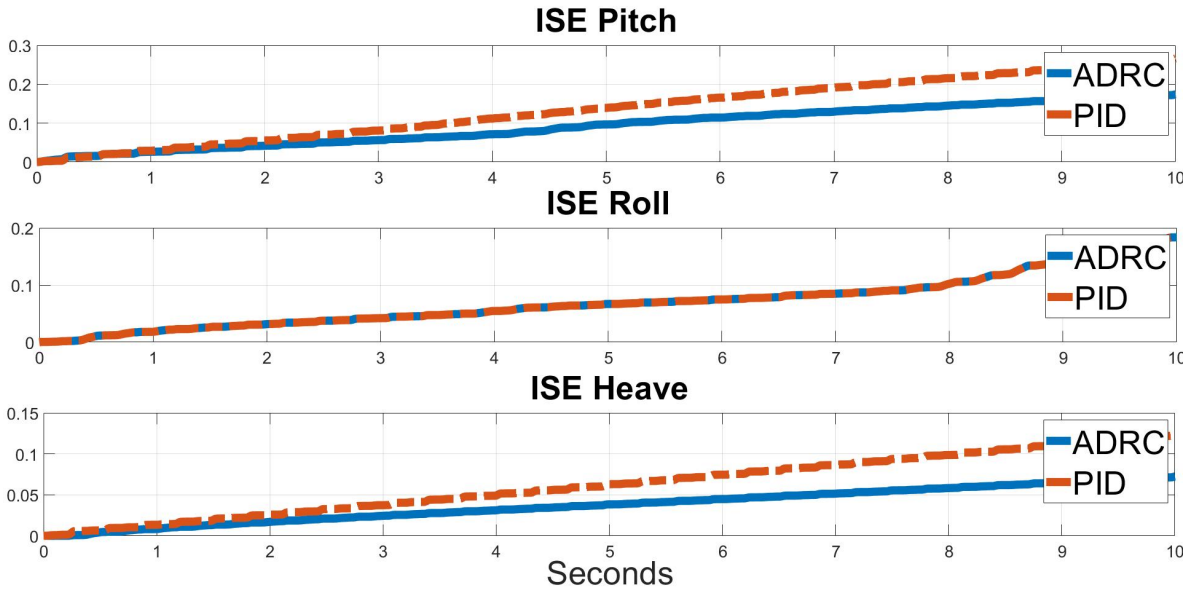

Figure 5.3: Platform's performance ISE Index comparison

Regarding the mathematical model of the platform, the RRS configuration, compared with the SP configuration, exchanges workspace for speed, also, given the non-linearity of the system, for the model to remain relevant, the angle (λ) must not get near 0° or 180°, where the model reaches a point where it fails completely, therefore, maintaining the system within 10° and 170° is vital.

The behaviour of the simulated motors was also tested for the star-stop maneuver, Fig. 5.4 shows the plot of the position and speed of all three motors during that maneuver. It's worth noticing that the motor's speed varies over time depending of the rate at which the position changes on any given time interval. Therefore, the maximum value of angular speed on this plot is the minimum speed for a motor to track this specific maneuver with the current MCA values. For the full scale platform implementation it would be necessary to simulate a maneuver the worst case scenario that is meant to be implemented. With that, the minimum speed of required for the motors could be quantitatively calculated.

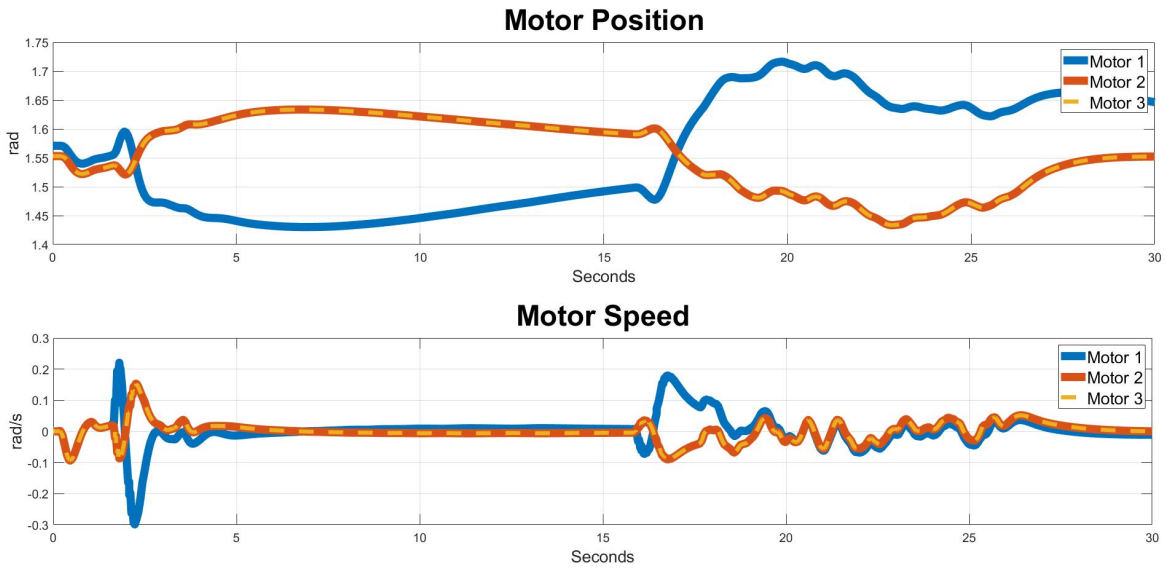

Figure 5.4: Platform's performance ISE Index comparison

5.1.2 Finite Element Simulation Results

The finite element simulation was performed in SolidWorks Simulation 2019. A static analysis was performed were the motor shaft was fixed and a load of 150Kg was applied. It can be seen in Fig. 5.5. the output of the strain test (VonMises) were the color red is set to be the yield limit of the weakest steel of the design (ASTM A36).

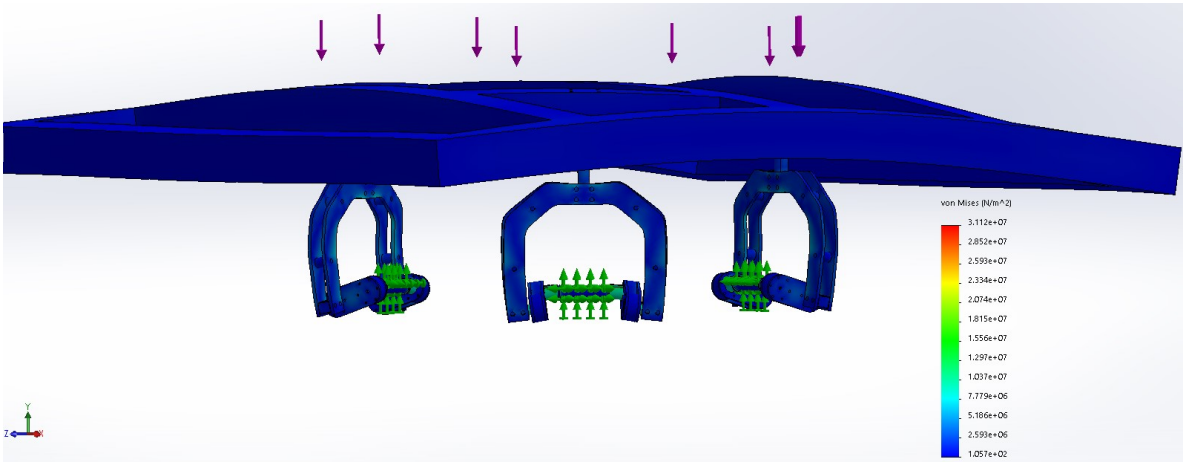

Figure 5.5: Full-Scale Platform's VonMises analysis

To identify the weakest points of the structure SolidWorks Simulaton provides a test tat identifies the sections that are carrying most of the load. The results of such analysis are

shown in Fig. 5.6. were it can be seen that the such sections are the joint with the motor shaft and the rod at the end of the crank.

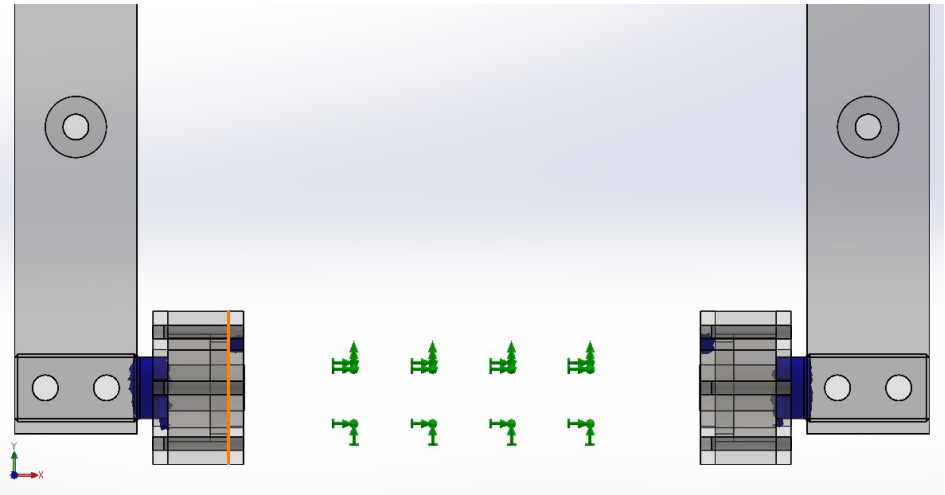

Figure 5.6: Sections carrying most of the load

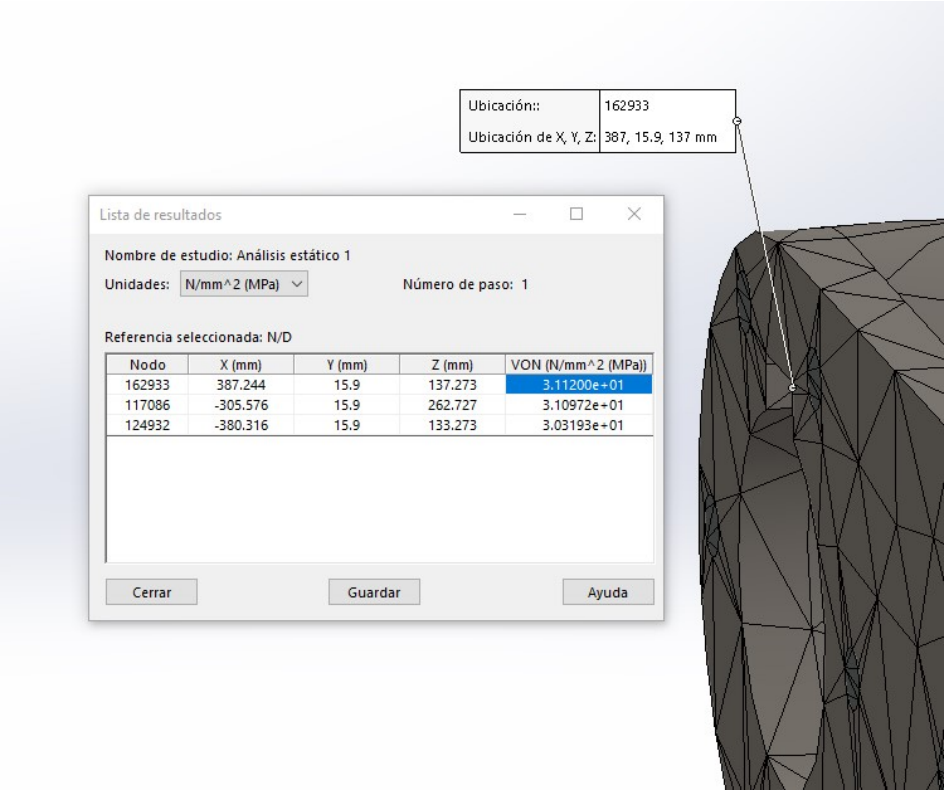

Figure 5.7: Top 3 points with greater strain

Once the weakest sections of the structure are identified it's necessary to know exactly how much strain is been punt in those sections. The points with the most strain are being shown in Fig. 5.7 were the calculated strain is around 31 MPa. It's worth mentioning that the yield limit of both types of steel used in this design are over 200 MPa.

5.2 Automatic Control Performance

To measure the performance of the embedded control it´s necessary to measure the time required to calculate and update the control signal. To do so, six auxiliary digital signals were used to measure the execution time of each part of the program with a logic analyzer. The time chart is shown in Fig. 5.8. It can be seen that the sampling rate is 2.5ms, the overall time needed to calculate the control signal is 0.56ms,to calculate the ESO is 0.249ms, to calculate the control is 0.112ms, to update the control signal is 0.096ms and to acquire the reference is 0.226ms. Regarding the reference signal, it is updated every 30ms for this specific case but the frequency in which the reference signal gets updated is dictated by the PC.

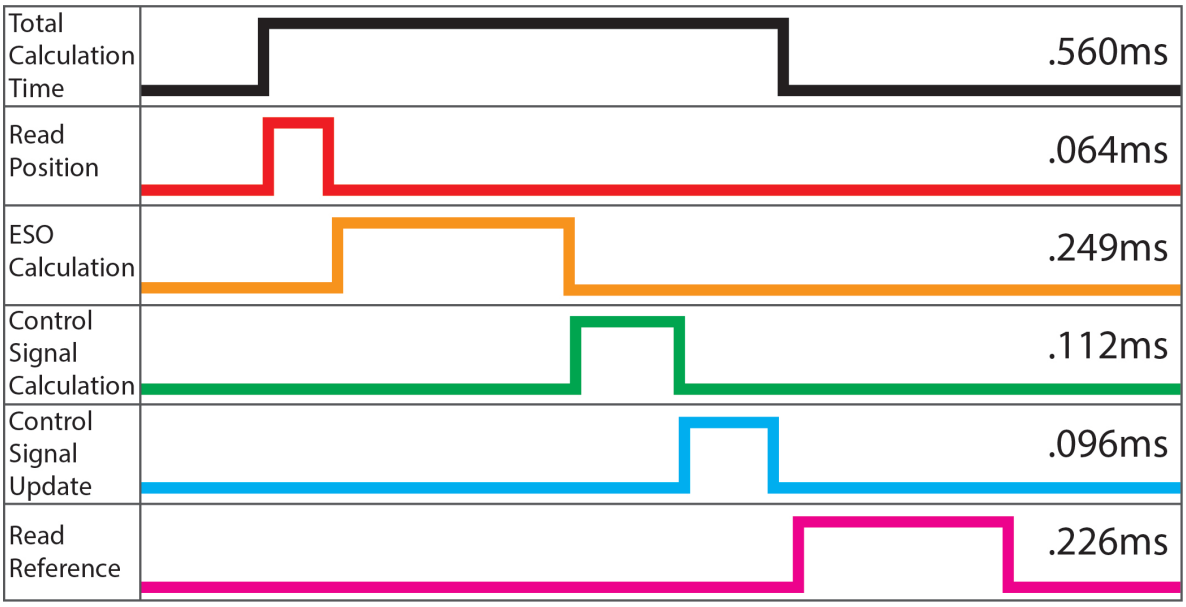

Figure 5.8: Time charts of the calculation time

To measure the control performance, a square wave signal was generated in LabVIEW and used as a reference for the embedded control. The MCU sends the measured position to LabVIEW of a single motor and both reference and measured positions are recorded in an Excel spreadsheet. For a quantitative analysis of the control performance the collected data was processed in MATLAB. In MATLAB Fig. 5.9 was obtained, in the upper plot the

evolution of the motor's angular position is shown and in the lower plot the evolution of the ISE index.

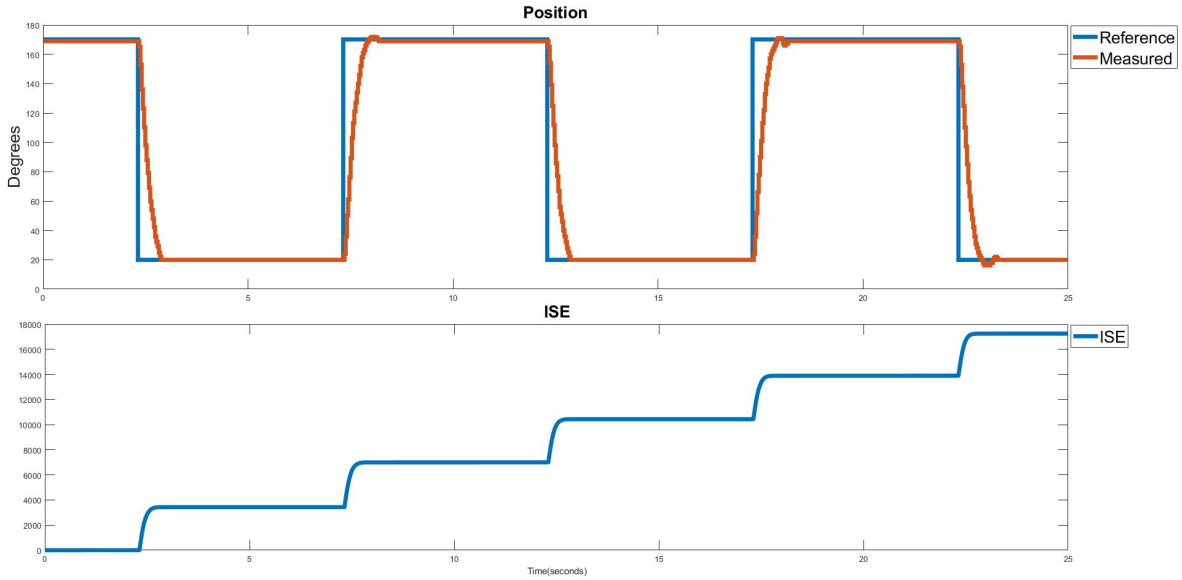

Figure 5.9: Measured Performance of the control

It's important to note that the implemented control strategy is for regulation and not tracking as described in section 3.2.3 in (3.27). The reason for that will be explained in more detail in chapter 6.

5.3 Driving Simulator

The final implementation of the re-configurable driving simulator prototype is comprised of at least two displays, a steering wheel, two pedals, a scale 3-DOF platform and a button panel. Regrading the displays, one is needed to show the 3D simulated environment to the simulator's operator, the other one is needed to display the moderator software GUI, a third display can be used to show the CARLA server and monitor the simulated vehicle from a third person view.

The re-configurable driving simulator prototype's software is running in a "desktop" digital computer with the following specifications:

- CPU: AMD FX-8350 Eight-Core Processor 4.0GHz.
- RAM: 2400 MHz 8GB DDR3 dual chanel.
- GPU: Nvidia GeForce GTX 1050 Ti.

- Storage: SATA SSD.
- SO: Windows 10 x64 version 20H2.

The workloads generated by executing the software were measured using the Window's task manager as shown in Fig. 5.10. The current computer hardware is capable of executing the software without issues but is worth noticing that the workload is GPU heavy, utilizing up to 70% of the 1050Ti. The current computational hardware is relatively old, therefore, modern GPUs and CPUs could perform the tasks better. Nevertheless, The CARLA Simulator documentation recommends at least a 6GB GPU and to follow the Unreal Engine hardware recommendations which recommends at lest a 6 threads CPU and a SSD.

Nombre	Estado	55% CPU	54% Memoria	0% Disco	0% Red	77% GPU	Motor de la GPU
Aplicaciones (4)							
Administrador de tareas		0.2%	24.3 MB	0 MB/s	0 Mbps	0%	
CARLA UE4		23.4%	848.3 MB	0 MB/s	0 Mbps	70.0%	GPU 0 - 3D
LabVIEW 19.0f2 Development System (3)		5.9%	64.1 MB	0 MB/s	0 Mbps	0%	
Python 3.7		20.3%	126.2 MB	0 MB/s	0 Mbps	0%	

Figure 5.10: Window's task manager screenshot

When the system is working with the scale platform connected, the input devices connected and 3 display monitors the system shows as in the screenshot shown in Fig. 5.11. The center screen displays the moderator software, the screen of the right displays the CARLA server that can be controlled with the mouse and keyboard, the left screen displays the CARLA client. The Driving simulator operator will only see the CARLA client screen while another person can monitor the simulation using the moderator software and the CARLA server.

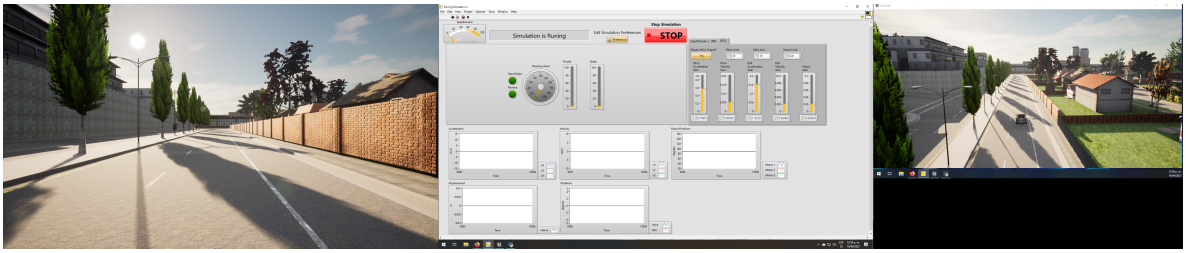

Figure 5.11: Re-configurable driving simulator prototype's desktop screenshot

5.3.1 Moderator Software

On the simulation run-time, the moderator software displays information useful for the prototype stage of the system. While the moderator window can be seen in Fig. 5.11, in Fig. 5.12,

it is possible to see in more detail the user interface. The waveform chats in the bottom of the screen plot the evolution of several variables of interest, like the acceleration and angular velocity measured from the CARLA's IMU, the desired scale platform position, separated in displacements and rotations and labeled with the DOF's name, finally, the motors desired position that is sent to the embedded control.

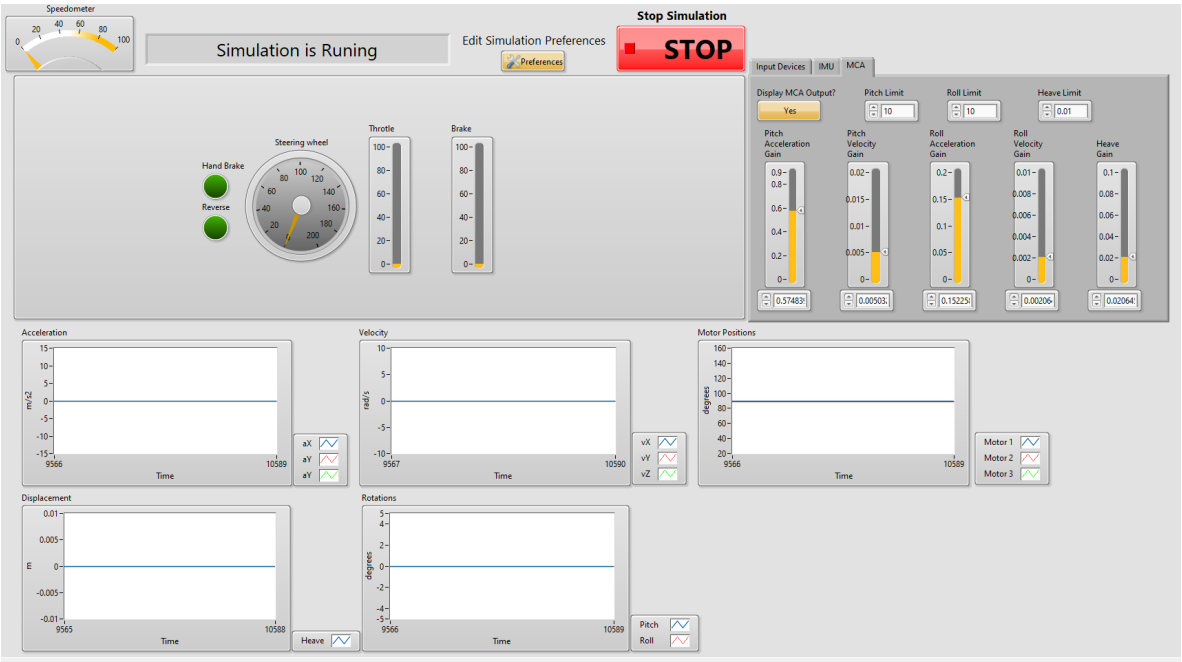

Figure 5.12: Moderator Software Screenshot

In the moderator software its also possible to monitor the driving simulator's input with the gauge for the steering wheel, the vertical fill slides for the pedals and the handbrake and reverse from the LEDs. A speedometer is also shown in the upper left side of the screen, the speed value is calculated from the angular velocity of measured with the IMU sensor. The speed $= 3.6\sqrt{v_x^2 + v_y^2 + v_z^2}$ where v_i are the angular velocity of each axis. This formula was provided by the CARLA team in an example code.

5.3.2 Driving Simulator's Station

The final implementation of the presented re-configurable driving simulator will be described in up next. Fig. 5.13 shows a photograph of the system running without operators. The full station is made up by two parts, the driver station and the supervisor station. The drivers station is the section where the human operator uses the input devices to interact with the 3D simulated environment. The supervisor station is the section where a second human operates

the moderator software and the 3D simulation environment's server screen to supervise all the simulation steps (defined on section 2.6).

Figure 5.13: Re-Configurable Driving Simulator's Full Station

Where the numbers on Fig. 5.13 mean:

1. Moderator software screen.
2. CARLA server screen.
3. CARLA client screen.
4. Scale platform.
5. Steering wheel.
6. Button panel.
7. Pedals.

Fig. 5.14 shows a closer look to the driver station, which is conformed by the CARLA client screen that screen (that displays the 3D simulated environment from the driver's perspective),

the steering wheel, the pedals and the button panel. The platform is not meant to be part of the drivers station in the current stage of development since a scale model doesn't provide any valuable feedback to the driver. Therefore, the scale platform, at this point in time, must be considered part of the supervisor station.

Figure 5.14: Re-Configurable Driving Simulator's Driver Station

Fig. 5.15 shows the pedals of the driver station. the silver pedal is the throttle pedal and the black one is the brake.

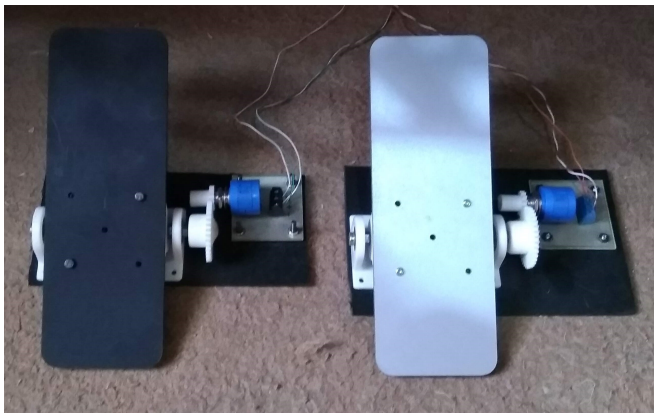

Figure 5.15: Driver Station pedals

Fig. 5.16 shows the button panel. The big red button is the emergency stop, the switch on the left is the reverse and the switch on the right is the hand brake. Since the button panel is the input device that is used the less, the current implementation serves merely as a proof of concept. Therefore, not much time an resources were put on this device, thus the lack of features like labeling or proper packaging.

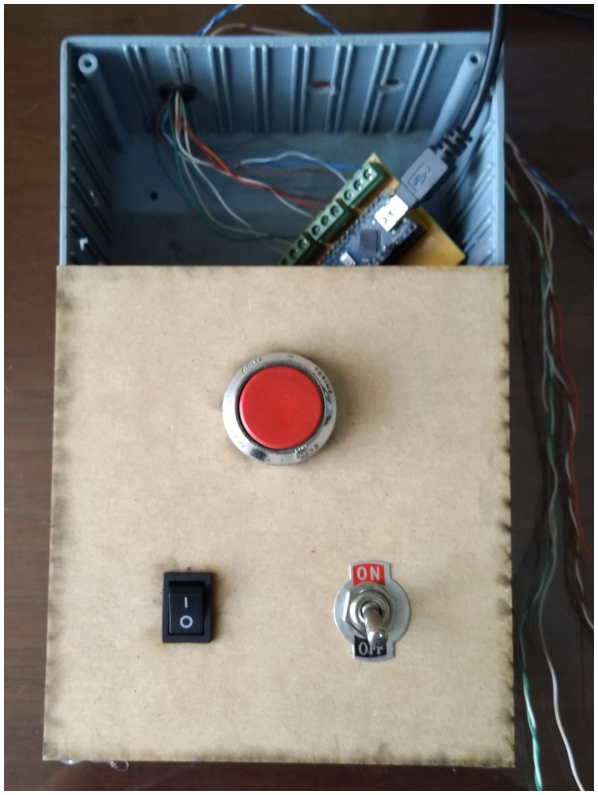

Figure 5.16: Button Panel

Fig. 5.17 shows the scale platform. On the background of this photograph, it can be seen that the embedded control card is powered by a PC Power Supply Unit (PSU) rewired to work as a power supply of fixed 3.3V for the MCU card and 12V for the motors. The PCB is designed to accept a third power lane of 5V in case of needing to power the motor wit more than 12V, on that case the H-bridge module requires to be powered with an external 5V power supply.

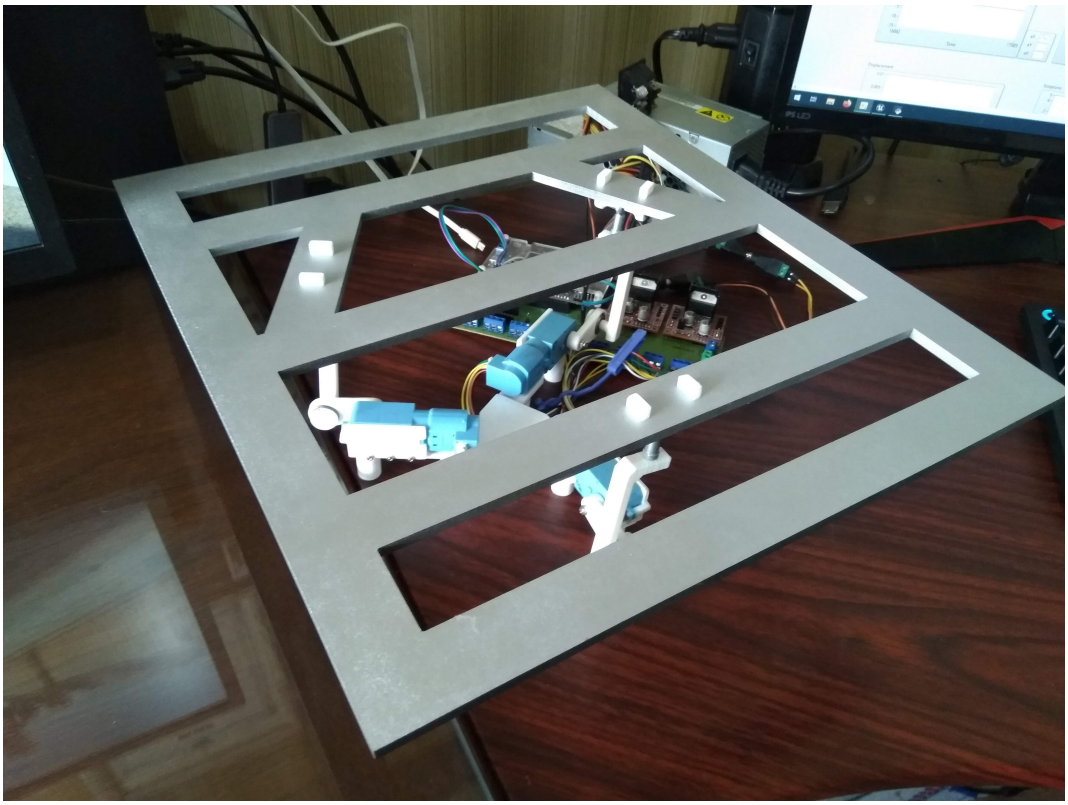

Figure 5.17: Scale Platform

Fig. 5.18 shows the acquisition card described on section 4.6.1 and depicted on Fig. 4.14. The PCB is a two layers design with one plane as a data plane and the other as a common ground.

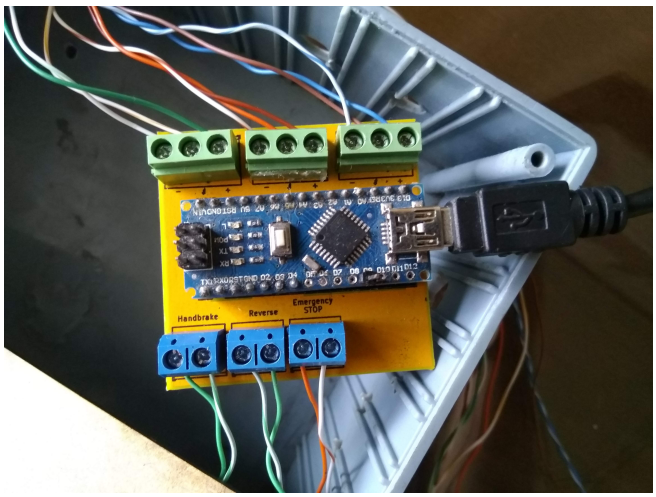

Figure 5.18: Button Panel's acquisition card

Finally, Fig. 5.19 shows the embedded control card with the Ophyra development board and the two h-bridge modules. On this photograph the PCB has extra pin headers and terminal blocks since when it was manufactured, the original design contemplated the inclusion of the input devices acquisition feature in a single PCB. That feature was drooped since, while testing, it was found that having both systems in the same place would increase the wire length of the analog signals unnecessarily and increment the complexity of the set up. The feature was removed from the final design and documentation as it can be seen on Fig. 4.19.

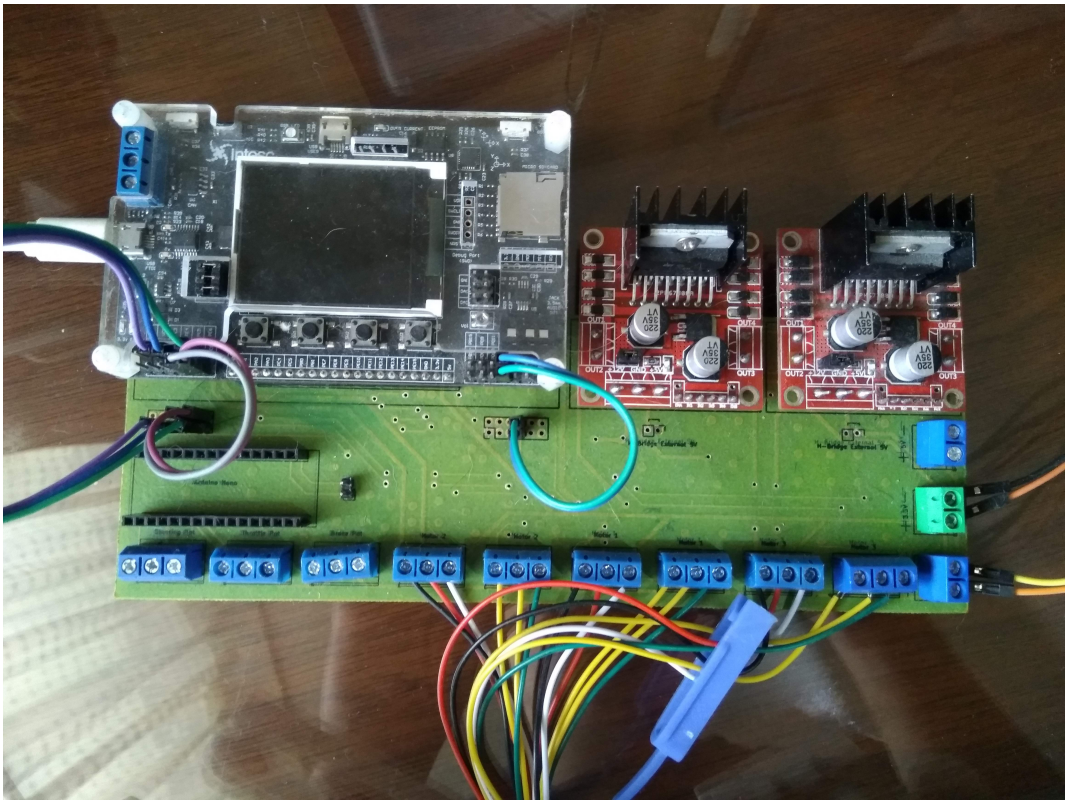

Figure 5.19: Scale platform electronic card

5.4 Testing

To test the implemented system, a simulation based approach was selected. Due to the current 2019-2020 global pandemic, the scope of testing was reduced from a human in the loop approach to a simulation based testing. Te goal of this section is to validate the behaviour of the moderator software, to do so, the MATLAB's simulation is contrasted with the LabVIEW's output.

First, the overall platform's desired position is tested, by doing so, the MCA functional behaviour is validated. Fig. 5.20 shows the evolution of the desired Pitch in both software,

the simulated maneuver is a run-stop, therefore, only the pitch axis is relevant. The run-stop maneuver was selected due to it's simplicity while analyzing the motor behaviour while utilizing all 3 motors. The maneuver was performed using the driver's input devices (steering wheel, pedals, etc.).

In Fig. 5.20 it can be seen that the vehicle starts to accelerate around the second second and stops at second nine. In this graph the vehicle never reaches a constant velocity and the deceleration is sudden and stops the vehicle entirely in less than four seconds. The pitch behaviour simulates a constant acceleration by tilting the platform until the braking start, when that happens the platform tilts on the opposite direction to simulate a negative acceleration, finally, the platforms returns slowly to the home position. It can be seen that both the simulation on MATLAB and the implementation on LabVIEW have the same behaviour with a difference in magnitude and an some kind of phase delay. The reasons for that will be discussed on section 6.3.

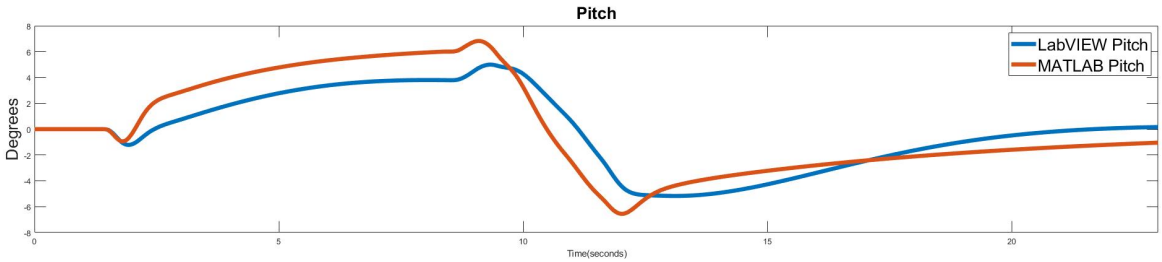

Figure 5.20: Pitch comparison

Fig. 5.21 shows the motor's reference position contrasting the MATLAB's output and the LabVIEW's output. This comparison intends to validate the inverse kinematics algorithm's implementation by comparing it with the simulation results. The graph shows that motor 1 rotates on the opposite direction than motors 2 and 3, this results in a platform tilt on the pitch axis. While the implementation and simulation show a similar behaviour, the difference in magnitude and phase is the same of Fig. 5.20, therefore the source of error seems to be the MCA.

The embedded control performance could not be validated in a similar way but the performance was measured on section 5.2 with the result shown on Fig. 5.9. Nevertheless, while performing the tests, it was observed that the motor's gear's backlash introduced a significant amount of error on the platform. To quantify the amount of error it would be necessary to directly measure the platform's position, this task was impossible within the time constraints and resources available for this project, nevertheless, it's highly recommendable to change the motors for future implementations.

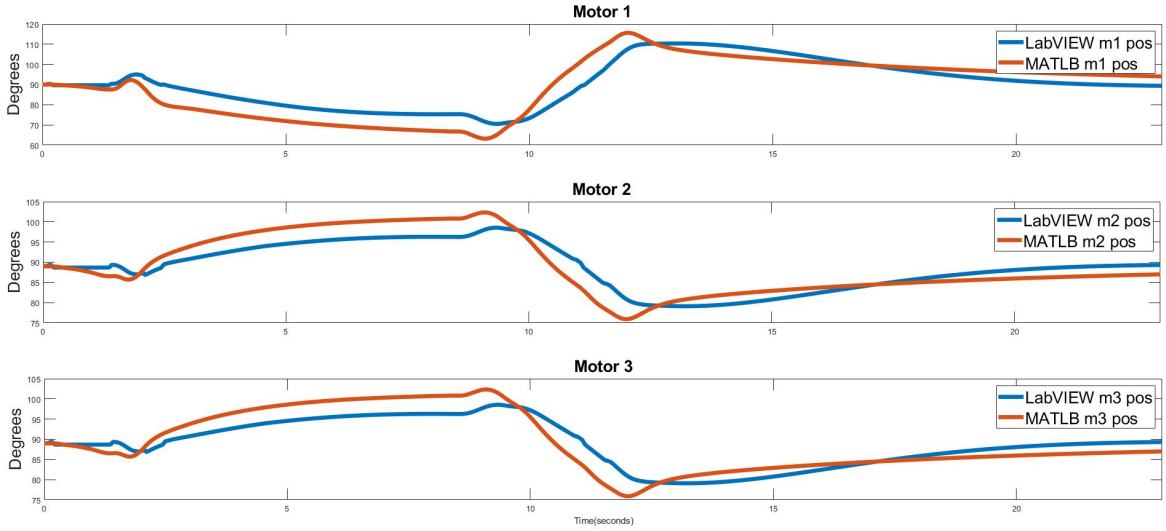

Figure 5.21: Motor's position comparison

The drivers input devices were tested by driving in the simulator freely, since it's impossible to quantify the validity of those devices on the current development stage, as anecdotal evidence, three voluntary test subjects were asked to use the simulator and "dive around". The main complain of the test drivers were the lack of counter-force on the steering wheel and the fact that it doesn't return by it self when performing a turn. A similar complain was provided for the pedals, this issue could be fixed by adding a spring to the pedals. Finally, it was reported that lack rear-view mirrors hindered the realism of the simulation.

Chapter 6

Conclusions

In this chapter presents the appropriate conclusion and results discussion. The first section presents a formal conclusion for the objectives stated at the begging of the research. The following sections will discuss the obtained results. Finally, the last two sections are dedicated to the future of this research, stating future work and viable research topic that can be deviated from this work.

6.1 Conclusions by Objectives

The general objective of this research was "Design and implement a motion platform for a three-degree-of-freedom re-configurable driving simulator prototype". Regarding this statement, and based on the obtained results shown in chapter 5, it is concluded that the main objective was completed successfully.

The motion platform was designed as shown in section 5.1.1 and implemented as shown in section 5.3.2. The driving simulator was implemented as shown in 5.3 in compliance with the re-configurability criteria stated in 2.6. The implementation is still on a prototyping phase as stated in the objective statement.

6.1.1 Design the platform using CAD tools

The design of the platform was divided in two, the scale and full-scale model. Both model's design process is documented in section 4.2, the full-scale design was validated in section 5.1.2 and the scale model was implemented and shown in section 5.3.2. With this results it is concluded that the design of the platform is successful.

6.1.2 To generate and implement the 3 degrees of freedom control algorithm

The platform control algorithm is divided by two components, the inverse kinematics and the motor's angular position control. The inverse kinematics design is presented in section 3.1.1, it's simulation is presented in section 5.1.1 and it's implementation in section 5.3.2. The selected control strategy for the motor's angular position was the ADRC; it's design is described in section 3.2, it's simulation in section 5.1.1 and it was implemented in a embedded control system. The embedded control system's implementation is presented in section 4.5 and it's performance in section 5.2. With this results it's concluded that the 3DOF control algorithm for a 3RRS parallel manipulator was successfully implemented.

6.1.3 To perform the signal conditioning system for the following devices: steering wheel, pedals and lever

The steering wheel and pedals were implemented by using an analog signal acquired with a 10 bits ADC, the analog signal was conditioned by re sizing the data and formatting it to be sent by serial communication. The leaver was implemented with two states switches (Drive/reverse and handbrake) that were acquired with digital pins of a MCU, the conditioning of those signals consisted on formatting the data to be sent by serial communication. The short length of the wires and the noise free environment of the implemented system allowed the singals to retain integrity without any other conditioning. The inaccuracy of the input devices discussed on section 5.4 are not related with the conditioning of the signals, therefore this objective was successfully implemented.

6.1.4 To implement the communication between sensors, input devices and actuators

Upon researching, this objective became more complex that it was originally proposed. To have a functional driving simulator it's not enough to just communicate the senor, input devices and actuators. As described in section 2.1 and depicted in Fig. 2.1, many more

components are needed. Furthermore, if the driving simulator needs to be re-configurable, the communication must also fulfill the criteria established in section 2.6.

The first necessary component is the 3D simulator which was proposed in section 3.4.1 and implemented in section 4.3. The next component is the MCA which was proposed in section 3.3.1, simulated in section 4.1.1 and validated in section 5.1.1. Finally, the communication of all component was described in section 3.4.2 and 3.5. The implementation is documented in section 4.4 and a custom electronic board was designed, but not implemented, to serve all the communication requirements; it's design is documented in section 4.6.3. Therefore, all components communicate with each other correctly and it can be concluded that this objective was successfully implemented.

6.1.5 To perform system testing

The testing consisted on a simulation-based approach and some anecdotal evidence was also provided. The testing is documented on section 5.4. The system's behaviour matches the expected results obtained from the simulation with some magnitude mismatch and delay due to the limitations on implementing the filters required for the MCA. The impact of this mismatches on the simulator's validity require a full-scale platform and a human-on-the-loop approach. The anecdotal evidence suggest the biggest impact on drivers immersion are on the input devices which lack of important features like proprioception feedback on the steering wheel.

Within the constraints of this project, the testing validated all the components that could be tested which are, the MCA, the inverse kinematics and the embedded control (on section 5.2). Therefore, it can be concluded that the testing was successfully performed.

6.2 Forward and Inverse Kinematics

While the forward and inverse kinematics for a 3DOF PM are a well known problem and many solutions can be found in the literature, a solution that could be easily implemented could not be found. Therefore, by modifying the inverse kinematics of a 6DOF SP PM, as shown in section 3.1.1, and developing a simulation based forward kinematics algorithm, the problem was solved.

Nevertheless, the performance of the algorithms is far from perfect, the inverse kinematics gets less precise as the angle λ approaches 0° or 180°. This means that, in the overall performance of the system, the inverse kinematics is contributing to the position error. To quantify exactly how much the inverse kinematics is contributing to the total error can be challenging, since the only way to compare the reference with the output position is with

the forward kinematics, which, might also be susceptible to present an error. A method to identify and quantify the injected error by both algorithms could not be found.

The root of the inverse kinematics problem comes from the fact that is an open loop control, therefore, it requires a very precise modeling, where any false assumption translates in an output error. For that reason, many researches have provided models that take into consideration factors like elasticity of the joints and links or non-linearities in the model. Nevertheless, the models become exponentially more complex which translates in more computation time or a fairly more delicate implementation.

For a driving simulator a precision of a decimal part of degree it's not necessary, since the principle of operation is the acceleration and angular velocity that produces the movement of the platform, for that reason the proposed inverse kinematics is valid. On the other hand, the porpoised algorithm can be further improved, easily implemented and tested. This is one of the major advantages of a re-configurable system.

6.3 MCA

The implemented MCA is the classic one, it was chosen to be used as a comparison point to future implementations of new proposed MCAs, Nevertheless, while the obtained behaviour, both in simulation and implementation, matches with the results found in the literature, some problems in the implementation were found.

The classic MCA is defined in section 3.3.1 and is composed of several filters modeled as transfer functions that depend on the constants $\xi and \omega_n$. The values used were really hard to find since the details of implementation are rarely reported, thus, the implemented values are not agreed by different authors. Furthermore, the reference where the information was found is very old (1988), this leaves a some uncertainty and a headroom for improvement.

Another issue with the MCA is the implementation, while in simulation the filters can be fully defined with the values found in literature, LabVIEW does not provide an out of the box tool to implement a transfer function. To do so the "Control Design and Simulation Module" is needed, and it only runs in LabVIEW 2020 or letter. While this might not be a major issue, the current development was performed in LabVIEW 2018 and upgrading the software is outside the time and constraints of the project. To solve this issue, the point by point filter tool was used and tuned manually to obtain the same behaviour than the transfer function based filters.

6.4 Embedded Control

In the embedded control system exist a discrepancy between the simulation results and the implementation, in the simulation the ADRC control law is for tracking and the implemented control law is for regulation, in other words the simulated control law is (3.20) and the implemented one is (3.21), or to be more precise (3.27). The reason for that is that the tracking control requires the first two derivatives of the position reference, this comes with several problems.

The first problem born from the need of the derivatives is the fact that in certain circumstances the derivative can "explode", this phenomena was found to happen in vehicle collisions and even in simulation created major problems. If the derivative is left as is when it becomes very large causes a failure in simulation, if the derivative is bounded to a limit, the ESO presents estimation errors. A possible solution to this problem is to calculate the position and velocity on the inverse kinematics algorithm instead of just calculating the position. Some information about the velocity calculation was found for the SP PM, unfortunately, for a RRS PM no previous works could be found and a solution couldn't be developed.

Another problem regarding the derivatives is the implementation of the calculation of the derivatives per se. If the calculation is performed on the MCU, some computation time will be added to the control signal computation time. This issue can be solved using a faster MCU or programming the MCU in C instead of MicroPython to optimize the performance. Nevertheless, this approach will face the problem of derivatives growing too much with sudden changes in the reference. The other option is to calculate the derivatives on the PC and send them to the MCU, this might be the best possible solution but it will triple the length of message to be sent. This becomes a issue since the current communication protocol is a virtual serial communication that is somehow slow an unreliable. To fix this issue a USB HID or USB MSC communication must be implemented instead of the USB CDC that is currently implemented. The USB classes are already built in the MicroPython interpreter but the USB port of the STM32F407 is used by default by the MicroPython REPL interface, this can be changed by re-compiling the interpreter.

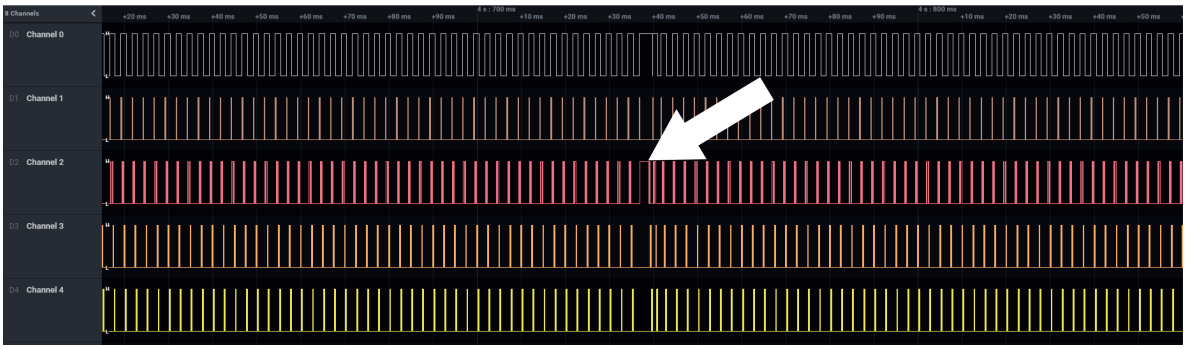

Figure 6.1: Bug in the MCU continuous operation

Regarding the MicroPython interpreter, a inconsistency was detected when the system is running continuously, a glitch shows up were the code delays in random points of the code. A screenshot of the Seleae´s logic analyzer software is shown in Fig. 6.1 where the auxiliary signals are acquired to measure the execution time of the different parts of the code as reported in section 5.2. It can bee seen that the program executes very consistently until a bug appears were the withe arrow points, after which the program returns to operate normally. The error happens periodically every 2s, the source of the delay could not be found, therefor, it's assumed to be generated by the interpreter itself. Nevertheless, even after all this know issues the performance of the embedded control is enough for a prototype, in that note, the benefits of the proposed implementation outshine the problems discussed in this section.

6.5 CARLA Simulator

The CARLA client example code provided by the developers of CARLA Simulator is written with the OOP programming paradigm, nevertheless, to implement features required for the re-configurable driving simulator it´s necessary to break encapsulation of the defined classes as methods. This issue does not hinder the implemented client performance, nevertheless, it's not a good practice to break encapsulation, thus, it's necessary to re define new classes that are best suited to implement a re-configurable driving simulator.

6.6 Moderator Software

The moderator software is the cornerstone of a re-configurable driving simulator. The selected programming environment was LabVIEW since it offers out of the box features that are essential for a system like this, on the other hand, it also comes with some limitations. Since LabVIEW is not an open source software, the available tools are limited to the ones provides by the developers, this hinders the implementation of new features. Another issue with

LabVIEW is the fact that is a licensed software, this means that to use it is necessary to purchase an expensive license, while the cost is not astronomically high, it adds to the total cost of the development.

An alternative is to develop a moderator software with Python, Java, C# or even directly in C++. This would fix the licensing issue and set the growing potential of the moderator software limitless. Unfortunately, this would come with it's own set of problems; the developing time of a new feature will grow exponentially, it will be harder to implement tasks that run in parallel, any modification will require a higher level of understanding of the code, memory leaks will become an issue, bugs will be harder to detect and solve and all those issues will be exacerbated if OOP is not used. Finally, using a DAQ would be impossible, thus, it can be concluded that, while the system remains in a prototype state, LabVIEW wins the trade-off with the other programming environments. In further developing stages it can be recommend to switch to Python or similar to maintain a open source philosophy across all the software pieces.

6.7 Future Work

Since the implemented driving simulator is still in a prototype phase, many tasks are left as future work. In this section the to-do tasks are enumerated, discussed and organized in two sections. The required actions needed to implement a fully functional full-scale driving simulator and the most immediate improvements that can be performed on the system. It's important to mention that the task listed in this section are not the only improvements that can be performed on the system since The re-configurability feature of this design allows the developing of straight forward methodologies for any improvement or new feature to be added.

6.7.1 To Implement the Full-Scale System

In general terms the things that need to be done to build the fully a fully functional diving simulator are: build the platform, add audio feedback, add load feedback to the steering wheel, develop input devices robust enough to work in the field and implement the electronics for the full-scale system. Up next, those points will be discussed in greater detail.

Build the Platform

The mechanical design of this platform is presented in section 4.2.1, nevertheless, before fabrication some considerations adjustments must be performed depending on the available resources.

- Validate the ball bearing: The selected ball bearing was supposed ideal in the finite element analysis, therefore, more testing might be needed.
- Select a motor: The proposed motor was selected in based on availability ans similarity to commercial driving simulators, therefore, an exhaustive review of different motor types must be performed.
- Implement a power stage: Since the scale model worked just fine with a simple H-Bridge the power stage was supposed ideal during the design and validation stage, this can not be the case a full-scale model. The selection or design of a proper power stage for the selected motor must be performed along side with it's simulation and validation.
- Adapt the base and crank: If the motor changes, the design of the platform's base and crank might need a dimensional adjustment or a complete rework.
- Manufacture the platform: The manufacture process comes with it's own set of challenges and depending on the available resources some of the proposed parts might be modified to fit constraints. Nevertheless, a CNC and a plasma cutter should be sufficient tools to manufacture all the pieces.
- Build the platform: This process might not need further detailing but it's important to consider that an appropriate area must be available along with the proper tools needed for assembly.

Audio Feedback

This feature, ideally, must be added to the CARLA simulator directly, in other words, a new feature must be added to the CARLA server as a contribution to the opens source project. To perform this task is necessary to create the feature in C++ in compliance with the contribution guidelines of the project. This is the ideal case, nevertheless, it's possible to create a different solution that would run alongside all the other software pieces as part of the CARLA client, the moderator software or as a stand alone software. Furthermore, whichever option is selected, this task will need the generation of the audio samples, implying the use of specialized recording equipment and audio editing software.

Load Feedback

As depicted in Fig. 2.1, the proprioception sense of the diving simulator's human operator is stimulated by a loading of the control input device, in this case, the steering wheel. This means that a torque must be applied to the steering wheel to simulate the resistance on a real vehicle's steering wheel. The process of implementing this feature is similar to the audio

feedback where, ideally, this feature would be added to the CARLA simulator project as a contribution. Nevertheless, since the resistance of the steering wheel is related to the friction on the wheels and the performance of the hydraulic transmission of the vehicle, the modeling of both phenomena must be taken into consideration. The CARLA team is already working in a wheel friction sensor, this feature might be useful in the implementation of this feature once is released.

Input devices

The implemented input devices are suited for a laboratory environment thus a robust system must be implemented. A suggested proposal is to use a steering sensor for the steering wheel and for a throttle and break use automotive spare parts. This will require some inverse engineering on the devices but the robustness of the system will be unmatched. Whichever system implemented must be able to communicate thru a fieldbus and have a robust performance in noisy environments.

Electronics

In section 3.5.1 the electronics for a full-scale system was proposed and in section 4.6.3 an electronic board was designed for that propose. In appendix A all the components and design elements are detailed. To implement the board all electronic components must be acquired from a supplier, after that, the PCB must be acquired by a manufacturer, the recommended suppliers and manufacturer are Digikey, PCBway or JLCPCB, finally, the components must be soldered to the board. Some modifications on the boards design might be needed to fit the availability of semiconductors on the given time.

The custom board that communicates with the PC thru a USB port can be replaced by a DAQ like the National Instrument's PXI or CompactRIO, this would solve many of the issues described in section 6.4. Regarding the Embedded control, if the programming of the MCU, ideally, the current program should be modified and translated to C++, nevertheless, if the program will be intended to remain in MicroPython, the interpreter should be recompiled to fit the custom board. No matter which programming language is used in the next step of implementations, the software will need to be modified to change the communication protocol used to update the reference position. Finally, security features like a emergency stop are mandatory and security features on the control itself are also recommended.

Driver's Performance Measurement

Once the system is functional, a case of use must be stated and with that comes performance metrics. The human-on-the-loop approach of a driving simulator is intrinsically related with

the performance measurement of a certain set of variables. Therefore a system that measures the performance of the selected variables must be implemented, ideally, this feature should be part of the moderator software or a piece of software that communicates with the moderator software.

6.7.2 To Improve the System

The tasks described here are not mandatory to implement the Full scale system, but, implementing such features would enhance the performance of the system and add to the validity of the driving simulator. Regarding the validity of driving simulators, in [5] many resources about the topic are addressed. The validity of a simulator is closely related to the intended task that will be performed, therefore, once a research goal is set, is necessary to evaluate first the validity of the driving simulator. Normally, when evaluating the validity, a simulator would pass or not and that's it, on the other hand, thanks to the re-configurability of the current prototype, it's possible to build or enhance the features needed to fulfill the validity requirements. Up next a list of desirable features.

- Implement a VR capabilities.
- Implement a different MCA.
- Implement a different control strategy.
- Implement a different platform modeling algorithm.
- Implement driving performance measurement protocols in the moderator software.
- Implement map and weather selection options to the moderator software.
- Implement maps based on local streets or highways.
- Implement the option to add pedestrians and other drivers from the moderator software.
- Implement a feature to control the traffic control devices during the simulation run time.
- Enhance the built-in vehicle model on the CARLA Simulator.
- Optimize the CARLA client code.
- Optimize the embedded control performance.
- Select a motor that doesn't need a gear transmission.

6.8 Feasible Research Topics

Once the full-scale driving simulator has been constructed, a wide range of research topics will be available. Based on the current state of the art of the published topics in scientific magazines and conferences, the following research topics are suggested. Those research topics have the goal of solving local or national issues that lack solutions based on scientific evidence and have an impact in traffic safety. Also, some research topics suggested are focused on contributing to the state of the art of driving simulators, addressing issues that are not yet published.

Driving Culture, it is known that in Puebla City, Mexico, the driving culture is not optimal and the government's efforts to improve the drivers culture have been not as effective as expected. Therefore, a research to find the root and possible short and long term solutions to this issues can be performed using a driving simulator.

Electric vehicles feasibility, the automotive industry is shifting towards electric vehicles and this arouse the question about the feasibility of such vehicles in the current city and highway layout. While the electric vehicles are expected to perform as good as their combustion counterparts, the end user experience in a Latin American context might be different to the North American and European experience were those vehicles are being developed. A driving simulator based research can be used to measure quantitatively the end user experience of electric vehicles.

Traffic control devices effectiveness, in the local context (Mexico), many road signs or traffic control devices are ineffective, lackluster or ignored, thus, a driving simulator based research can help in the design and development of a more a effective signal topology and placement. This kind of research is popular in European counties and it's a service that some research facilities offer to governments, meanwhile in Mexico this kind of testing is rarely performed.

Driving behaviour of local population by groups of age, gender, profession etc. This topic is highly reported in countries like USA and, since the results differ depending on the location's cultural, economical and social background, it's important to perform localized studies that can return valuable results. Historically many sectors of society have been segregated and pointed as less capable or dangerous drivers, only a a quantitative analysis can cast light over this subject.

Validation of a novel MCA, the design of MCAs is a living topic were every year new proposals are published, many of which never get tested in a real driving simulator. The design, implementation and testing of a novel MCA is a high impact research topic in the driving simulator's research field. At the moment, most (if not all) of the available MCA proposals are in a open loop control configuration, therefore, a closed loop control-based MCA

can be a revolutionary proposal. A re-configurable driving simulator is an essential tool to validate a MCA.

Automotive dashboards and HUDs validation, automotive dashboards are always evolving in the industry and many researching institutes are starting to develop HUD interface to enhance the ergonomic performance of the in-vehicle comfort, entertainment and safety control devices and indicators. This is a hot topic in the driving simulator's research field, and entire motion based driving simulators have been constructed to test novel dashboards and HUDs proposals. A re-configurable driving simulator will be a perfectly suited tool to develop such proposals.

References

[1] G. e. I. I. Instituto Nacional de Estadística, "Accidentes de tránsito terrestre," 2018. [Cited on page 1]

[2] F. H. Administration, "Roadway Human Factors and Behavioral Safety in Europe," Tech. Rep. 2, U.S. Department of Transportation, 2015. [Cited on page 1]

[3] J. J. Slob, "State-of-the-Art Driving Simulators, a Literature Survey," Tech. Rep. DCT 2008.107, Eindhoven University of Technology, Eindhoven, 2008. [Cited on page 1]

[4] H. Arioui and L. Nehaoua, *Driving simulation*. London: CPI Group, 2013. [Cited on pages 1, 2, 11, and 13]

[5] D. L. Fisher, M. Rizzo, J. K. Caird, and J. D. Lee, *Handbook of driving simulation for engineering, medicine, and psychology*. CRC Press, 2011. [Cited on pages 1, 2, 6, 9, and 87]

[6] J. S. Freeman, G. Watson, Y. E. Papelis, T. C. Lin, A. Tayyab, R. A. Romano, and J. G. Kuhl, "The iowa driving simulator: An implementation and application overview," in *SAE Technical Papers*, SAE International, feb 1995. [Cited on page 2]

[7] S. Hahn and W. Käding, "The daimler-benz driving simulator-presentation of selected experiments," in *SAE Technical Papers*, SAE International, feb 1988. [Cited on page 2]

[8] W. Käding and F. Hoffmeyer, "The advanced daimler-benz driving simulator," in *SAE Technical Papers*, SAE International, feb 1995. [Cited on page 2]

[9] Bosch Rexroth AG, "Motion Platforms for Driving Simulators." [Cited on page 2]

[10] S. Kharrazi, B. Augusto, and N. Fröjd, "Vehicle dynamics testing in motion based driving simulators," *Vehicle System Dynamics*, vol. 3114, 2019. [Cited on page 2]

[11] T. Chapron, J.-p. Colinot, and P. S. a. Peugeot-citroën, "The new PSA Peugeot-Citroën Advanced Driving Simulator Overall design and motion cue algorithm," tech. rep., Direction de la Recherche et de l'Innovation Automobile, Vélizy Villacoublay Cedex, 2007. [Cited on pages 2 and 9]

[12] E. Baumgartner, A. Ronellenfitsch, H. C. Reuss, and D. Schramm, "Using a dynamic driving simulator for perception-based powertrain development," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 61, pp. 281–290, feb 2019. [Cited on page 2]

[13] Q. Yao, *A Compact Driving Simulator to Support Research and Training Needs - Hardware , Software , and Assessment*. Master science in mechanical engineering, Clemson University, 2015. [Cited on pages 2, 8, and 9]

[14] E. S. Batalla Goinzález, *DESAROLLO DE UN SIMULADOR 3d DE MANEJO PARA LA CAPACITACIÓN DE CONDUCTORES DEL SISTEMA DE TRANSPORTE COLECTIVO METROPOLITANO*. PhD thesis, Instituto Politécnoco Nacional, 2012. [Cited on page 2]

[15] O. A. Perez Reyes, *Desarrollo de un Simulador de Tractocaminoes Utilizando un Ambiente Inmersivo 3D*. PhD thesis, Instituto Politecnico Nacional, 2012. [Cited on pages 2 and 8]

[16] K. Guldbrand and R. B. Gustafson, *Developing a low-cost driving simulator and the physical components effect on validity*. PhD thesis, Aalborg University, 2011. [Cited on pages 2 and 8]

[17] A. Pérez De La Cruz, *ENTORNO VIRTUAL PARA EVALUAR PRÁCTICAS DE MANEJO*. Ingeniero en computación, UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO, 2018. [Cited on page 2]

[18] Organisation Internationale des Constructeurs d'Automobiles (OICA), "2018 Statistics | OICA," 2019. [Cited on page 3]

[19] J. L. Gabbard, M. Smith, K. Tanous, H. Kim, and B. Jonas, "AR DriveSim: An Immersive Driving Simulator for Augmented Reality Head-Up Display Research," *Frontiers in Robotics and AI*, vol. 6, no. October 2019, pp. 1–16, 2019. [Cited on page 7]

[20] Z. Medenica, R. Miucic, and M. Andrews, "Integrating a Real Vehicle into a Physics-Based Driving Simulator for Human-Machine Interaction Research," *International Conference on Human System Interaction, HSI*, vol. 2019-June, pp. 174–178, 2019. [Cited on page 8]

[21] M. Wilkinson, T. Brown, D. Ph, and O. Ahmad, "The National Advanced Driving Simulator (NADS) Description and Capabilities in Vision - Related Research," *Optometry - Journal of the American Optometric Association*, vol. 01, no. 83(6), pp. 79–84, 2012. [Cited on pages 8 and 9]

[22] G. J. Wilde, "Risk homeostasis theory: An overview," *Injury Prevention*, vol. 4, no. 2, pp. 89–91, 1998. [Cited on page 8]

[23] S. Wright, N. j. Ward, and A. G. Cohn, "Enhances Presence in Driving Simulators Using Autonomous Traffic with Virtual Personalities," *Presence Teleoperators and Virtual Environments*, vol. 11, no. 6, pp. 578–591, 2002. [Cited on page 8]

[24] J. T. Reason and J. J. Brand, *Motion sickness*. Academic Press, 1975. [Cited on page 9]

[25] D. Stewart, "A Platform with Six Degrees of Freedom," *Proceedings of the Institution of Mechanical Engineers*, vol. 180, no. 1, pp. 371–386, 1965. [Cited on page 9]

[26] L. Nehaoua, H. Mohellebi, A. Amouri, H. A. Arioui, S. Espié, and A. Kheddar, "Design and control of a small-clearance driving simulator," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 2, pp. 736–746, 2008. [Cited on page 9]

[27] N. A. Kaptein, J. Theeuwes, and R. D. Van Der Horst, "Driving simulator validity: Some considerations," *Transportation Research Record*, no. 1550, pp. 30–36, 1997. [Cited on page 9]

[28] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," *1st Conference on Robot Learning*, no. CoRL, pp. 1–16, 2017. [Cited on page 11]

[29] H. Autrum, R. Jung, W. R. Loewenstien, D. M. MacKay, and H. Teuber, *Handbook of Sensory Physiology - Vestibular System*, vol. 53. Springer-Verlag Berlin, 2019. [Cited on page 11]

[30] G. L. Zacharias, "Motion Cue Models for Pilot-Vehicle Analysis," tech. rep., BOLT BERANEK AND NEWMAN INC, Alexandria, VA, 1977. [Cited on page 12]

[31] Z. Bingul and O. Karah, "Dynamic Modeling and Simulation of Stewart Platform," in *Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization*, InTech, mar 2012. [Cited on pages 14 and 21]

[32] S. Kizir and Z. Bingul, "Position Control and Trajectory Tracking of the Stewart Platform," in *Serial and Parallel Robot Manipulators - Kinematics, Dynamics, Control and Optimization*, InTech, mar 2012. [Cited on page 14]

[33] W. Zhou, W. Chen, H. Liu, and X. Li, "A new forward kinematic algorithm for a general Stewart platform," *Mechanism and Machine Theory*, vol. 87, pp. 177–190, 2015. [Cited on page 14]

[34] D. Jakobović and L. Budin, "Forward kinematics of a Stewart platform mechanism," *Journal of Mechanical Design*, vol. 115, no. 4, pp. 277–282, 2002. [Cited on page 14]

[35] T. Itul and D. Pisla, "On the kinematics and dynamics of 3-DOF parallel robots with triangle platform," *Journal of Vibroengineering*, vol. 11, no. 1, pp. 188–200, 2009. [Cited on page 15]

[36] C. C. Ng, S. K. Ong, and A. Y. Nee, "Design and development of 3-DOF modular micro parallel kinematic manipulator," *International Journal of Advanced Manufacturing Technology*, vol. 31, pp. 188–200, nov 2006. [Cited on page 15]

[37] S.-z. Liu, Y.-q. Yu, Z.-c. Zhu, L.-y. Su, and Q.-b. Liu, "Dynamic modeling and analysis of 3-RRS parallel manipulator with flexible links," *Journal of Central South University of Technology*, vol. 17, pp. 323–331, apr 2010. [Cited on page 15]

[38] Z. Qu and Z. Ye, "Kinematics analysis of certain novel 3-dof parallel manipulator," in *Proceedings - PACCS 2011: 2011 3rd Pacific-Asia Conference on Circuits, Communications and System*, pp. 1–3, IEEE, jul 2011. [Cited on page 15]

[39] D. Brecht, *A 3-DOF Stewart Platform for Trenchless Pipeline Rehabilitation*. PhD thesis, The University of Western Ontario, 2015. [Cited on page 15]

[40] K. Bürüncük and Y. Tokad, "On the Kinematic of a 3-DOF Stewart Platform," *Journal of Robotic Systems*, vol. 16, no. 2, pp. 105–118, 1999. [Cited on page 15]

[41] J. Li, J. Wang, W. Chou, Y. Zhang, T. Wang, and Q. Zhang, "Inverse kinematics and dynamics of the 3-RRS parallel platform," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2506–2511, 2001. [Cited on page 15]

[42] O. Carbajal-Espinosa, F. Izar-Bonilla, M. Díaz-Rodríguez, and E. Bayro-Corrochano, "Inverse kinematics of a 3 DOF parallel manipulator: A conformal geometric algebra approach," in *IEEE-RAS International Conference on Humanoid Robots*, pp. 766–771, IEEE Computer Society, dec 2016. [Cited on page 15]

[43] H. Tetik, R. Kalla, G. Kiper, and S. Bandyopadhyay, *Position Kinematics of a 3-RRS Parallel Manipulator*, vol. 569, pp. 65–72. Springer, 01 2016. [Cited on page 15]

[44] H. Huang, L. Wu, J. Han, G. Feng, and Y. Lin, "A new synthesis method for unit coordinated control system in thermal power plant - ADRC control scheme," *2004 International Conference on Power System Technology, POWERCON 2004*, vol. 1, no. November, pp. 133–138, 2004. [Cited on pages 16 and 26]

[45] B. Hassan, J. Berssenbrugge, I. Al Qaisi, and J. Stocklein, "Reconfigurable driving simulator for testing and training of advanced driver assistance systems," *Proceedings - 2013 IEEE International Symposium on Assembly and Manufacturing, ISAM 2013*, pp. 337–339, 2013. [Cited on page 16]

[46] H. Chalawane, A. Essadki, T. Nasser, and M. Arbaoui, "A new robust control based on active disturbance rejection controller for speed sensorless induction motor," *Proceedings of 2017 International Conference on Electrical and Information Technologies, ICEIT 2017*, vol. 2018-Janua, pp. 1–6, 2018. [Cited on page 26]

[47] J. F. Guerrero-Castellanos, H. Rifaï, V. Arnez-Paniagua, J. Linares-Flores, L. Saynes-Torres, and S. Mohammed, "Robust Active Disturbance Rejection Control via Control Lyapunov Functions: Application to Actuated-Ankle–Foot-Orthosis," *Control Engineering Practice*, vol. 80, pp. 49–60, nov 2018. [Cited on page 26]

[48] J. F. Guerrero-Castellanos, A. Pulido-Flores, J. Linares-Flores, S. E. Maya-Rueda, J. U. Alvarez-Munoz, J. Escareno, and G. Mino-Aguilar, "Active Disturbance Rejection Control for Attitude Stabilization of Multi-rotors UAVs with bounded inputs," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1181–1188, IEEE, jun 2018. [Cited on page 26]

[49] S. Casas, R. Olanda, and N. Dey, "Motion Cueing Algorithms: A Review," *International Journal of Virtual and Augmented Reality*, vol. 1, no. 1, pp. 90–106, 2016. [Cited on page 30]

[50] M. A. Nahon and L. D. Reidj, "Simulator Motion-Drive Algorithms : A Designer ' s Perspective," *J. Guidance*, vol. 13, no. 2, pp. 356–362, 1988. [Cited on page 31]

[51] MicroPython, "MicroPython - Python for microcontrollers." [Cited on page 34]

[52] STMicroelectronics, "STM32F4 - ARM Cortex-M4 High-Performance MCUs - STMicroelectronics." [Cited on page 35]

[53] CARLA Team, "CARLA Simulator Documentation." [Cited on pages 47 and 48]

Appendix A

Custom Board

To design the custom board KiCad 5.1 software was used, all components were selected using the Digikey catalog, for the PCB manufacturing capabilities JLCPCB was selected, to select the MCU pinout the STM32CubeMX was used. Fig. 4.6.3 shows the 3D simulation generated with KiCad. The full schematic and gerber files along with the source KiCad files can be found in the GitHub repository of the project.

A.1 STM32F407VG6

The STM32F407VG6 requires to be powered with 3.3V along side with the specifications stated in chapter 5 of the "STM32F405xx, STM32F407xx Datasheet" and the application note "Getting started with STM32F4xxxx MCU hardware development" that can be summarized in the following:

- 1 4.7uF capacitor
- 7 100nF capacitors ($6V_{DD} + 1V_{BAT}$)
- 2 low ESR 2.2uF capacitors

For the analog reference, an extra filtering stage is necesary for V_{DDA}, V_{REF+} and V_{SSA}. Those pins are accessible thru male pin headers connected to the main power source by default with jumpers. The analog filtering stage is shown in Fig. A.1.

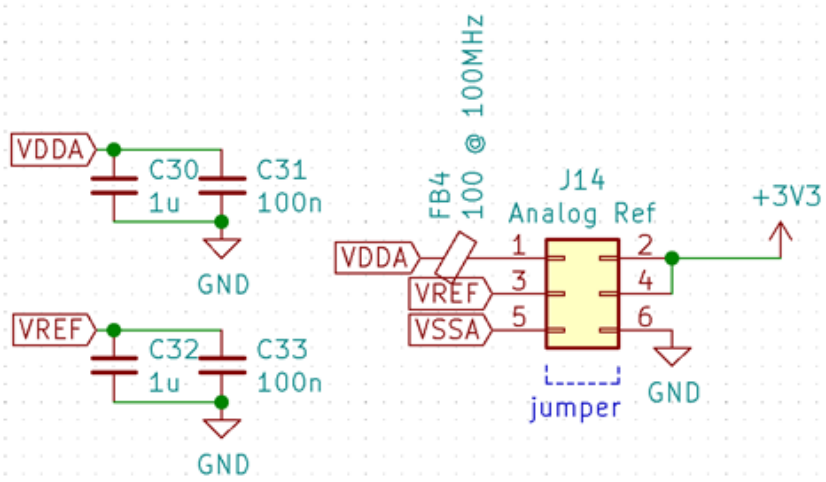

Figure A.1: Analog Filtering schematic

To program the MCU is possible to use the built in bootloader or one of the programming/debug tools offered by the manufacturer like the ST-Link. The ST-based tools use the protocol SWD that is accessible thru male pin headers as shown in Fig. A.2.

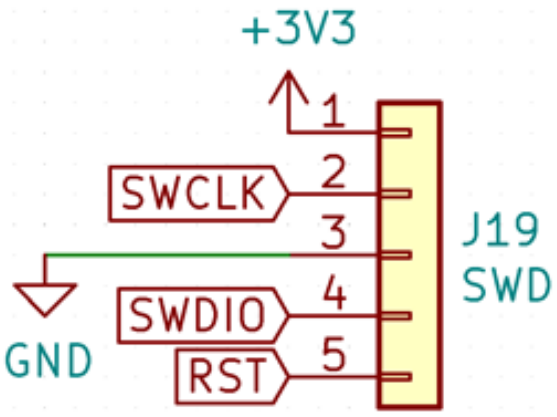

Figure A.2: SWD port

The bootloader is accesible thru the USB-UART or the USB OTG port. that are connected to the pins as shown in the following table.

USB OTG FS	A11/A12
USART3	B10/B11

Nex, the pinout of the MCU is depicted in Fig. A.3. The GPIO pins are connected directly to a terminal block to ease a reliable connection with wires. The ADC pins are also connected to terminal blocks. The communication protocol pins like I2C, SPI and USART are connected to male pin headers. The crest of the pins are connected to other sub systems that will be described ahead.

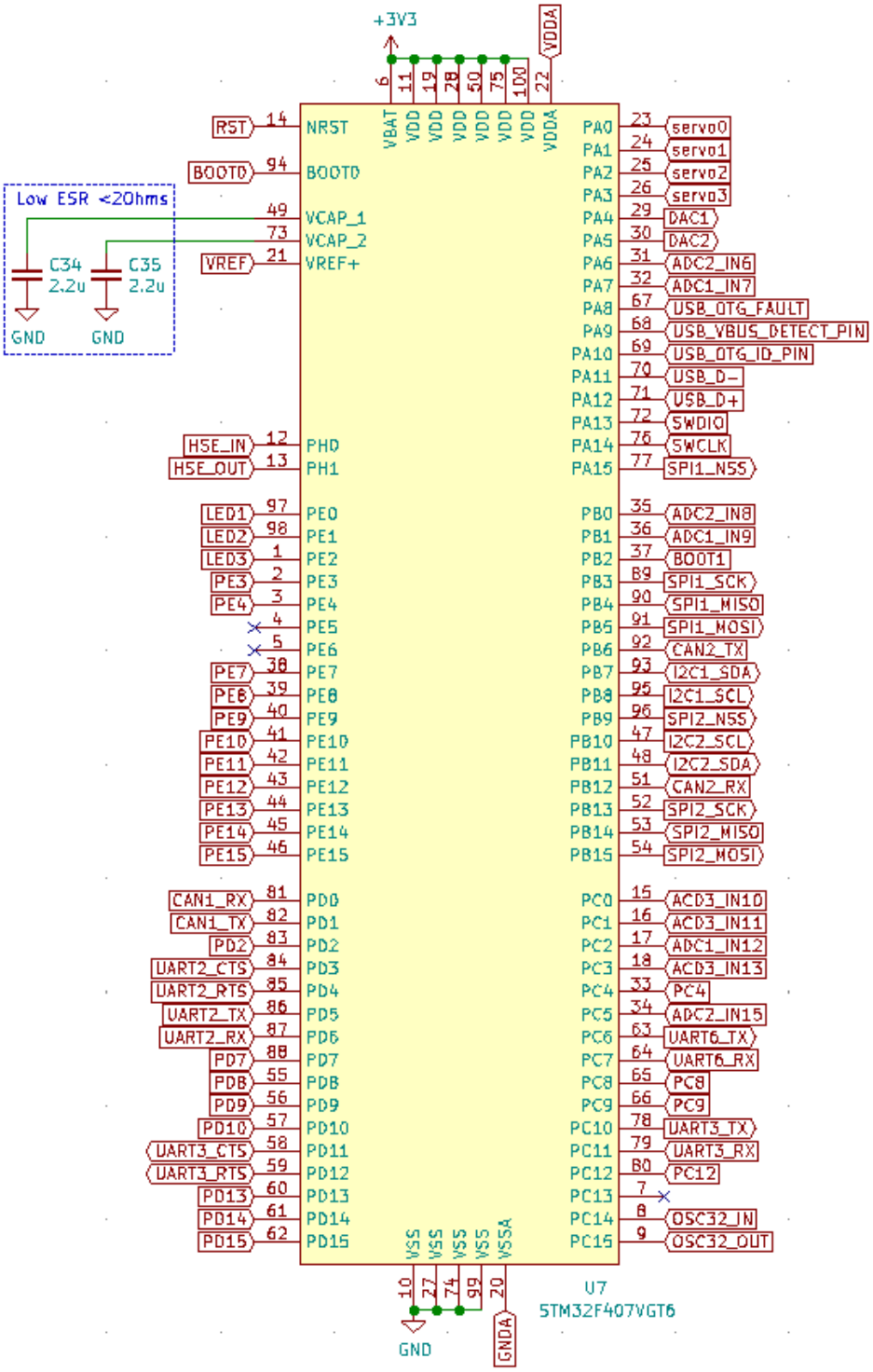

Figure A.3: STM32F407VG6 pinout

A.2 Power Protection

The board can be powered by 3 ways, USB OTG port, USB-RS232 port and a external 6v-24V power source thru a barrel jack connector. The power stage requires to provide 5v for the CAN transceptor and 3.3V for the MCU. The implemented protections are, inverse polarity, over current and a source multiplexer to connect several power sources at the same time.

To implement the reverse polarity protection a DMP3056L P-MOSFET was implemented as depicted in Fig. A.4. This transistor have a threshold voltage of -2.1V, a drain-source resistance of 50mΩ, a max V_{DSS} of -30V and a max current of 4.2A. This configuration makes that the transistor is activated only if the polarity is correct.

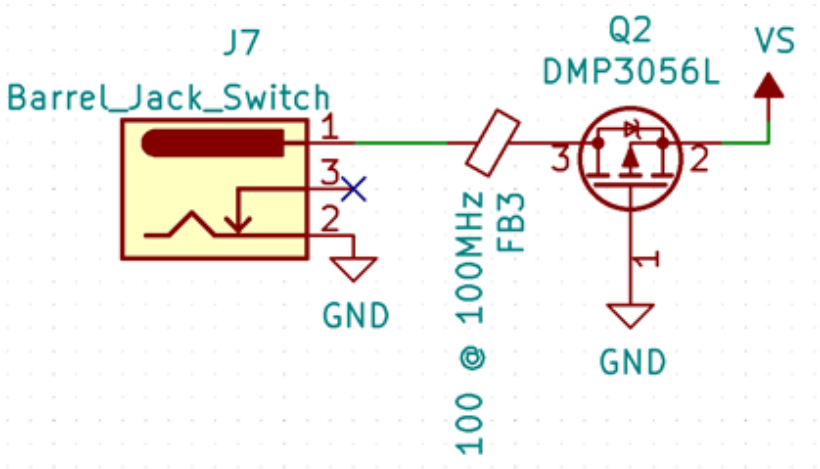

Figure A.4: Reverse polarity protection

To implement the USB power stage, the USB OTG can be set a s host or device, therefore, the board can be powered or power another device from the USB OTG port. To do so the same P-MOSFET transistor is implemented to protect the USB-RS232 and a MIC2025-2YM CI is used to protect the USB OTG as shown in Fig. A.5. The selected CI the OTG is recommended in the “USB hardware and PCB guidelines using STM32 MCUs”” provided by ST, this CI also provides over current protection.

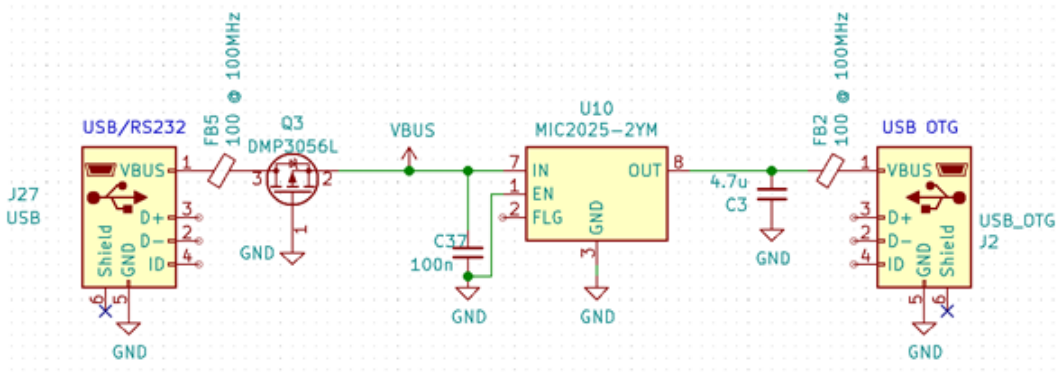

Figure A.5: USB power stage

The recommended configuration by the manufacturer is shown in Fig. A.6. In this configuration the voltage bus is bidirectional and the over current alarm signal provides a low logical level when asserted, therefore is necessary a pullup resistance. The anable pin can be active low or active high depending of the device's part number, this specific device is active low to make it activated by default.

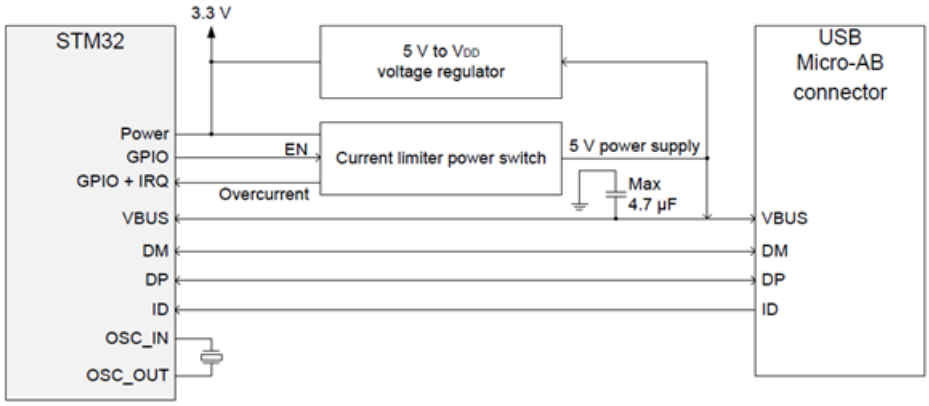

Figure A.6: USB OTG manufacturer recommendation

The internal configuration of the MIC2025-2YM is shown in Fig. A.7. In this implementation the pin C12 is used to control the devices enable pin, the pins A8 and A9 are used to read the over current signal and the presence of voltage on the bus.

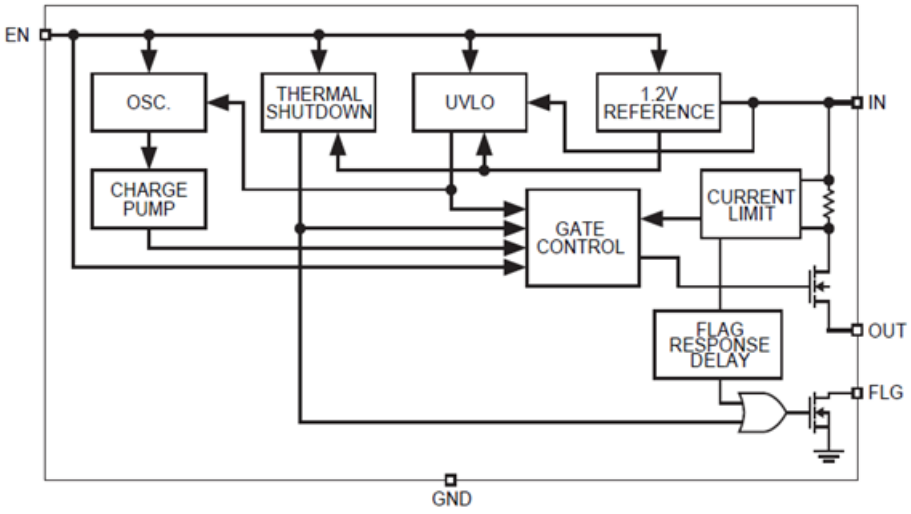

Figure A.7: MIC2025-2YM manufacturer recommendation

The detailed implementation of the MIC2025-2YM is shown in A.8. The voltage precense pin have a voltage divider made up by a 1KΩ and a 470Ω since its possible that the voltage bus have voltage while the MCU is not powered. While the STM32F407VG6 pins are 5V tolerant, they are actually tolerant to $V_{CC} + 4V$ tolerant, therefore, if $V_{DD} = 0$ the pin can be damaged by powering with the 5v of a USB connector.

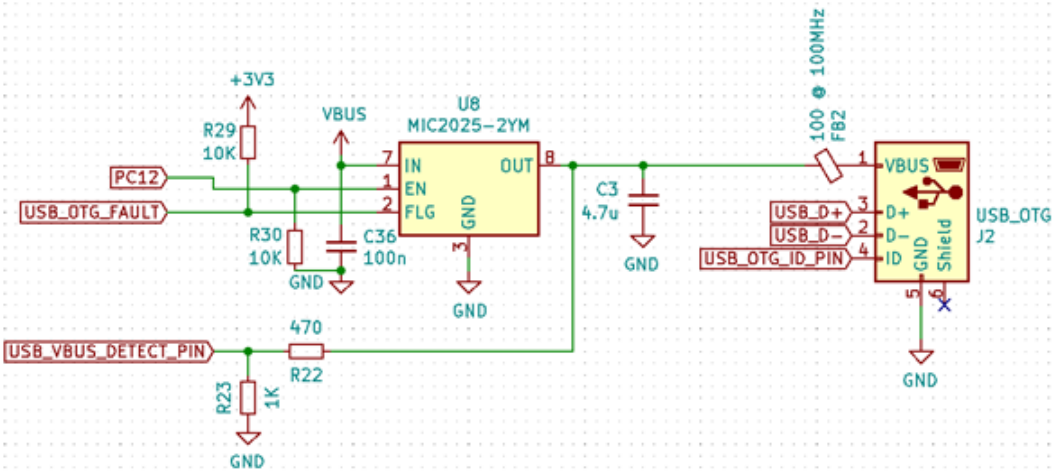

Figure A.8: USB OTG MIC2025-2YM implementation

A.3 Voltage Regulators

To regulate the voltage 3 CI are needed, one to lower the 6V-24V to 5V, another to lower the 5V to 3.3V and another to commute the 5V sources. The selected voltage regulator to lower the external power source to 5V is the MIC4680-5.0YM that is a "step-down(buck)" with a fixed 1.3A 5V output, 6V-34V input.

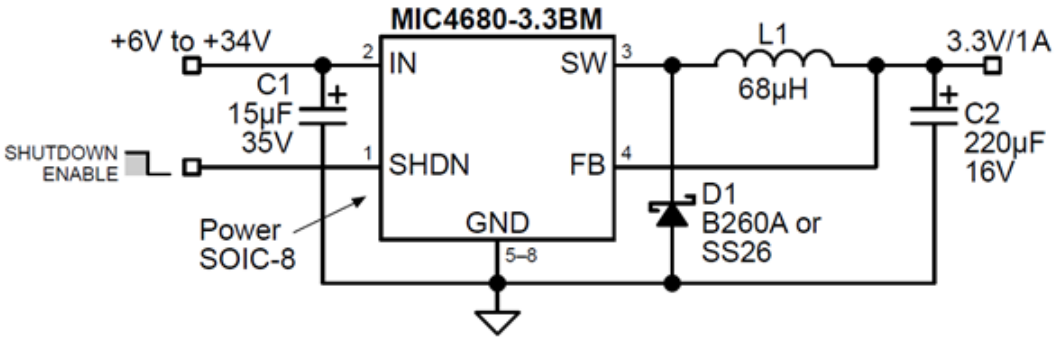

Figure A.9: MIC4680 manufacturer recommendation

The schematic of the implementation for the MIC4680-5.0YM is shown in Fig. A.10. The enabling pin is fixed to ground since the pin is active low and it's not necessary to shut down the device.

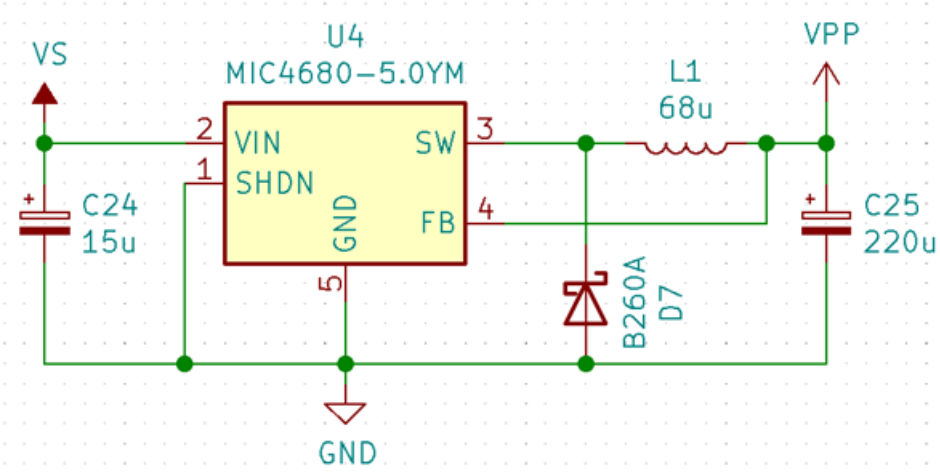

Figure A.10: MIC4680 manufacturer recommendation

In the data sheet of the MIC4680-5.0YM many PCB layout recommendations are provided like the capacitor max voltage, the inductor current, the diode voltage along with recommended part numbers. The implemented layout is depicted in Fig. A.11.

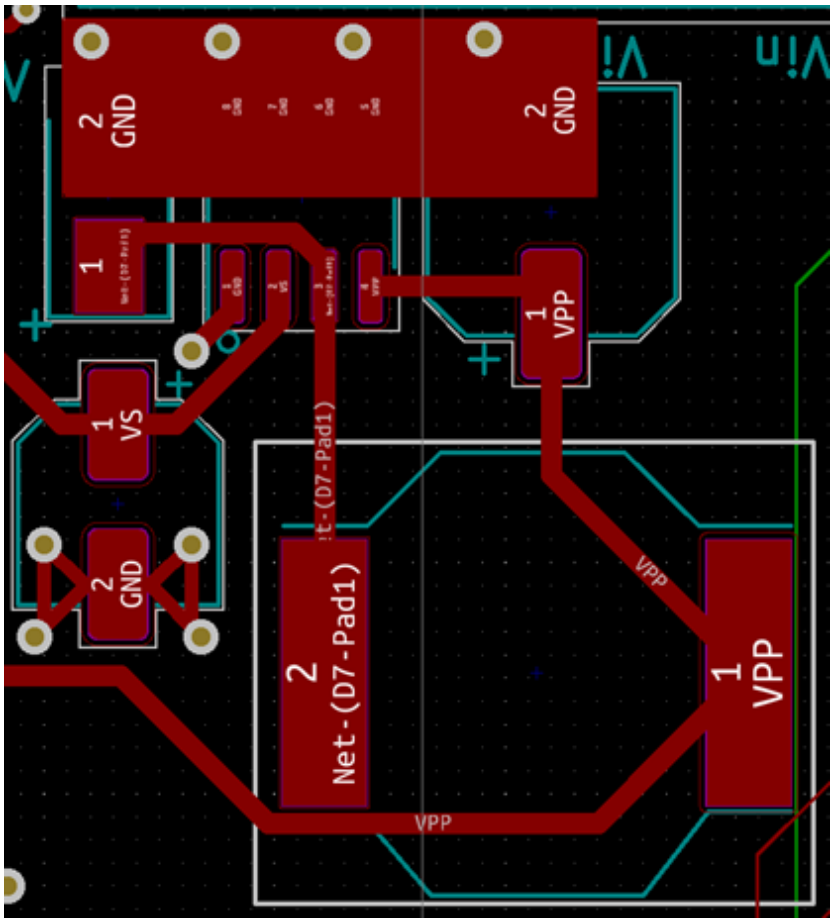

Figure A.11: MIC4680 manufacturer recommendation

Now, in the board, there are 2 5V sources that can be present at the same time, therefore it's necessary to add a "power ORing" of the sources. The selected CI to perform the task automatically was the TPS2112A, this device has a configurable over current protection, a source indicator pin (STAT) and a active low enable pin VSNS that is connected to ground since it's not necessary to disable the device.The TPS2112A's manufacturer recommends the topology shown in Fig. A.12. The behaviour of the device is shown in the following table.

EN	**IN2>IN1**	**STAT**	**OUT**
0	NO	0	IN1
0	YES	Z	IN2
1	X	0	Z

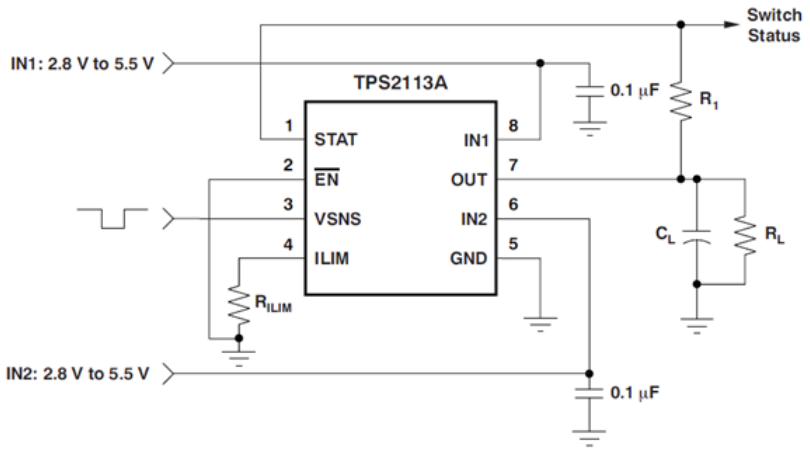

Figure A.12: TPS2112A manufacturer recommendation

The implementation of the TPS2112A is shown in Fig. A.13 were the IN1 is connected to the output of the step down buck converter of the external poser source an IN2 to the dual USB voltage bus. Since the voltage provided by a USB connector is around 4.8V and the buck converter provides a stable 5V output, the case where IN1=IN2 will likely never happen. The current limiter is set to approximately 750mA by the 330Ω resistor.

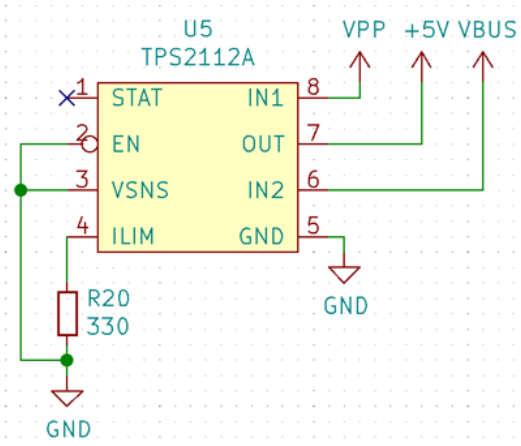

Figure A.13: TPS2112A implementation schematic

The next power stage is a 5V to 3.3V reduction, the selected CI is the MCP1603 with a fixed 500 mA 3.3V output. The manufacturers recommended implementation is shown in Fig. A.14, the manufacturer recommends an inductor with a maximum DCR of 0.1-0.35Ω and a minimum saturation current of 700mA.

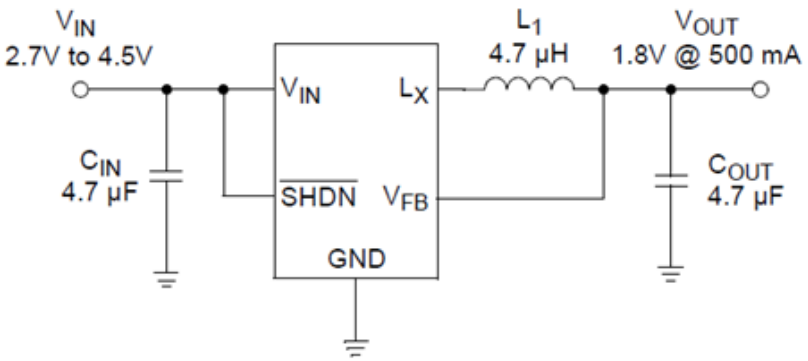

Figure A.14: MCP1603 manufacturer recommendation

That's the end of the power stage of the board, finally the different voltage domains are accessible thru terminal blocks and male pin headers as shown in Fig. A.15

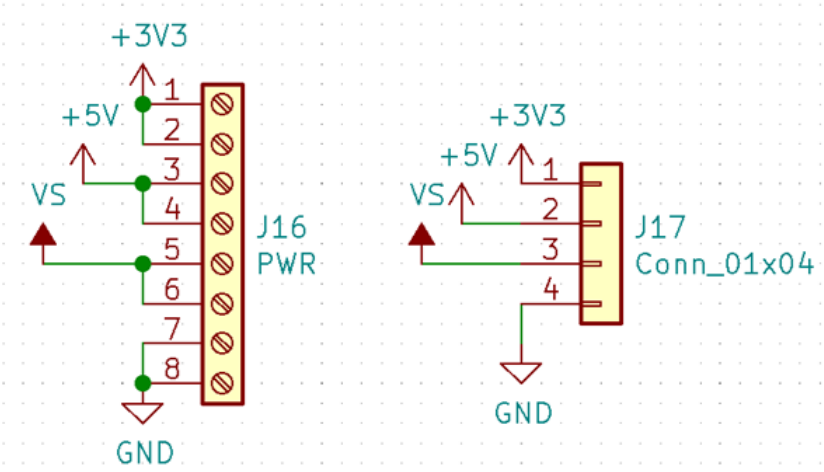

Figure A.15: Power pins

A.4 Oscillators

The STM32f407GVT6 can use two external oscillator a HSE (High Speed External) and LSE (Low Speed External), the HSE oscillator is used to drive the CPU and most of the peripherals, meanwhile the LSE is used for the real time clock (RTC). Fig. shows the MCU's oscillator schematic. The application note "Oscillator design guide for STM8AF/AL/S, STM32 MCUs and MPUs" contains the manufacturer's recommendation. A 8MHz HSE oscillator and a 32.7685KHz LSE oscillator were selected.

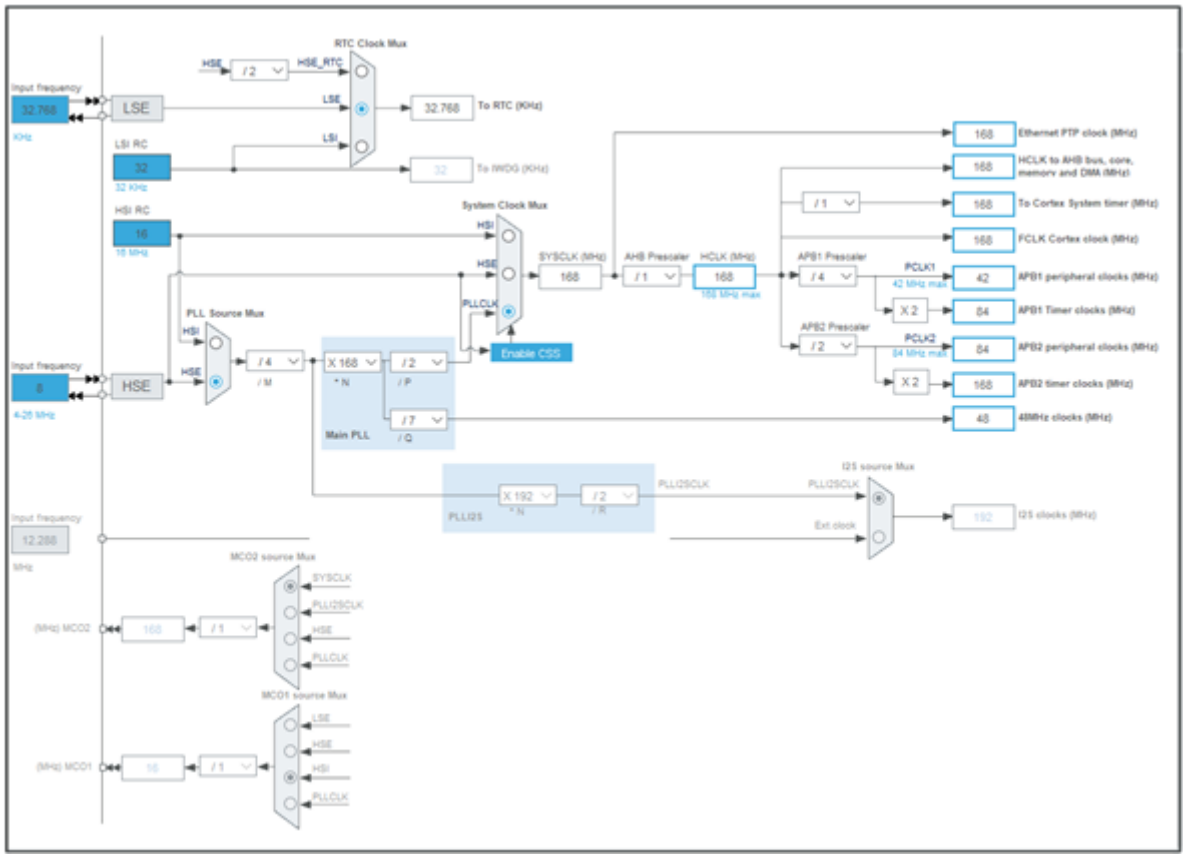

Figure A.16: STM32f407GVT6 Oscillator's Shematic

Fig. A.17 shows the HSE's implementation schmatic that includes two load capacitors C_{L1} C_{L2} that form a total load capacitante C_L, the oscillator circuit also requires a current limiter resistance.

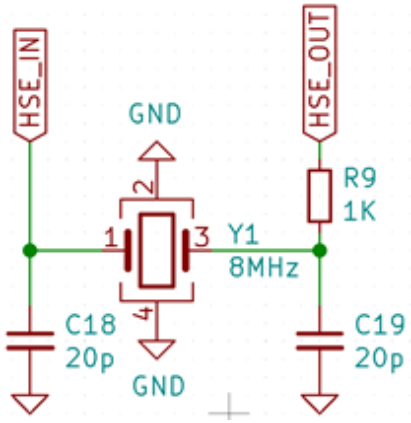

Figure A.17: HSE Shematic

To calculate the load capacitance (A.1) is used

$$C_L = \frac{C_{L1}C_{L2}}{C_{L1} + C_{L2}}C_s \tag{A.1}$$

where

1. C_L = Oscillator crystal's manufacturer recommend load capacitance
2. C_{L12} = Load capacitors
3. C_s = parasitic capacitance

If $C_{L1} = C_{L2} = C$ then (A.1) becomes:

$$C_L = \frac{C}{2} + C_s \tag{A.2}$$

The external resistor is calculated using (A.3) were f is the oscillator's frequency

$$R_{EXT} = \frac{1}{2\pi f C_{L2}} \tag{A.3}$$

It's also necessary to calculate the minimum oscillator's trans-conductance, for the STM32F407VGT6 the minimum trans-conductance is $g_{m(min)} = 5\frac{mA}{V}$ and the full oscillator system's trans-conductance is calculated with the next equation.

$$g_m >> g_{mcrit} = 4(ESR + R_{EXT})(2\pi f)^2(C_o + C_L)^2 \tag{A.4}$$

Where:

- g_m = oscillator's trans-conductance
- g_{mcrit} = mimimum trans-conductance needed to maintain stable the oscillator
- ESR = crystal's Equivalent series resistance
- C_o = crystal's shunt capacitance

The selected crystal is the ABM3B-8.000MHZ-B2-T with a 20μF load capacitance, 4 pins footprint, 8MHz nominal frequency. The crystal's foot print is shown in Fig. A.18, where it can be seen that it has a grounded armour.

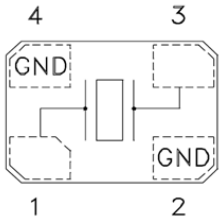

Figure A.18: ABM3B-8.000MHZ-B2-T

To calculate the passive elements shown in Fig. A.17, (A.5) and (A.6) are used to set the values.

$$C = 2(C_L - C_s) = 2(2\mu F - 10\mu F) = 20\mu F \tag{A.5}$$

$$R_{EXT} = \frac{1}{2\pi(8 \cdot 10^6)(20^{-12})} = 994.7\Omega \approx 1K\Omega \tag{A.6}$$

The HSE oscillator require a special layout, it's recommended to use an independent ground plane as show in Fig. A.19.

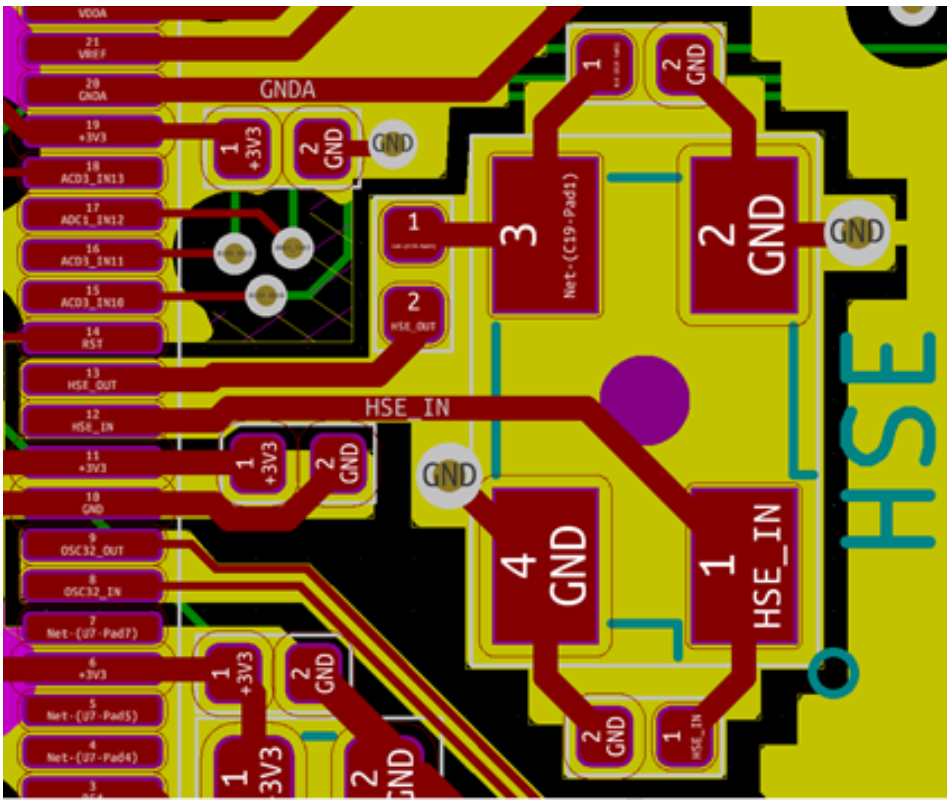

Figure A.19: HSE layout

To select the LSE it's possible to select a part number from the recommended list from ST, nevertheless, the stability of the oscillator is in function of the load capacitance as shown in Fig. A.20.

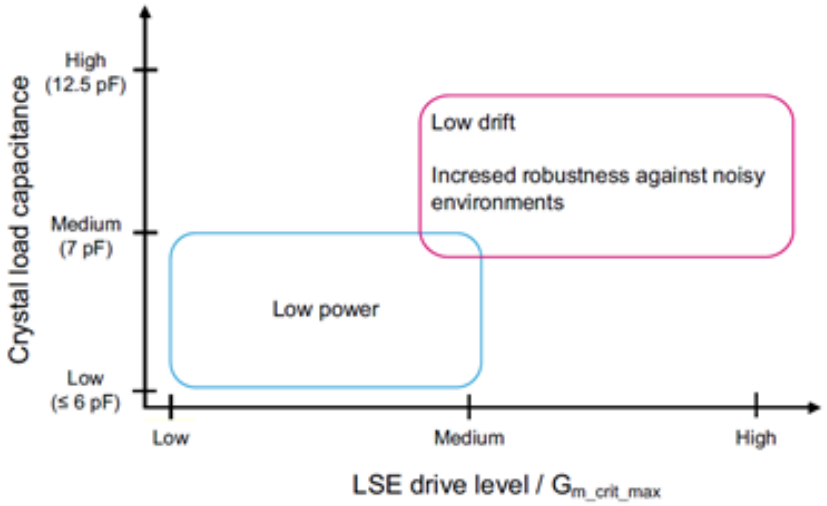

Figure A.20: LSE Gm stability

Due to availability the selected crystal is the ECS-.327-6-12R-TR with a nominal frequency of 32.768KHz, a load capacitance of 6pF, a ESR of 70KΩ and a two pin SMD package. Fig. A.21 shows the implementation's schematic.

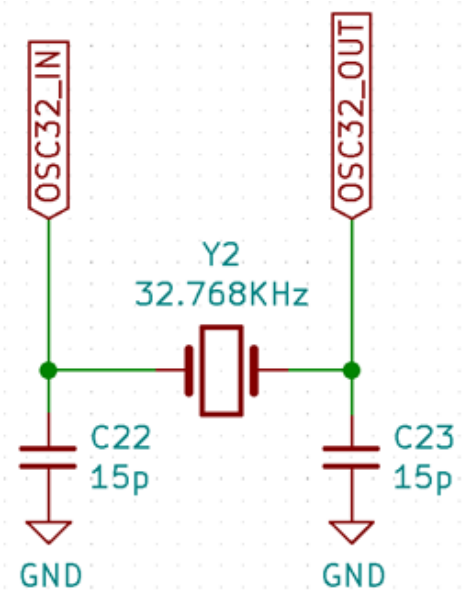

Figure A.21: ECS-.327-6-12R-TR implementation's shcematic

Similarly to the HSE a independent ground plane layout was implemented, this feature is not mandatory for the LSE oscillator but it was recommended by the application note.

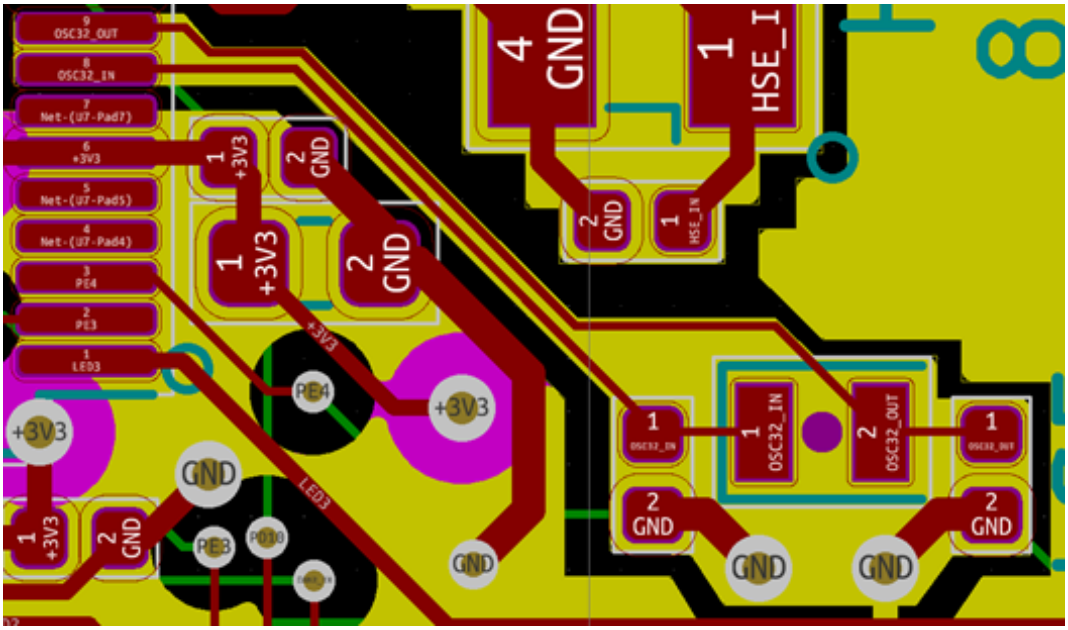

Figure A.22: LSE layout

A.5 USB-RS232 converter

To be able to use the boatloader of the USART6 by a USB port the FT230XS-R CI was selected. This part number from the FTDI manufacturer that is capable to convert the USB protocol to RS232, RS422 and RS485. Fig. A.23 shows the CI's pinout.

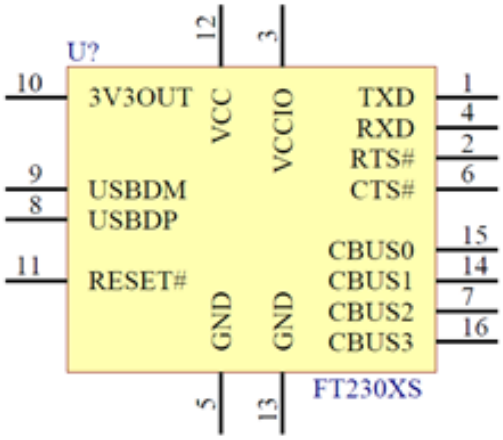

Figure A.23: FT230XS-R pinout

The pin function of the FT230XS-R is shown in the next table. The flow control pins are optional but the are implemented.

Pin number	**Name**	**Funtion**
1	TXD	RS232 data transmit
4	RXD	RS232 data receiver
2	RTS	RS232 request to send
6	CTS	RS232 clear to send
8	USBDM	USB+ differential pair
9	USBDP	USB- differential pair
15	CBUS0	RS485 data transmission enable
14	CBUS1	LED data receive
7	CBUS2	LED data send
16	CBUS3	Active low sleep
11	RESET	Active low reset

Fig. A.24 shows FTDI's recommended implementation. The USB is version 1.1, the CI need to be powered with 5V with decoupling capacitors.

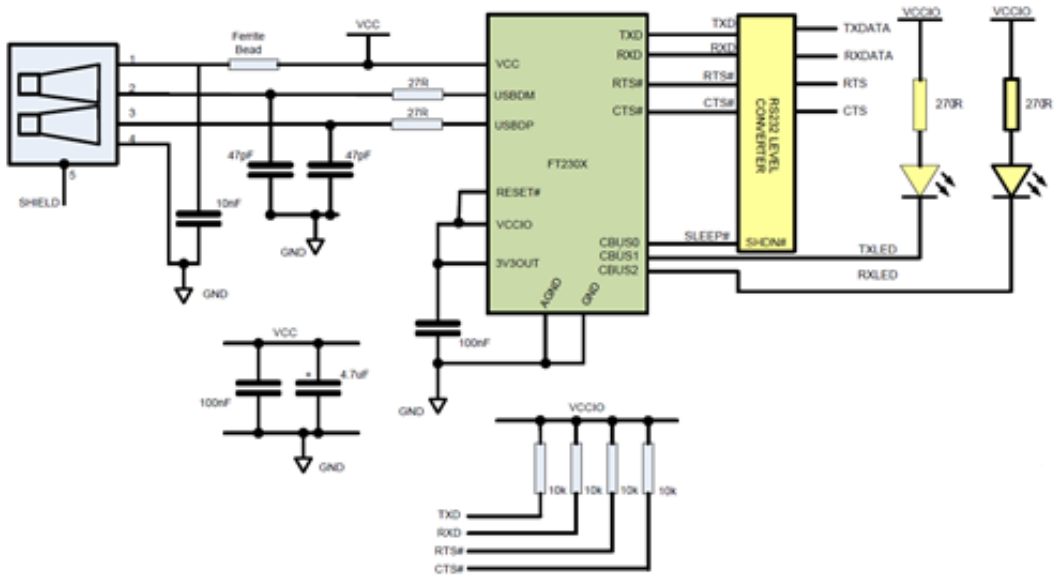

Figure A.24: FTDI's recommended implementation

Fig. A.25 shows the implementation that is fairly similar to the manufacturer's recommendation with the only difference on the power.

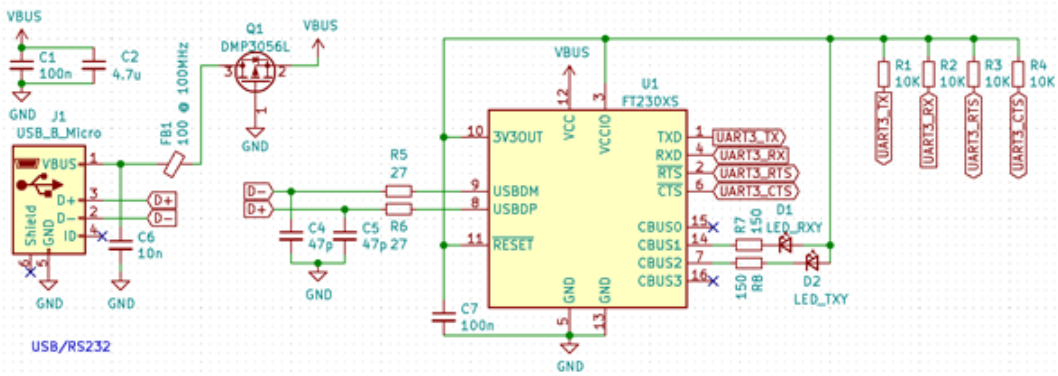

Figure A.25: FT230XS-R implementation's schematic

Fig. A.26 shows the layout that has a differential pair controlled 90Ω impedance from the USB connector to the 27Ω resistors.

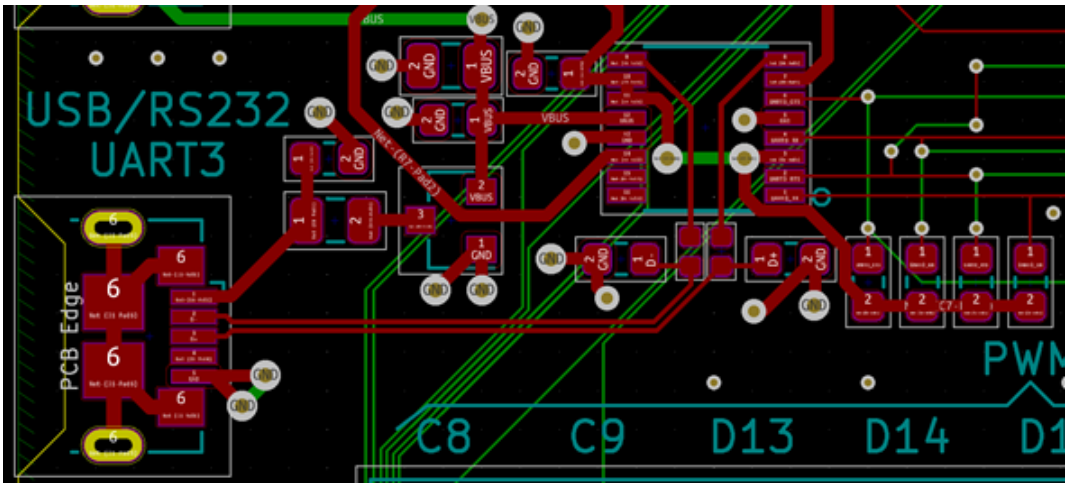

Figure A.26: FT230XS-R layout

A.6 USB OTG FS

The STM32f407VGT6 MCU has an embedded USB 2.0 physical layer for host/device/OTG. STMicroelectronic provides the aplication note “USB hardware and PCB guidelines using STM32 MCUs Introduction” where many recommendations are stated. Some of those recommendation were discussed on section A.2 but alongside those protections a ESD protection was implemented. The ESD protection was recommended on the online training “STM32 USB training” provided by STMicroelectronic. Fig. A.27 shows the recommended ESD protection.

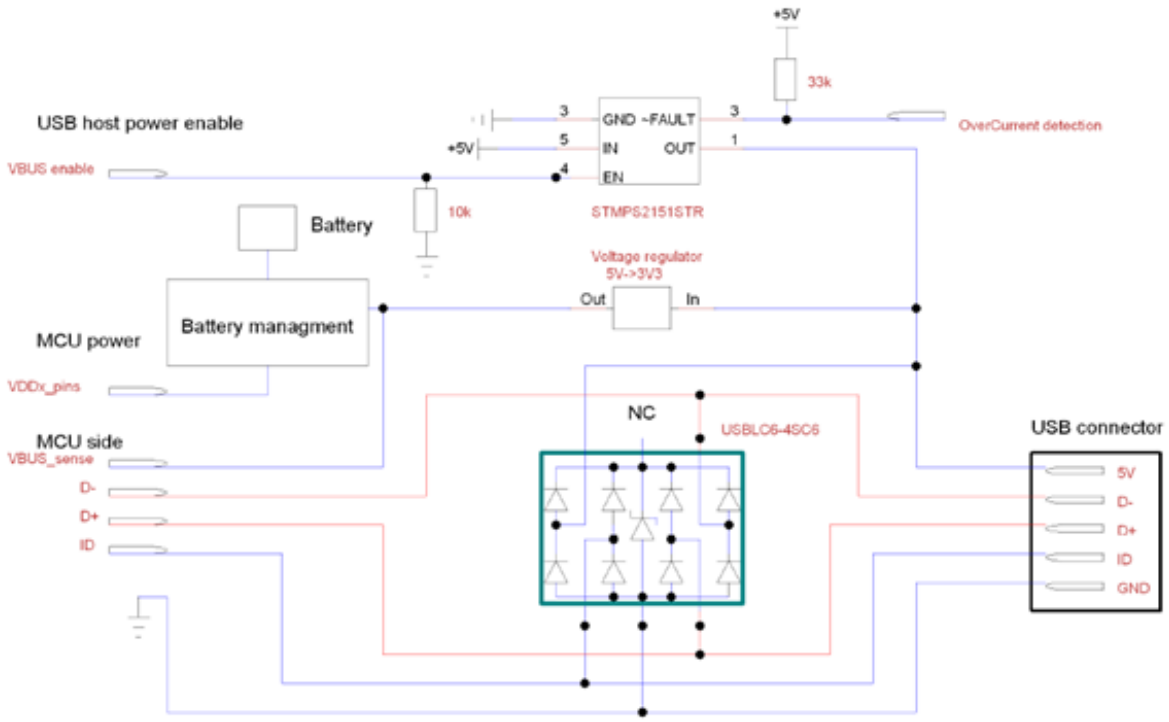

Figure A.27: Recommended ESD protection's schematic

To implement the ESD protection circuit the recommended CI USBLC-4SC6 was implemented. On the datasheet some recommendations about the layout are provided, Fig. A.28 shows the recommended layout. Fig. A.29 shows the ESD protection's implementation and Fig. A.30 shows the layout.

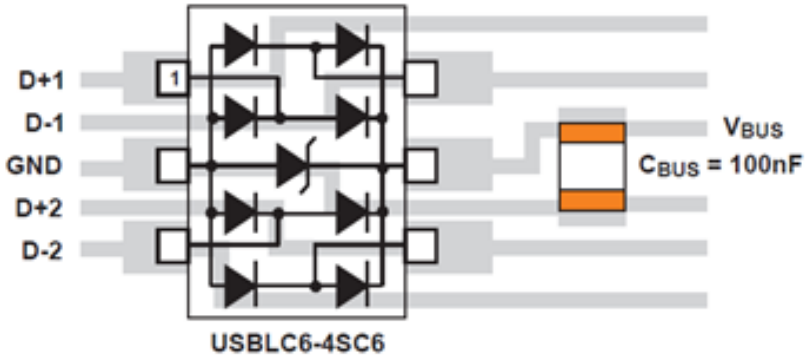

Figure A.28: USBLC-4SC6 recommended layout

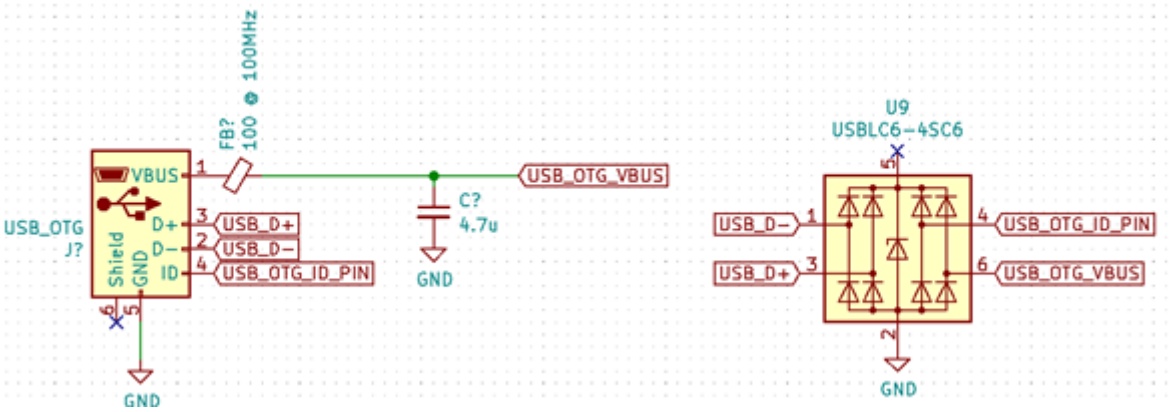

Figure A.29: ESD protection implementation's schematic

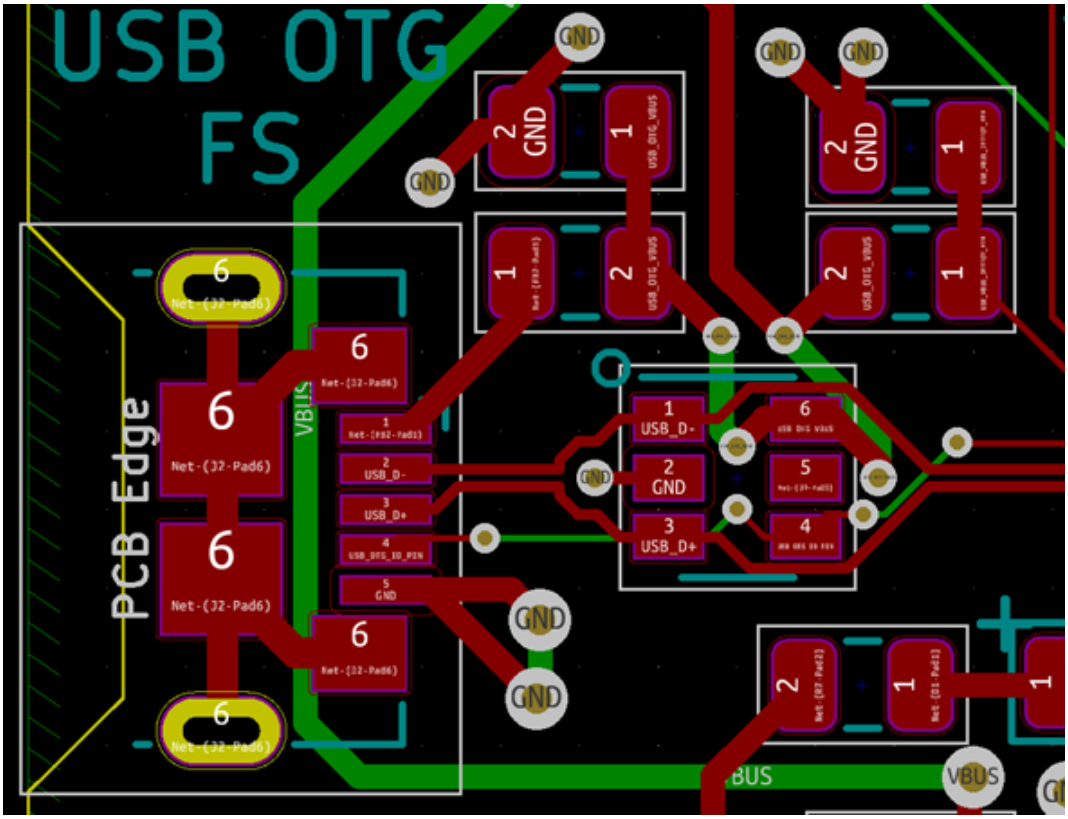

Figure A.30: ESD protection implementation's schematic

A.7 CAN BUS

The STM32F407BGT MCU has an integrated CAN BUS controller, but unlike the integrated USB controller, the CAN controller does not have the physical layer, therefore, it needs a

transceiver to handle the protocol voltage levels, ergo, a transceiver IC must be selected to perform this task.

The selected transceiver is the MCP2562-E/SN, this transceiver was chosen for its low price, the maximum speed supported (1 Mb/s) and the ability to configure the voltage level at which it communicates with the controller.

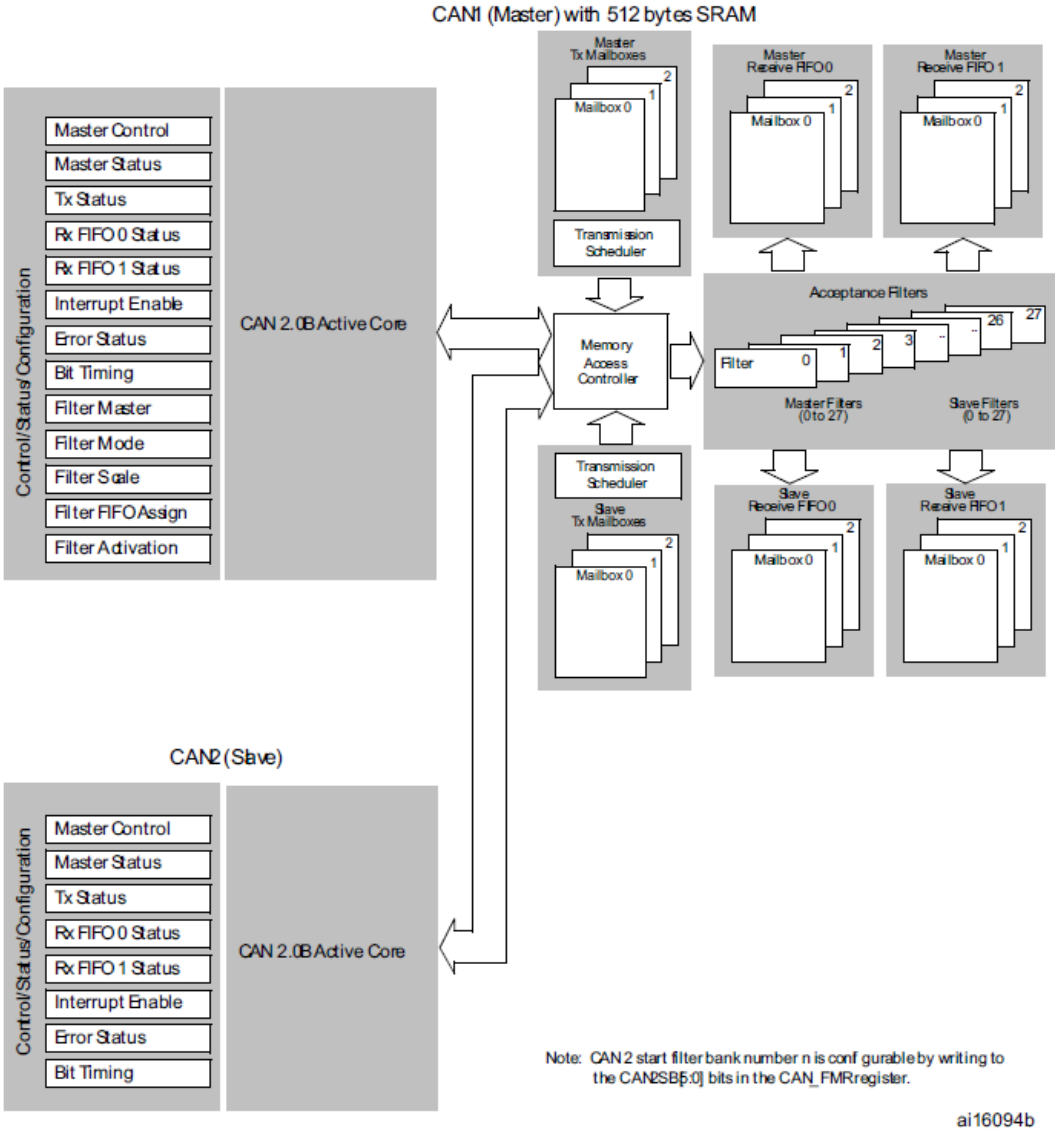

Figure A.31: CAN functional block diagram

The STM32f407VGT6 has two CAN ports available where CAN1 is the master and CAN2 is the slave, i.e. they are not completely independent, CAN2 does not have direct access to memory and requires the CAN1 controller to perform this addressing. Fig. A.31 shows the

CAN BUS controller's functional block diagram.

Fig. A.32 shows the recommended implementation provided by the MCP2562's manufacturer. It's recommended to add the 120Ω terminator but this resistor will be connected thru a jumper to make this terminator resistance optional.

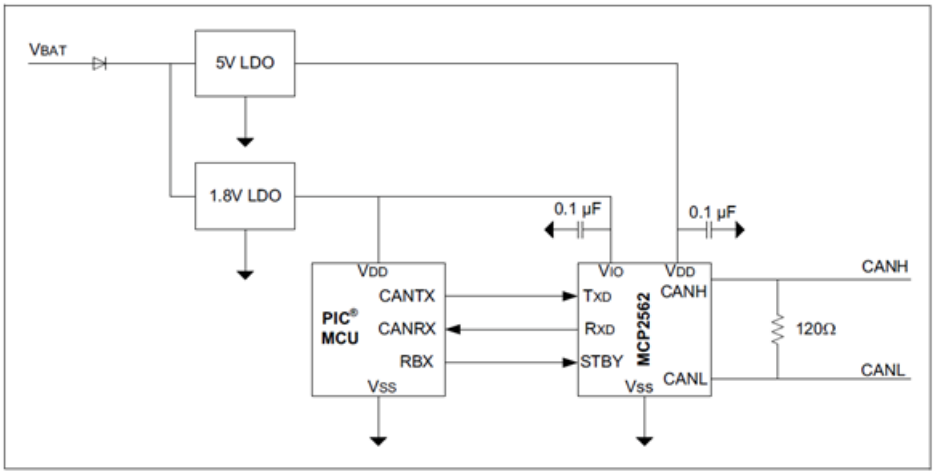

Figure A.32: MCP2562-E/SN recommended implementation

Fig. A.33 shows the implementation's schematic where V_{io} pin is powered with 3.3V to mathc the voltage level of the MCU and the active low enable pin (STBY) is connected to a GPIO with a 10KΩ pulldown resistor. Finally, Fig. A.34 shows the layout. that is the same for both MCP2562-E/SN. Finally, Fig. A.34 shows the layout. that is the same for both MCP2562.

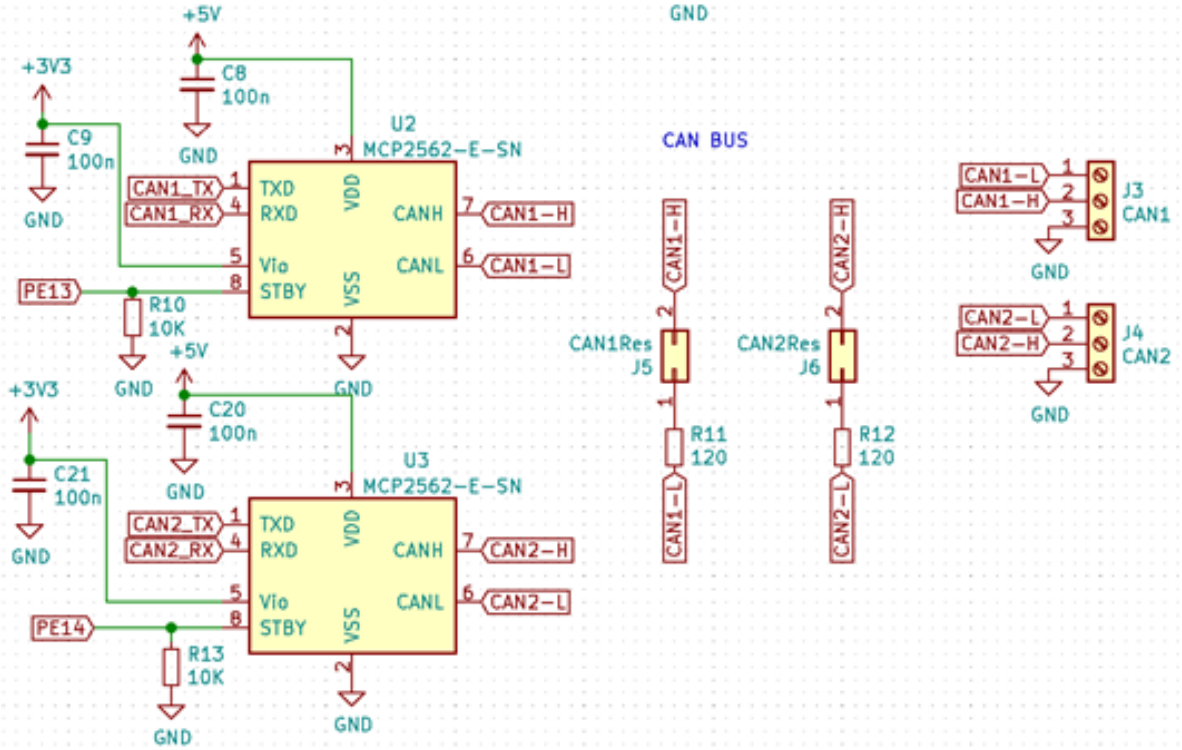

Figure A.33: MCP2562-E/SN implementation

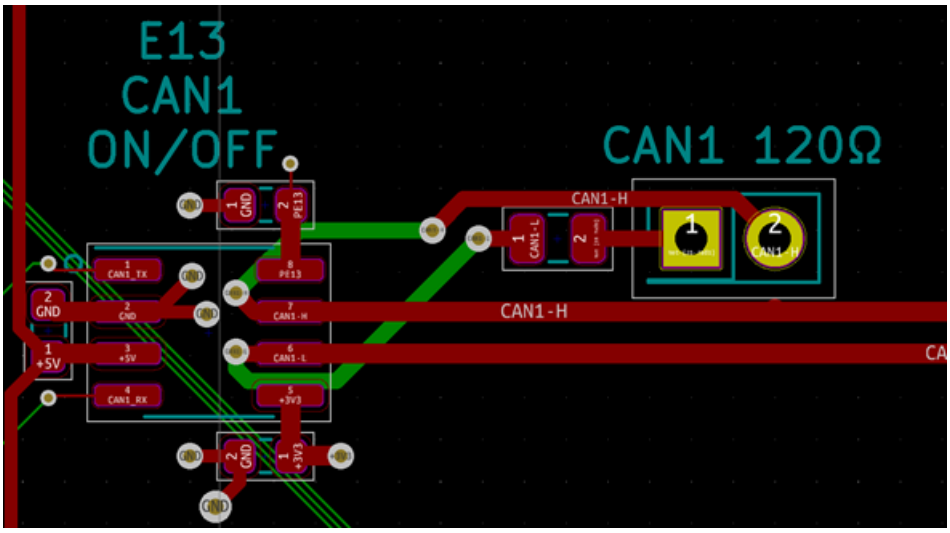

Figure A.34: MCP2562-E/SN layout

Appendix B

CARLA Client

The modifications on the manual_control.py provided by the CARLA Simulation development team are detailed in this chapter. The first modification is the creation of a socket object to connect with LabVIEW thru a TCP localhost

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('localhost', 10000)
sock.connect(server_address)
```

The next change is the definition of new global functions that are shown up next.

```
#takes a floating number and returns a string
#the string length is 3 including the sign
#the number must be lower to 100
def fromat_number(n):
    r = round(n,1)  #rounds float to a int
    a = abs(r)  #uses a variable to save the absolute value
    s = str(a)  #converts to string
    if n > 0:   #if the number is positive
        x = '0' #adds a zero at beginning of the string
    else:       #if the number is negative
```

```
        x = '-' #adds the sign
    if a > 9.99: #if the number is two digits
        x += s  #adds the two digits to the string
    else:       #if the number is one digit
        x += '0'
        x += s  #adds a zero and the digit
    return x    #returns the string

#formats a only positive three digit float
def fromat_speed(n):
    r = round(n)
    a = abs(r)
    s = str(a)  #up this point equal to las function
    if a < 10:  #if the number is one digit
        x = '00'
        x += s  #add two zeros and the digit
    elif a < 100:   #if is two digits
        x = '0'
        x += s  #add two zeros and two digits
    elif a < 1000:  #if it's three digits
        x = s   #add the digits
    else:
        x = '000'
    return x    #return the string


#create an ascii string to be sent thru TCP
#n0-2 are accelerations
#m0-2 are angular velocity
#sp is normalized velocity
def crate_mensage(n0,n1,n2,m0,m1,m2,sp):
    y = n0 + '/' + n1 + '/' + n2 + '/' + m0 + '/' + m1 + '/' + m2 + '/' + sp
    y = bytes(y, 'ascii')
    return y
```

On the KeyboardControl class definition, over the _parse_vehicle_keys method the original lines are commented or deleted to be replaced with the following lines.

```
data = sock.recv(17)    #recieve 17 character string from LabVIEW
data = data.decode('ascii') #decodes the message
x = data.split("/") #splits the message separated by /
#saves the parts or the message on variables
throttle_input = x[0]   #the first part is the throttle
steer_input = str(x[1]) #second is the steering wheel
brake_input = x[2]      #tirth is the brake
hand_brake_input = x[3] #fourth is the handbrake
rev = x[4]  #fifth is the reverse state
EXIT = x[5] #last ends the programm when asserted
throttle_input = float(throttle_input)
throttle_input = throttle_input/100 #formats the trothle 0-1 float
steer_input = float(steer_input)
steer_input = (steer_input-100)/100 #formats the steering 0-2 float
brake_input = float(brake_input)
brake_input = brake_input/100   #formats the trothle 0-1 float
#uptates the vehicle inputs using carla.VehicleControl() methods
#_control object was created in class constructor
self._control.throttle = throttle_input
self._control.steer = steer_input
self._control.brake = brake_input
if rev == '1':
    self._control.gear = -1
else:
    self._control.gear = 1
if hand_brake_input == '1':
    self._control.hand_brake = True
else:
    self._control.hand_brake = False
#returns True to end the program
if EXIT == '1':
    return True
else:
    return False
```

On the HUD class definition on the tick method, the following lines are added at the beginning.

```
#read the IMU sensor
v = world.player.get_velocity() #gets the linear velocity
x = world.imu_sensor.accelerometer  #gets acceleration
x0 = fromat_number(x[0])    #formats the acceleration
x1 = fromat_number(x[1])
x2 = fromat_number(x[2])
#reads the gyroscope
va = world.imu_sensor.gyroscope #gets the angular velocity
v0 = fromat_number(va[0])   #formats angular velocity
v1 = fromat_number(va[1])
v2 = fromat_number(va[2])
#calculates the nominal linear velocity
vel = (3.6 * math.sqrt(v.x**2 + v.y**2 + v.z**2))
vel = fromat_speed(vel) #formats the speed
#creates the message to be sent to LabVIEW
message = crate_mensage(x0, x1, x2,v0,v1,v2,vel)
sock.sendall(message)   #sends the message to LabVIEW
```

To be able to end the LabVIEW program by using the keyboard on the game_loop function the following lines are added just before pygame.quit()

```
#if the exit key on the keyboard is hit ends LabVIEW program
m = 'exit00000000000000000000000000123456789'   #exit code
m = bytes(m, 'ascii')   #formats exit code
sock.sendall(m) #sends exit code
sock.close()    #closes TCP socket
```

Appendix C

LabVIEW Program

The LabVIEW program executes using 5 frames of a "Flat Sequence" control structure, in the next figure shows the titles of each frame. Depending on the frame a local variable that is modified to inform the user on the front panel the state of the simulation.

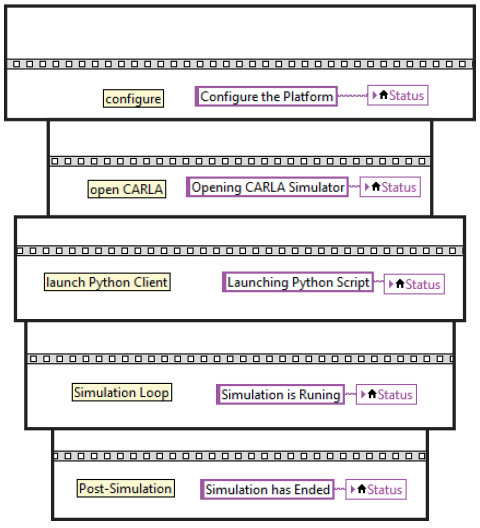

Figure C.1: Flat Sequence frames

C.1 Configure

This is the first frame that's executed. this frame executes a "While" control structure that stops when a error happens or the finish button is activated. When the finish button is activated the software check for error blocks the execution of the next frame until the error is fixed. To do so, the next figure shows the process that checks if the path and platform's dimensions are valid. A error log is displayed if that happens.

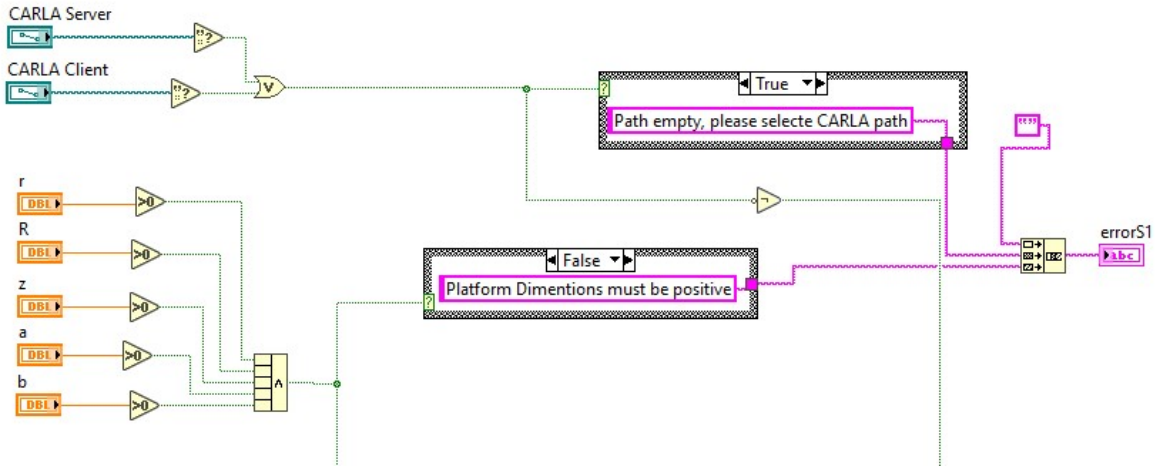

Figure C.2: Error checking

The next figure shows the error checking process. The error statements and the finish button must be True to end the While loop and continue to the next frame. **If more error conditions need to be added this is the section to do so by increasing the size of the "compound arithmetic" block or adding more logic to this section.** If the logic grows too much a subVI should be implemented instead.

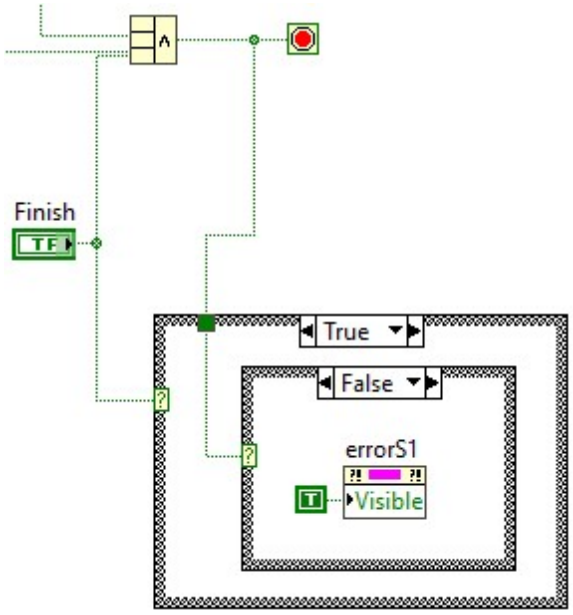

Figure C.3: Error checking

On the configuration frame is possible to add external devices like the platform, it's also possible to test the devices. When the "Set Motor Position Manually" button is activated, the sequence shown on the next figure is executed. This sequence stops the execution of the rest of the program until the "Set Current Position as Home" button has been activated. This block connects to the COM port specified, executes a while loop that sends the position of each individual motor and when it ends, closes the communication. This sequence also grays out the rest of the interface while executing and hides itself after executing. **This section need more features like error handling.** It's recommended to hide the code on a subVI after the improvements.

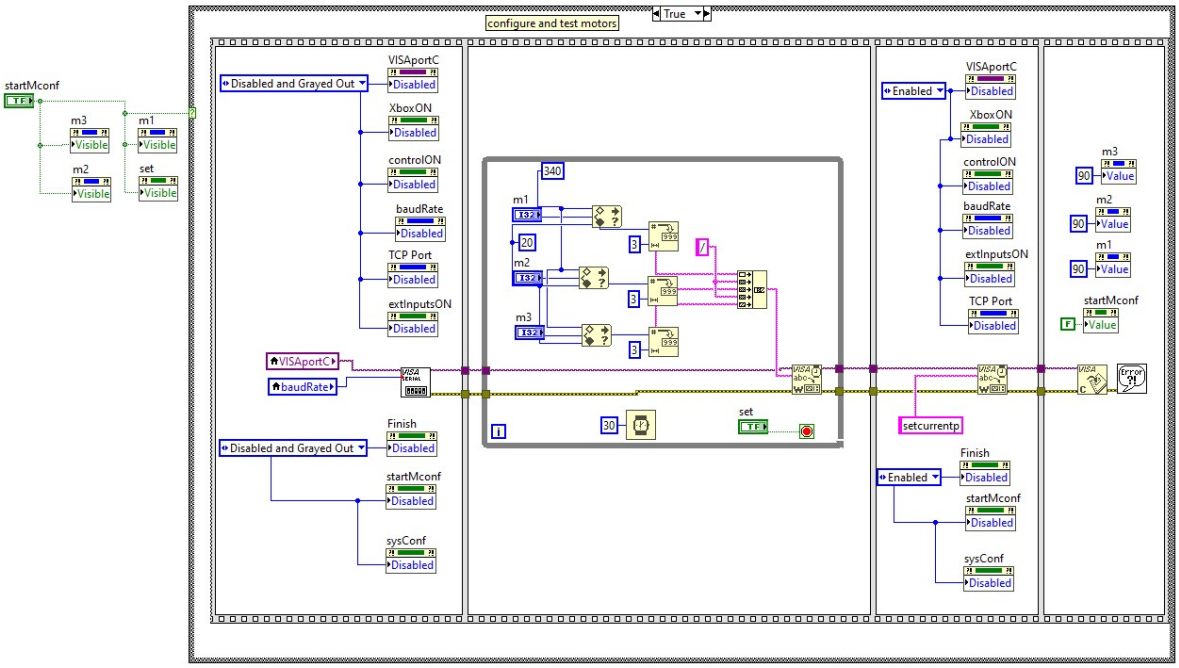

Figure C.4: Test motor sequence

Similarly to the motor testing position, the following figure shows the sequence to test the input devices. It starts when the "Test Input Devices" is activated and it ends when the "Finish Testing" is activated. **This sequence also needs the same improvements as the last sequence.**

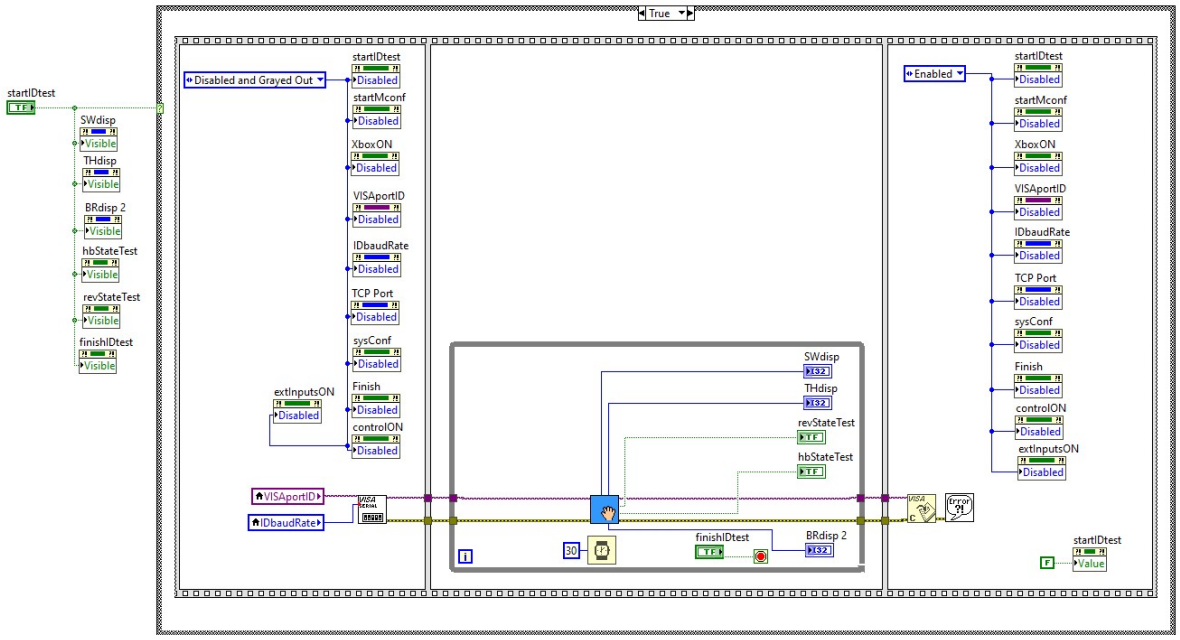

Figure C.5: Test input devices sequence

This frame sets many variables by using buttons and controllers, most of them are shown on the next figure. All of those controllers values are acquired letter on by using local variables. As a general rule, all of then should be used on the same way, but that is not the case on this implementation. As future work **all controllers must be accessed by local variables to reduce the wires on the block diagram window**

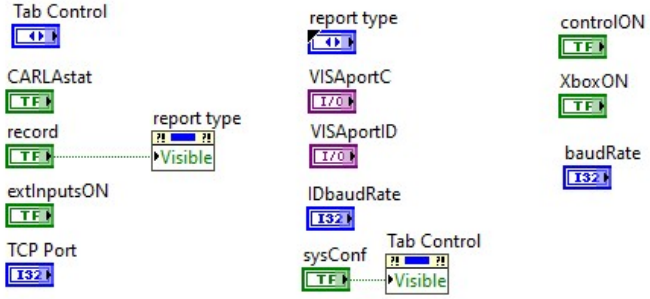

Figure C.6: configuration controllers

All frames show and hide the things that are not being used on that frame. This is implemented by using "Property Nodes" of the blocks. The next figure shows the property nodes used used on this frame, all of which are outside of the main while loop of the frames.

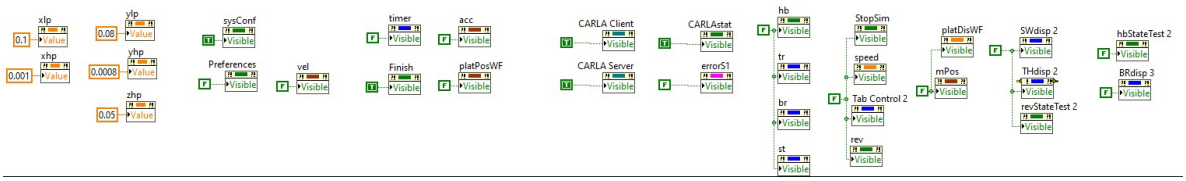

Figure C.7: Property Nodes

The following figure show the control of the property nodes to show the controllers of the motor and input devices testing. This is an example of things that should be implemented with local variables.

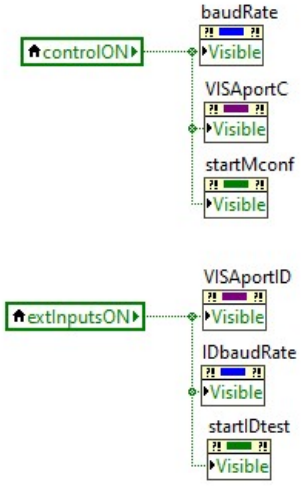

Figure C.8: Testing sequences property nodes

C.2 open CARLA

This frame is the simplest one, it opens the CARLA server by creating a Windows command line path provided on the first frame. It executes the command with the "System Exec" block, after that waits for 40 seconds for the program to open. This is and "open loop" control, therefore, **the program must communicate with the CARLA server to know when its running** and end the while loop when that happens. This is left as future work.

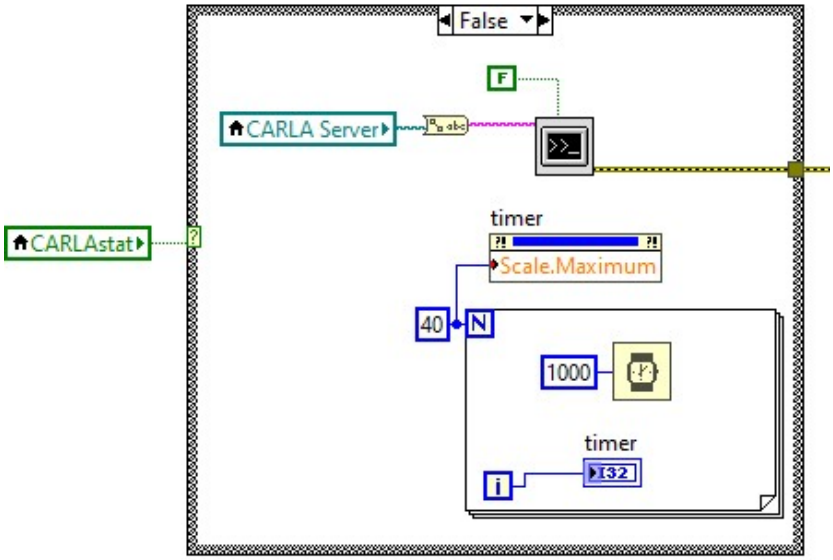

Figure C.9: Open CARLA server

The next figure shows the property nodes used to hide the controllers and displays not used on this frame. Also shows a timer that shows the remaining time.

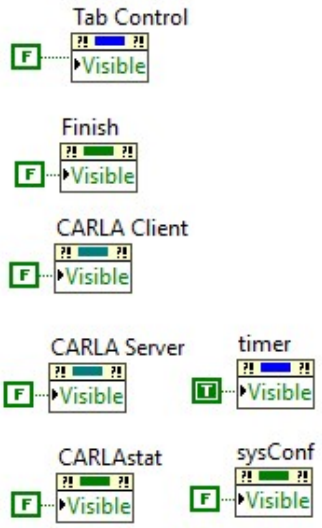

Figure C.10: Proprety nodes

C.3 Launch Python Client

This frame launches the CARLA client by executing a command line. The Windows command line is created with the path provided on the configuration frame and the python version command. The current version of the CARLA client script accepts arguments, while the currently defined arguments are not useful, new ones can be added if that is implemented **the command line must include arguments to control the client with the moderator software**, the most compelling option would be a map selection. This is left as future work.

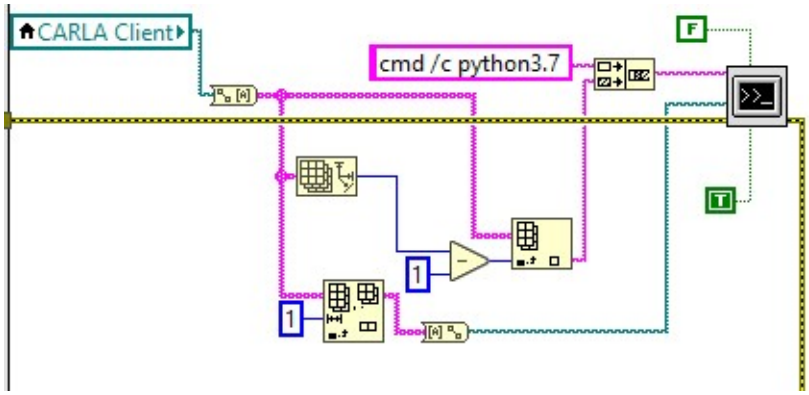

Figure C.11: Launch CARLA client

On this frame the communication sockets are opened. The only one that executes unconditionally is the CARLA TCP socket. The next figure shows the open socket blocks for the CARLA client and the platform controller that uses a serial socket. The open socket blocks also configure the communication with variables provided by the user on the configuration frame.

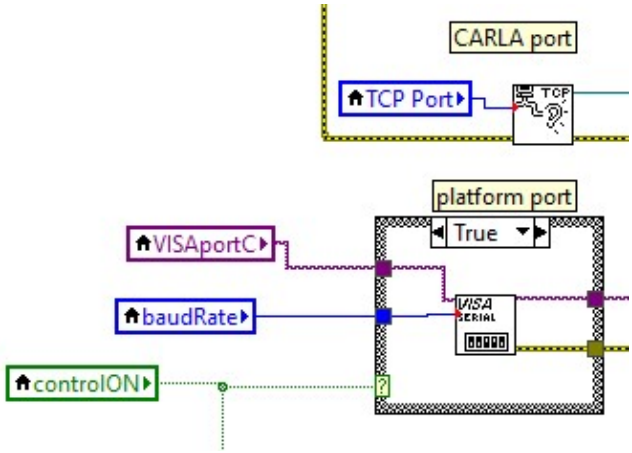

Figure C.12: Open CARLA and platform ports

The following figures show the configuration for the ports used to connect with the Xbox control and the vehicle control devices.

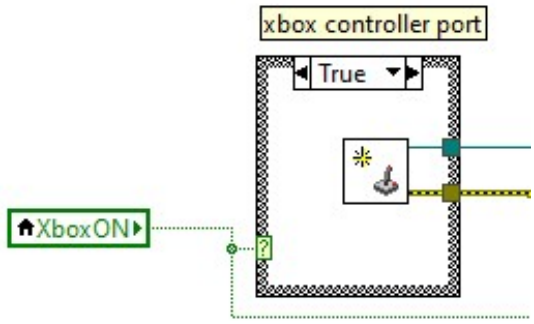

Figure C.13: Open Xbox port

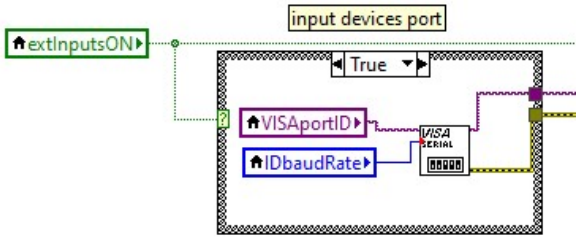

Figure C.14: Open vehicle control devices port

C.4 Simulation Loop

This frame is the one that executes the simulation run time. It uses a while loop with a 30ms delay, while on the current system all logic executes faster, the CARLA server tends to have a non stable execution time, therefore its better to have a fixed time rate bigger than the execution and small enough to be considered “real time”.

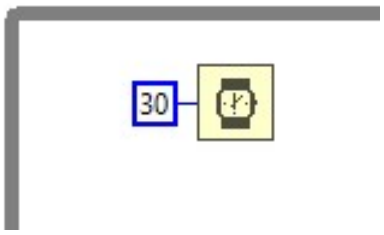

Figure C.15: Execution time

If any other input is selected, like Xbox of vehicle control devices where configured on the configuration frame with a selector it's possible to change the input signal. **If more input devices are added, a new case must be added to this case structure and it's logic must be added to that case**.

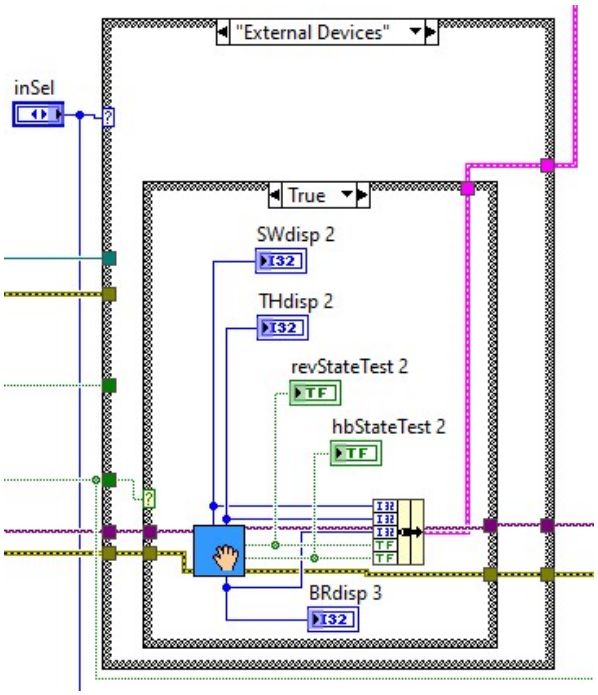

Figure C.16: Input devices case structure

The program also change the front panel depending on the current input device, the next figure shows how that's done.

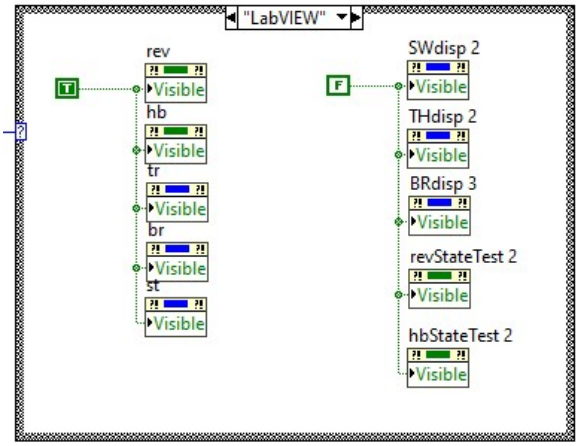

Figure C.17: Property nodes of for each input device type

The simulation run time front panel allows the modification of some parameters wihle the simulation is running. When the "Preferences" button is activated it shows a tab control that contains many controllers inside 3 tabs. The next figure shows how that's implemented.

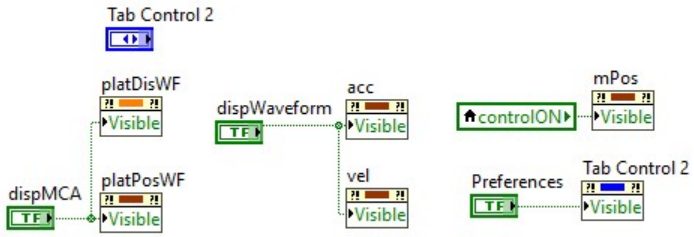

Figure C.18: Preferences tab control

The most important section of this software is the implementation of the communication with the CARLA simulator, the calculation of the MCA and the inverse kinematics. The following figure shows that part of the code, those tasks were implemented in subVIs since the complexity is considerable. **To modify any of those three processes just modify the subVI**, any modification on the connectors would cause all of the to disconnect on this VI, nevertheless some of the input/output signals could be passed as clusters to avoid the wire mess that currently is this implementation.

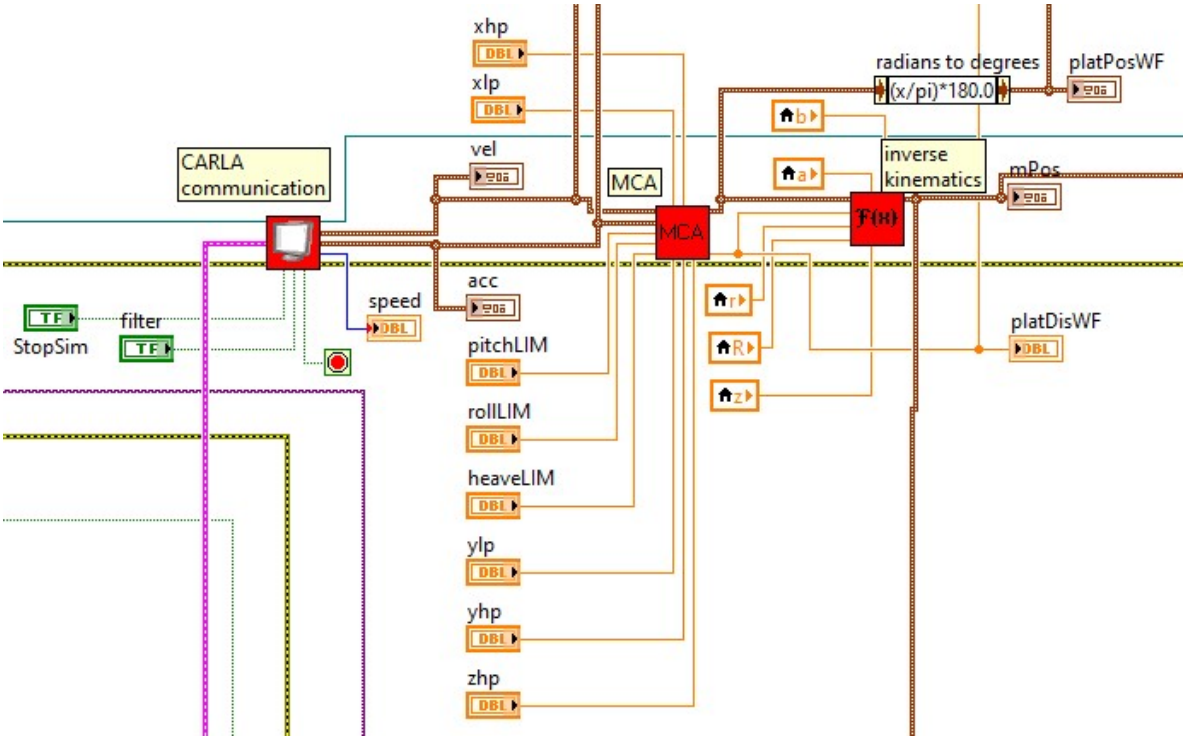

Figure C.19: CARLA connection, MCA and inverse kinematics

To communicate with the platform, LabVIEW needs to send the position of each motor in a single string with a fixed length. The following figure shows the sequence that does that, while in a case loop that only executes if the platform was configured on the configuration frame. **This logic could be in a subVI**.

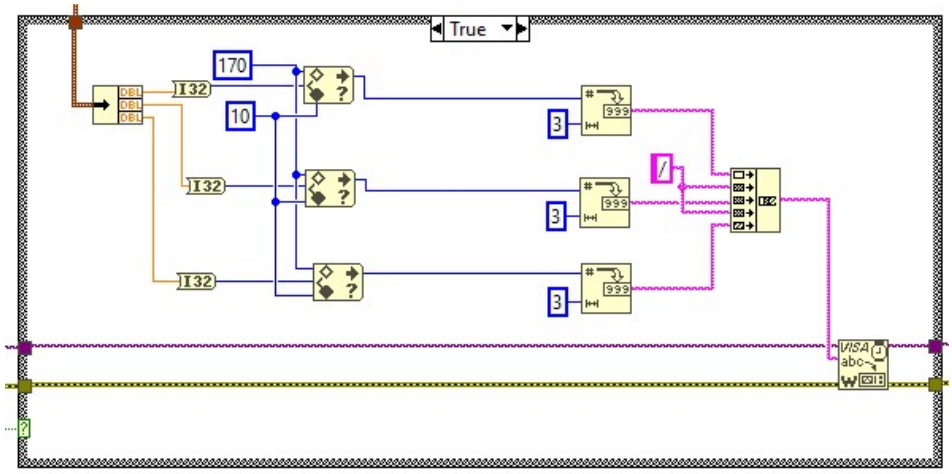

Figure C.20: Embedded controller communication

To save some variables of the simulation to be recorded in a external file after the simulation ends, the signals of such values go out of the while structure **with the tunnel mode set as indexing**. This is a pretty inefficient way or recording data of the simulation run time, as future work **a better solution to this problem should be implemented**.

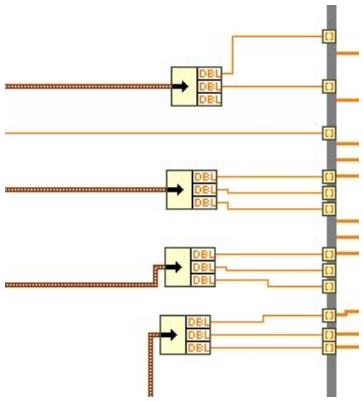

Figure C.21: Getting signals out of the while structure

C.5 Post-Simulation

This is the last frame of the flat sequence structure. The recorded signals are saved with a "Create Report" block only if the recording was enabled on the configuration frame. The next figure shows the case structure where this feature was implemented.

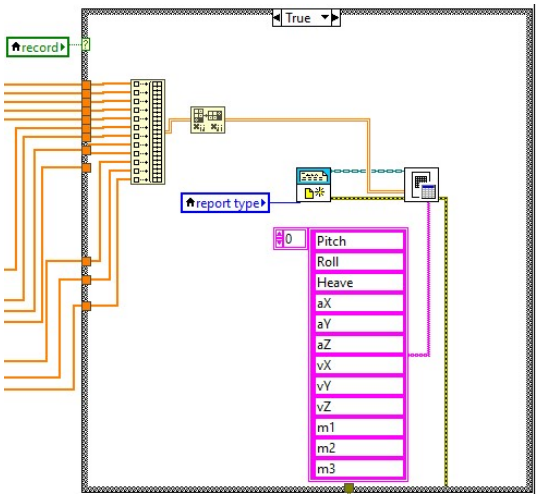

Figure C.22: Crate Report

This frame also closes all the communication ports used, if they where used. The following figure shows the three close block.

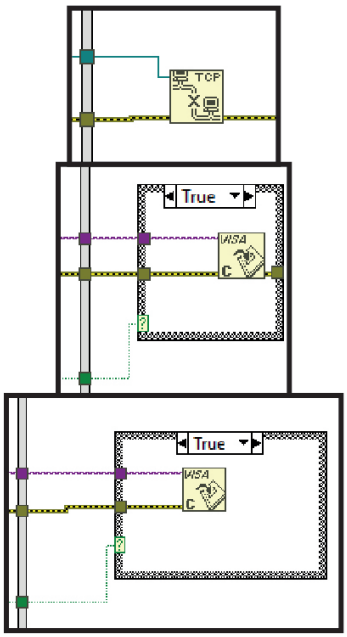

Figure C.23: Close communication

Finally, the software has a small error reporting feature. **A robust error handeling featur must be implemented** as future work.

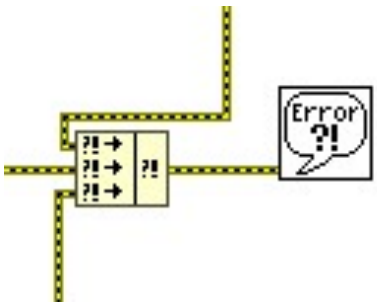

Figure C.24: Error handling

Appendix D

MicroPython Program

This chapter details the software that runs on the embedded system.

```
from pyb import Pin     #MicroPython library to control the GPIO
from pyb import Timer   #MicroPython library to control the Timers
from pyb import UART    #MicroPython library to control the UART

#encoder's pulses per revolution
ppr = 1056

#UART port configuration
buff_length=11      #length of the incoming reference signal
PueSer = UART(3)    #create a UART object for UART 3 that is goes to USB/UART conv.
PueSer.init(11560,bits=8,parity=None,stop=1,timeout=0,rxbuf=buff_length)
#Initialize the UART port setting the buffer size and speed

#Encoder pins configuration
pin_a = Pin('A0', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF1_TIM2)
pin_b = Pin('A1', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF1_TIM2)
#create a pin object without pull up or pull down and with
```

```
#the alternate function 1 for the selected timer

pin_c = Pin('E9', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF1_TIM1)
pin_d = Pin('E11', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF1_TIM1)

pin_e = Pin('D12', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF2_TIM4)
pin_f = Pin('D13', Pin.AF_PP, pull=Pin.PULL_NONE, af=Pin.AF2_TIM4)

#encoder timer configuration
enc_timer1 = Timer(2, prescaler=0, period=ppr)
enc_timer2 = Timer(1, prescaler=0, period=ppr)
enc_timer3 = Timer(4, prescaler=0, period=ppr)
#creates a timer object with the number of pulses in the encoder

enc_channel1 = enc_timer1.channel(1, Timer.ENC_AB)
enc_channel2 = enc_timer2.channel(1, Timer.ENC_AB)
enc_channel3 = enc_timer3.channel(1, Timer.ENC_AB)
#creates an object of a timer channel and set the mode to encoder AB

#Pins used to measure the system's performance
debug = Pin(Pin.cpu.D10, Pin.OUT_PP)    #creates a pin object set as output
debug_1 = Pin(Pin.cpu.D11, Pin.OUT_PP)
debug_2 = Pin(Pin.cpu.D8, Pin.OUT_PP)
debug_3 = Pin(Pin.cpu.D9, Pin.OUT_PP)
debug_4 = Pin(Pin.cpu.B14, Pin.OUT_PP)
debug_5 = Pin(Pin.cpu.B15, Pin.OUT_PP)
debug.off()     #sets the pin to low
debug_1.off()
debug_2.off()
debug_3.off()
debug_4.off()
debug_5.off()

#Pins to control the direction of the motors
D1 = Pin(Pin.cpu.C4, Pin.OUT_PP)
Iz1 = Pin(Pin.cpu.C5, Pin.OUT_PP)
D1.off()
```

```
Iz1.off()

D3 = Pin(Pin.cpu.E4, Pin.OUT_PP)
Iz3 = Pin(Pin.cpu.E3, Pin.OUT_PP)
D3.off()
Iz3.off()

D2 = Pin(Pin.cpu.C1, Pin.OUT_PP)
Iz2 = Pin(Pin.cpu.C0, Pin.OUT_PP)
D2.off()
Iz2.off()

#PWM pins to control the speed of the motor
En1 = Pin('A3') #creates an uninitialized pin object
tim1 = Timer(5, freq=900)   #creates a timer object initialized to 900Hz
ch1 = tim1.channel(4, Timer.PWM, pin=En1)
#creates a timer channel object in PWM mode
ch1.pulse_width_percent(0)  #turns off the pwm pin

En3 = Pin('E6')
tim3 = Timer(9, freq=900)
ch3 = tim3.channel(2, Timer.PWM, pin=En3)
ch3.pulse_width_percent(0)

En2 = Pin('B1')
tim2 = Timer(3, freq=900)
ch2 = tim2.channel(4, Timer.PWM, pin=En2)
ch2.pulse_width_percent(0)

#ESO constants
l0=100000000
l1=6000000
l2=110000
l3=600

#initialization of variables
tik=0   #flag to control the sampling rate
```

```
clutch=0    #variable to count the first iterations of the the program
fag=0
Vsource = 12        #Voltage source that powers the H-Bridge
Vmax = 9            #maximum voltage level for the motors
pwm_max = 100       #maximum level for the PWM with
initial_angle = 90  #Initial position for the motors
angle_shift = 90    #angle shifting to prevent 0 crossing
send_flag = 4       #flag to control the data sending

#initialization of control variables
hatx11=0
hatx21=0
z11=0
z21=0
hatx1old1=0
hatx2old1=0
z1old1=0
z2old1=0
xold1=0
v1=0,0

hatx12=0
hatx22=0
z12=0
z22=0
hatx1old2=0
hatx2old2=0
z1old2=0
z2old2=0
xold2=0
v2=0,0

hatx13=0
hatx23=0
z13=0
z23=0
hatx1old3=0
```

```
hatx2old3=0
z1old3=0
z2old3=0
xold3=0
v3=0,0

#function to map a value in a different range
def pymap(value, istart, istop, ostart, ostop):
    return ostart + (ostop - ostart) * ((value - istart) / (istop - istart))

#function that raises a flag if the sampling rate timer creates an interruption
def f(t):
    global tik
    tik=1

#function that formats the motor position
def readpos(x):
    position=pymap(x, 0, ppr, 0, 360)
    return position

#CONTROL LAW
def control(x,y,hatx,z):
    #control constants
    k1 = 90
    k2 = 20
    R=5.1
    Jm=0.050
    r=48
    Ka=0.010
    #calculate the error
    e = x-y
    eabs=abs(e)
    #calculate he control law
    if eabs < 1:
        u = 0               #if the error is small the control signal is 0
    else:
        u = -k1*(e)-k2*hatx #calculate the control signal
```

```
        u = u-z             #compensate the disturbances
    v = u*((Jm*r*R)/Ka)     #convert the control signal to a voltage signal
    #check if the voltage signal is within the voltage limits
    if v > Vmax:
        v = Vmax
    elif v < -Vmax:
        v = -Vmax
    return v,u  #return voltage and control signals

# functions to update the motors position and speed
def updateout1(vt1,Vsource):
    # motor 1
    # depending on the voltage value is the direction of the motor
    if vt1<0:
        D1.on()          #direction is updated
        Iz1.off()
    elif vt1>0:
        D1.off()
        Iz1.on()
    else:
        D1.off()
        Iz1.off()
    # the absolute value of the voltage is converted to a PWM width
    vt1 = abs(vt1)
    vt1 = pymap(vt1,0,Vsource,0,pwm_max)
    ch1.pulse_width_percent(vt1)    #speed is updated

def updateout2(vt2,Vsource):
    # motor2
    if vt2<0:
        D2.on()
        Iz2.off()
    elif vt2>0:
        D2.off()
        Iz2.on()
    else:
        D2.off()
```

```
        Iz2.off()
    vt2 = abs(vt2)
    vt2 = pymap(vt2,0,Vsource,0,pwm_max)
    ch2.pulse_width_percent(vt2)

def updateout3(vt3,Vsource):
    # motor 3
    if vt3<0:
        D3.on()
        Iz3.off()
    elif vt3>0:
        D3.off()
        Iz3.on()
    else:
        D3.off()
        Iz3.off()
    vt3 = abs(vt3)
    vt3 = pymap(vt3,0,Vsource,0,pwm_max)
    ch3.pulse_width_percent(vt3)

# function to make the output string always 3 characters long
def format_out(x):
    x = int(x)
    if x < 100:
        if x < 10:
            x = str(x)
            x = '00'+x
        else:
            x = str(x)
            x = '0'+str(x)
    else:
        x = str(x)
    return x

# pre-calculate variables
pwm_vmax = int(pymap(Vmax,0,Vsource,0,pwm_max))
#maximum value the PWM width can get to reach the maximum motor voltage
```

```
initial_pos = int(pymap(initial_angle + angle_shift, 0, 360, 0, ppr))
#calculate the initial position to initialize the enceder timer

# samplig rate configuration
t=0.0025    #sampling rate in seconds
tim7 = Timer(7, freq=400, callback=f)
#create at timer object initialized to create a interruption callback in 400Hz

# initialize encoder's timers
enc_timer1.counter(initial_pos)
enc_timer2.counter(initial_pos)
enc_timer3.counter(initial_pos)

# Control Loop
while True:
    #executes if the sampling rate timer has created an interruption
    if tik==1:
        tik=0       #lowers the interrupt flag
        debug.on()  #first aux pin set high to measure overall calculation time

        # Inputs
        debug_1.on()    #second aux pin set high to measure the motors position
        x1=enc_timer1.counter()     #reads motors actual position
        x1=readpos(x1)              #formats the position
        x2=enc_timer2.counter()
        x2=readpos(x2)
        x3=enc_timer3.counter()
        x3=readpos(x3)
        debug_1.off()       #second aux pin set low to measure the motors position
        #for the first 50 iterations the reference is
        #equal to the measured position to alow the ESO to converge
        if clutch < 50:
            y1=x1
            y2=x2
            y3=x3
            clutch=clutch+1
```

```
# Calculate the ESOs
# ESO1
debug_2.on()
#third aux pin set high to measure the ESO calculation time
hatx11 = hatx1old1 + t*(hatx2old1+l3*(xold1-hatx1old1))
#calculate estimated position
hatx21 = hatx2old1 + t*(v1[1]+z1old1+l2*(xold1-hatx1old1))
#calculate estimated velocity
z11 = z1old1 + t*(z2old1+l1*(xold1-hatx1old1))
#calculate estimated disturbance
z21 = z2old1 + t*(l0*(xold1-hatx1old1))
hatx1old1 = hatx11      #update last position
hatx2old1 = hatx21      #update last velocity
if z11 > 5000:      #limit estimated disturbance to avoid control loss
    z11 = 5000
elif z11 < -5000:
    z11 = -5000
z1old1 = z11        #update last disturbance
z2old1 = z21
xold1=x1            #update last measured position

# ESO2
hatx12 = hatx1old2 + t*(hatx2old2+l3*(xold2-hatx1old2))
hatx22 = hatx2old2 + t*(v2[1]+z1old2+l2*(xold2-hatx1old2))
z12 = z1old2 + t*(z2old2+l1*(xold2-hatx1old2))
z22 = z2old2 + t*(l0*(xold2-hatx1old2))
if z12 > 5000:
    z12 = 5000
elif z12 < -5000:
    z12 = -5000
hatx1old2 = hatx12
hatx2old2 = hatx22
z1old2 = z12
z2old2 = z22
xold2=x2

# ESO3
```

```
hatx13 = hatx1old3 + t*(hatx2old3+l3*(xold3-hatx1old3))
hatx23 = hatx2old3 + t*(v3[1]+z1old3+l2*(xold3-hatx1old3))
z13 = z1old3 + t*(z2old3+l1*(xold3-hatx1old3))
z23 = z2old3 + t*(l0*(xold3-hatx1old3))
if z13 > 5000:
    z13 = 5000
elif z13 < -5000:
    z13 = -5000
hatx1old3 = hatx13
hatx2old3 = hatx23
z1old3 = z13
z2old3 = z23
xold3=x3
debug_2.off()  #third aux pin set low to measure the ESO calculation time

# control calculation
debug_3.on()
#fourth aux pin set high to measure the control law calculation time
v1 = control(x1,y1,hatx21,z11)
#call the control calculation function for motor 1
v2 = control(x2,y2,hatx22,z12)
v3 = control(x3,y3,hatx23,z13)
debug_3.off()
#fourth aux pin set low to measure the control law calculation time

# Update output
debug_4.on()  #fifth aux pin set high to measure the output updating time
updateout1(v1[0],Vsource)   #call update function for motor 1
updateout2(v2[0],Vsource)
updateout3(v3[0],Vsource)
debug_4.off()   #fifth aux pin set low to measure the output updating time
debug.off()     #first aux pin set low to measure overall calculation time

# serial communication
#check if there's a full message in the buffer
if  PueSer.any() >= buff_length:
    debug_5.on()
```

```
            #sixth aux pin set high to measure the serial income communication time
            x = PueSer.read(buff_length)        #reads the UART buffer
            #"try"  to avoid program failure in case of miscommunication
            try:
                #decode the incoming message to a string format
                data= x.decode('ascii')
                #check if the received message is the set current position comand
                if data == 'setcurrentp':
                    #re-initialize encoder timer position
                    enc_timer1.counter(initial_pos)
                    enc_timer2.counter(initial_pos)
                    enc_timer3.counter(initial_pos)
                    #update reference position to initial position
                    y1 = initial_angle + angle_shift
                    y2 = initial_angle + angle_shift
                    y3 = initial_angle + angle_shift
                else:   #update reference position with message
                    #transform to integer the first 3 characters
                    y1 = int(data[0:3])
                    #add the angle shift to avoid zero crossing
                    y1 += angle_shift
                    y2 = int(data[4:7])
                    y2 += angle_shift
                    y3 = int(data[8:])
                    y3 += angle_shift
            except:
                pass    #do not update the reference position if any error occurs
            debug_5.off()
# END of normal program uncomment the next section to send back motor 3 position
        #     strtemp = format_out(x3)
        #     send_flag = 0
        # else:
        #     if send_flag <= 2:
        #         PueSer.write(strtemp[send_flag].encode('ascii'))
        #         send_flag = send_flag + 1
        #         debug_5.off()
        #     debug_5.on()
```

```
#     x3=int(x3)
#     if x3 < 100:
#         x3 = str(x3)
#         x3 = '0'+x3
#     else:
#         x3 = str(x3)
#     debug_5.off()
```

Appendix E

Publications

Further Results on Modeling and Control of a 3-DOF Platform for Driving Simulator Using Rotatory Actuators

Iván Cañedo Farfán, Roberto Carlos Ambrosio Lázaro, and José Fermi Guerrero Castellanos

Abstract The present work aims to develop a mathematical model and position control algorithm of a 3-RRS (revolute, revolute, spherical) parallel manipulator, for driving simulators. For this purpose, the inverse and forward kinematics are considered. Furthermore, the actuator model, DC-Motors, is taken into account. A simulation-based approach is presented using MATLAB/Simulink with Simscape complement for the mechanical constraints. Then, each actuator's position control is performed using the ADRC methodology, which provides robustness and simplicity in its implementation. Simulation results validate the proposal and the overall performance of the system is measured and compared with a PID-based solution using the ISE index.

Keywords Driving simulator · Parallel manipulator · Inverse kinematics · ADRC · 3-RRS

1 Introduction

Driving simulators are tools that allow performance of various tasks, like training new drivers or carrying out tests in a controlled environment that would otherwise be dangerous or expensive to perform in a real vehicle. Therefore, universities and industrial laboratories have been using driving simulators since the early 70s [1].

Nowadays, driving simulators have become more accessible, given that computational capabilities have been increasing in power and decreasing in price yearly. Thus, many configurations have been developed, ranging from static (0-DOF) sim-

I. Cañedo Farfán (✉) · R. C. Ambrosio Lázaro · J. F. Guerrero Castellanos
Benemérita Universidad Autónoma de Puebla, Puebla PUE, Mexico
e-mail: ivan.canedo@alumno.buap.mx

R. C. Ambrosio Lázaro
e-mail: roberto.ambrosio@correo.buap.mx

J. F. Guerrero Castellanos
e-mail: fermi.guerrero@correo.buap.mx

K. L. Flores Rodríguez et al. (eds.), *Recent Trends in Sustainable Engineering*,
Lecture Notes in Networks and Systems 297,
https://doi.org/10.1007/978-3-030-82064-0_6

ulators to systems with 9-DOF or more. The minimum DOF needed to determine the driving simulator's validity is closely related to the type of task that the driving simulator will be used for [2]. Ergo, a compromise between fidelity of the simulation and the minimum DOF must be reached. Therefore, a modeling and simulation of the control algorithm and inverse kinematic of a 3-DOF parallel platform using ADRC is presented in this work.

1.1 Motion-Based Driving Simulators

Recently, there has been an increasing interest in research for driving simulators with different configurations for automotive applications. The most widely accepted configurations are parallel manipulators (PM), which can be categorized by the type of joint on the kinematic chains that move the platform. The most popular configuration is the SP configuration, where the kinematic chains are conformed by two spherical passive joints and an actuated prismatic joint, this configuration is the best documented so far, but it is not the only existing configuration, other popular configurations include the RRS, RSS and RRR, where R stands for revolute joint.

To control the position of a moving platform two components are needed: the model of the platform, and the control of the actuators, those components have the goal of tracking a reference. This reference is a signal that contains the desired position of the platform. Since the platform being considered is meant to be used in a driving simulator, the reference signal is time variant, not deterministic and aperiodic, with low and high frequency components. Therefore, the validation of the tracking system can be more complex than the validation of a reference tracking system with a well-known reference signal.

1.2 Previous Works

In this section, a review of previous works regarding the modeling and control of parallel manipulators is presented. There are, mainly, three subjects of study regarding parallel manipulators: (1) the inverse model, (2) the forward model, and (3) the workspace calculation. When the platform uses an inverse model, the reference is the desired platform position and the output is the actuator's desired position. In a forward model, the reference is the desired actuator position and the output is the platform position. The workspace calculation consists in the determination of all possible positions that the platform can reach considering the mechanical constrains of the specific configuration.

Regarding the 3-DOF parallel manipulators, like its 6-DOF counterpart, the most common configuration for the 3-DOF parallel manipulators is based on actuated prismatic joints, but the main difference is that instead of using an SP confirmation, the most common configuration is RPS (revolute, prismatic, spherical) [3, 4]. Thus,

regarding the modeling of the platform, there are two approaches to the problem, the inverse kinematics, and the dynamics modeling, in reference [5] a comparison between the kinematics and dynamics of an RRS and an RSR configuration is performed considering the complexity, workspace, and singularities of every model; in [6] a velocity an acceleration analysis was performed for the RRS configuration, in [7] the inverse kinematics was solved using a conformal geometric algebra approach and the algorithm was validated with a small scale experimental platform.

2 Problem Statement

Medium-scale motion-based commercial driving simulators using a 3-DOF parallel manipulator with an RRS configuration and DC gearmotors as actuatora have recently become popular due to their capability of providing a motion feedback cue to the simulator operator. The advantage of a more affordable price than its 6–9 DOF counterpart, makes it more attractive for research in the field of driving simulators, nevertheless, real performance of those systems along with modeling and control strategies used are not openly available.

Driving simulators are complex systems with many interrelated components, therefore, to represent the system comprehensively it is necessary to simplify some components, thus a block diagram of a driving simulator is shown in Fig. 1, in this diagram the system is represented as a control problem focusing on the control of the platform's movement. To generate the platform reference position, it is necessary to have the acceleration and angular velocity of the simulated vehicle and process them in a motion cueing algorithm (MCA). The MCA is designed to transform the velocity and acceleration experimented by the vehicle to movements of the platform that have an equivalent stimulation in the vestibular organ of the driving simulator operator. With the desired position of the platform obtained from the MCA, the next step is controlling of the platform per se, the proposed control comprises the inverse kinematics of the platform and the control of each motor. To measure the

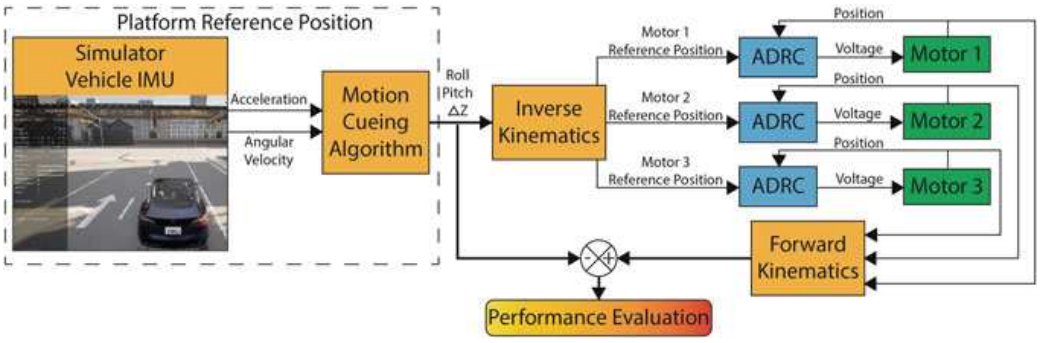

Fig. 1 Block diagram of the architecture of a driving simulator

performance of the control, the reference position is compared to the position of the platform obtained from forward kinematics.

As shown in Sect. 1.2, regarding the 3-DOF RRS parallel manipulator, most of the published works deal with the modeling of the platform. Nevertheless, the model of the motor actuating the revolute joint is rarely mentioned, Therefore, a simulation-based research can provide valuable information about the performance of a 3-DOF RRS parallel manipulator.

3 Modeling

The 3-DOF RRS parallel manipulator that is the focus of this research is shown in Fig. 2. Composed of two platforms and tree legs, the coordinate system *B-XYZ* is attached to the center of the base platform and it is considered to remain static. The coordinate system *P-xyz* is attached to the center of the moving platform, B_i (i = 1,2,3) are the connecting points to the base platform and similarly P_i are the connecting points of the moving platform, R_i are the union points for the passive revolute joints, L and l are the length of the links between $B_i - R_i$ and $R_i - P_i$ respectively, ℓ is the distance between B_i and P_i, the angle λ is the angular position of the actuated revolute joint B_i with respect to the vertical. The degrees of freedom of the system are roll, pitch and heave (ϕ, θ, $\triangle Z$).

3.1 Inverse Kinematics

Given that the separation between every point is 120°, the position for the union points of the base and platform are defined in (1) and (2).

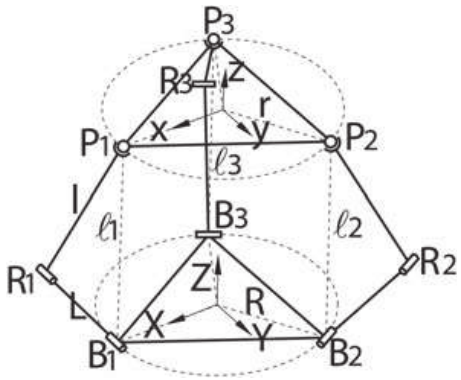

Fig. 2 The schematic diagram of the 3-RRS parallel manipulator

$$B_i = \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} R\cos a_i \\ R\sin a_i \\ 0 \end{bmatrix} \tag{1}$$

$$P_i = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} r\cos a_i \\ r\sin a_i \\ 0 \end{bmatrix} \tag{2}$$

with:

$$a_1 = 0^\circ, a_2 = 120^\circ, a_3 = 240^\circ \quad \text{and} \quad i = (1, 2, 3) \tag{3}$$

The next step is to calculate the new position of the platform, in order to do this, a translation vector T and a rotation matrix R are defined as a function of the three degrees of freedom of the platform. The translation vector is shown in (4) where the translation in the X and Y axis must be 0 and the translation on the Z axis ($\triangle Z$) is defined as the sum of the desired longitudinal translation and the platform height H, the height is set arbitrarily choosing a desired "home" position, on the other hand the rotation matrix is shown in (5) as a function of the desired orientation of the platform, where the rotation around the X axis and Y axis (roll, pitch) are ϕ and θ, the rotation around the Z axis is not taken into consideration, given that $\psi = 0$.

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ H + \triangle Z \end{bmatrix} \tag{4}$$

$$R = R_x(\phi)R_y(\theta)R_z(0) = \begin{bmatrix} \cos\theta & \sin\phi\sin\theta & \cos\phi\sin\theta \\ 0 & \cos\phi & \sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\sin\theta \end{bmatrix} \tag{5}$$

Next, the desired position of the platform points P_d is calculated using (6) with i = (1, 2, 3)

$$P_{di} = RP_i + T - B_i \tag{6}$$

Once the desired P_d is obtained, the next step is to calculate the angle λ, thus, three vertical planes are defined such that the points of P are on those planes. A graphical representation of this concept is shown in Fig. 3.

With the new planes defined, it is possible to model the kinematic legs, as shown in Fig. 4. And by using this geometric representation we can now calculate the angle λ, which is obtained by calculating the angles α, β an the line ℓ using (7) through (10).

$$\ell_i = \|P_{di}\| \tag{7}$$

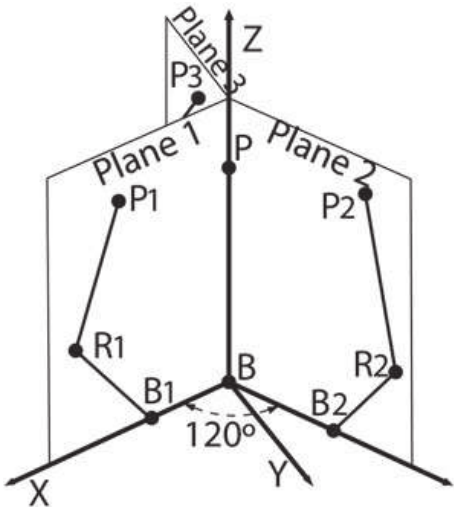

Fig. 3 Proposed vertical planes

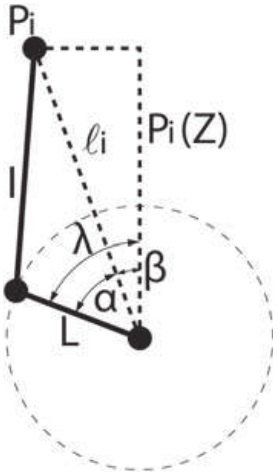

Fig. 4 Kinematic legs geometry

$$\alpha_i = \arccos \frac{L^2 + \ell_i^2 - l^2}{2\,L\ell_i} \tag{8}$$

$$\beta_i = \arccos \frac{P_{dzi}}{\ell_i} \tag{9}$$

$$\lambda_i = \alpha_i + \beta_i \tag{10}$$

3.2 Forward Kinematics

The problem of forward kinematics is more complex than the inverse kinematics, therefore a simulation based solution is presented, in this approach the position of the

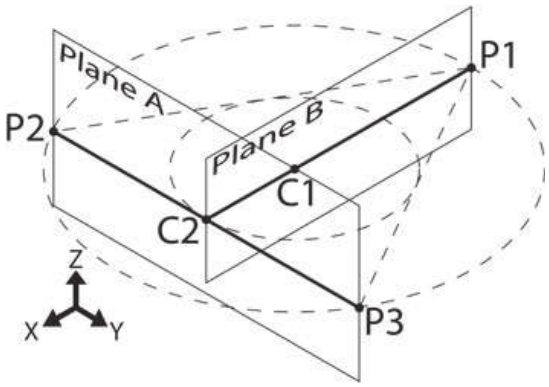

Fig. 5 Proposed geometry of the platform using points P_i

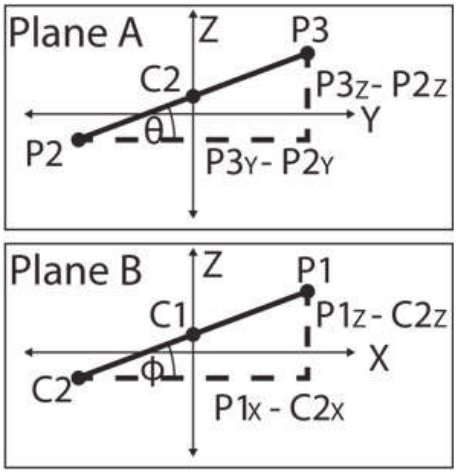

Fig. 6 frontal view of the planes generated from P_i

union points of the platform (P_{ri}) are assumed to be known, the proposed geometry for the platform its shown in Fig. 5, the Planes *A* and *B* defined in Fig. 5 are shown in Fig. 6, points *P2* and *P3*, defined in plane (A), are used to calculate the platform roll, then (11) is used to calculate the pitch of the platform (ϕ); (12) to calculate point *C2* on plane *A*; (13) together with *C2* on plane *B*, to calculate the pitch of platform (θ); and (14) is used to calculate heave ($\triangle Z$).

$$\phi = \arctan \frac{P3_Z - P2_Z}{P3_Y - P2_Y} \tag{11}$$

$$C2 = \frac{P2_Z + P3_Z}{2} \tag{12}$$

$$\theta = \arctan \frac{P1_Z - C2_Z}{P1_x - C2_x} \tag{13}$$

$$m = \frac{P1_X - C2_X}{P1_Z - C2_Z}$$
$$\triangle Z = P1_Z - mP1_X \tag{14}$$

4 Control Design

As regards control, the selected strategy must be capable of performing position tracking of a reference signal and compensating for external disturbances, on the other hand, the system must have a saturation that maintains the system within the workspace, in this research the maximum an minimum position of the actuator is arbitrarily selected since the scope of this project is not related with the determination of optimal performance to the workspace of the 3-RRS parallel manipulator.

4.1 Motor Model

The permanent magnet, brushed, DC gear-motor can be modeled in many ways, e.g. as a system of the first or second order, nevertheless, in this research the motor is represented as a perturbed double integrator, and (15) is proposed. In this equation the rotor acceleration is calculated as a function of the armature voltage and it takes into consideration the endogen perturbations as the factor including angular velocity, and the exogen perturbations as the factor that containing the torque applied to the motor. Once the motor's acceleration is calculated, it is integrated two times to obtain the position.

$$\ddot{q} = \left(\frac{K_a K_b}{J_m R} + \frac{B_m}{J_m}\right)\dot{q} + \frac{K_a}{J_m r R}V - \frac{\tau}{J_m r^2} \tag{15}$$

where:

- q = angular position.
- $\dot{q}$ = angular velocity.
- $\ddot{q}$ = angular acceleration.
- K_a = motor-torque constant.
- K_b = counter electromotive constant.
- J_m = rotor inertia.
- R = armature resistance.
- B_m = viscous resistance.
- τ = torque applied to the motor.
- r = gear relation.

4.2 ADRC Control

The selected control strategy is the active disturbance rejection control (ADRC) since it has the feature of compensating internal and external disturbances. This control strategy was first introduced in [8], the main principle behind this strategy is to reduce all perturbations to a single term that is compensated in the control law. To estimate the disturbance, an extended state observer (ESO) is implemented, this observer estimates the states of the system and the perturbation based on the control signal and the output of the system, the methodology to determine the observability, controllability and stability of the system are beyond the scope of this research, Fig. 7 shows the block diagram of the ADRC.

To design the control, (15) is modified to group all disturbances in a single term ζ, the resulting expression is shown in (16), then, making $\ddot{q} = u$, where u is the control signal, and leaving only v on the left side of the equation as shown in (17); defining the system states as (18), the next step is to define a control law, the proposed control law is stated in (19) where x_d represents the motor's desired position.

$$\ddot{q} = \frac{K_a}{J_m r R} V + \zeta \tag{16}$$

$$V = \frac{J_m r R}{K_a}(u - \zeta) \tag{17}$$

$$\begin{aligned} x_1 &= q & \dot{x_1} &= x_2 \\ x_2 &= \dot{q} & \dot{x_2} &= u + \zeta \end{aligned} \tag{18}$$

$$u = \ddot{x_d} - K_1(x_1 - x_d) - K_2(x_2 - \dot{x_d}) - z_1 \tag{19}$$

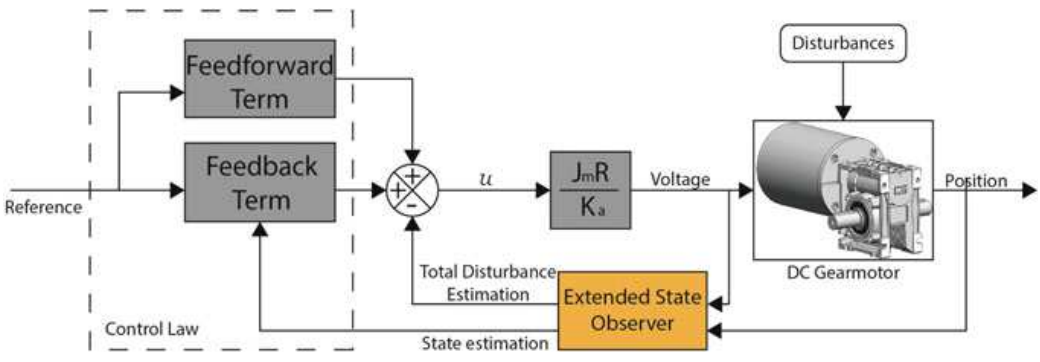

Fig. 7 ADRC block diagram based on [9]

where:

- K_i = control gains $i = (1, 2)$
- z_1 = estimated disturbance ($z_1 \approx \zeta$)

4.2.1 Extended State Observer

The ESO is a key feature of the ADRC since it can estimate the system disturbance z_1, and the states of the system $\hat{x_i}$ with $i = (1, 2)$, the equations that define the observer are shown in (20), these equations use the constants o_i with $(i = (0 - 3))$, these constant values must be carefully calculated since they rule over the speed of the observer, as a rule of thumb, the ESO must be faster than the controlled system but the exact values needed to tune the system require a fair amount of trial and error, nevertheless there is a useful methodology to calculate those values, a fourth-order Hurwitz polynomial is defined as shown in (21). Then the roots of that equation are used as the constants o_i. This leaves only the natural frequency ω_n and the damping factor ξ to be defined as shown in (22).

$$\begin{aligned} \dot{\hat{x_1}} &= \hat{x_2} + o_3(x_1 - \hat{x_1}) \\ \dot{\hat{x_2}} &= u + z_1 + o_2(x_1 - \hat{x_1}) \\ \dot{z_1} &= z_2 + o_1(x_1 - \hat{x_1}) \\ \dot{z_2} &= o_0(x_1 - \hat{x_1}) \end{aligned} \tag{20}$$

$$(s^2 + 2\omega_n\xi s + \omega_n^2)(s^2 + 2\omega_n\xi s + \omega_n^2) = 0 \tag{21}$$

$$\begin{aligned} o_3 &= 4\omega_n\xi \\ o_2 &= 2\omega_n^2(1 + 2\xi^2) \\ o_1 &= 4\omega_n^3\xi \\ o_0 &= \omega_n^4 \end{aligned} \tag{22}$$

5 Simulation Results

The simulation was performed using MATLAB/Simulink, the basis of the simulation is a 3D platform that uses the Simscape complement, the 3D model is shown in Fig. 8 in the block called "platform", in this model it is possible to visualize the platform's movement, measure the position of points P_i and measure the position and acceleration of points B_i. Those latter points are actuated using the output of the control strategy.

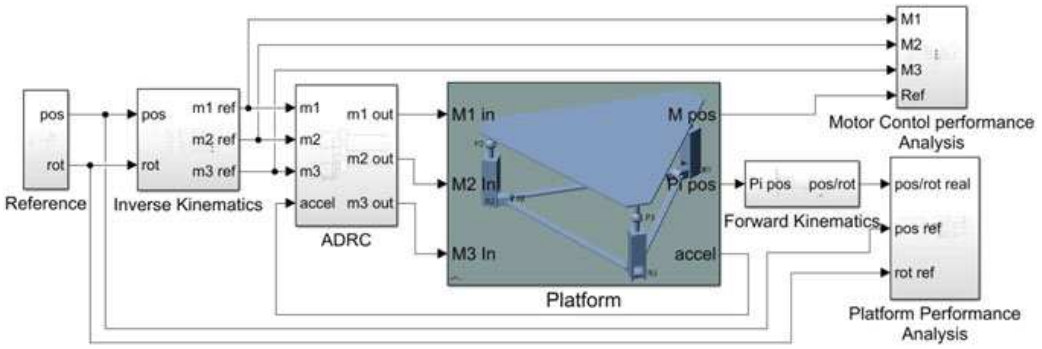

Fig. 8 MATLAB/Simulink simulation

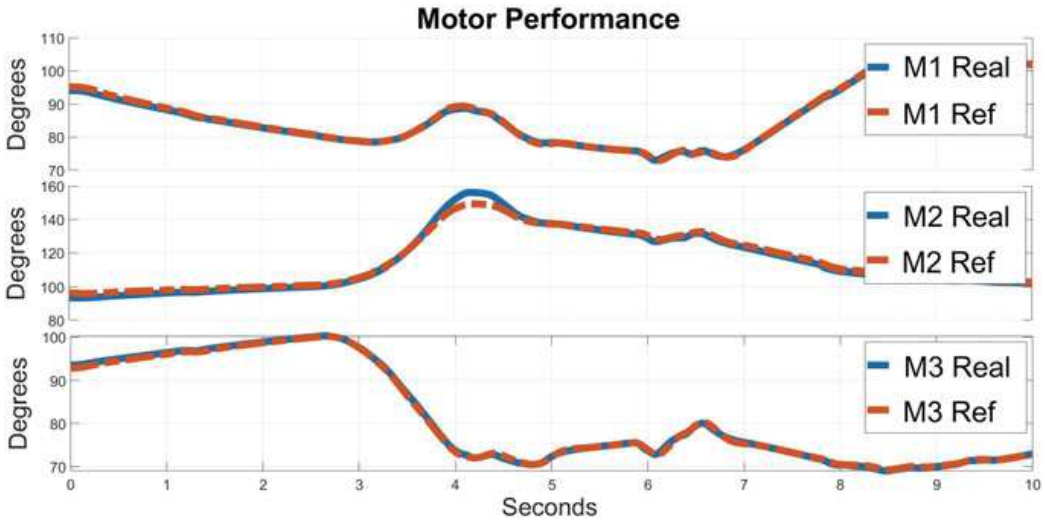

Fig. 9 DC motor position control performance analysis

The overall structure of the simulation is shown in Fig. 8, the reference position of the platform is being generated from an Excel sheet that contains the reference position of a driving simulator, the nature of this signal is beyond the scope of this research, nevertheless this signal is being used to measure the performance of the platform.

The control strategy was simulated using the motor model, the ADRC control and the ESO; at the same time the measurement of the acceleration in the joints is compared with the ideal acceleration of the motor and that difference is added as a disturbance, the performance of the control is shown in Fig. 9 where the plotted position is the measured position of the joints and the reference is the reference generated from the inverse kinematics.

The overall performance of the system is being measured comparing the input reference position (roll, pitch and heave) with the measured position of the platform, Fig. 10 shows the comparison between these signals. To visualize the performance of the system quantitatively, the performance index ISE (integral squared error) is calculated.

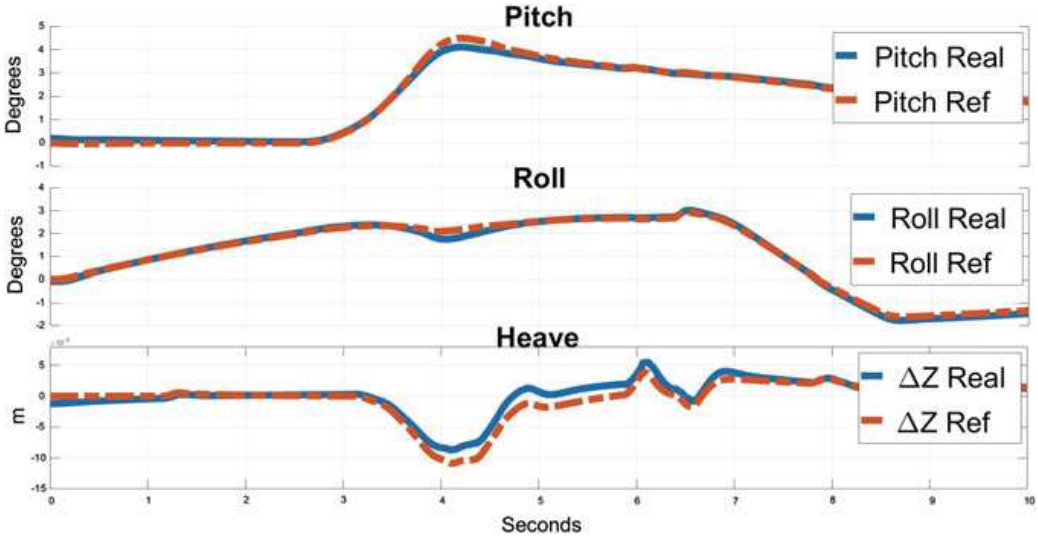

Fig. 10 Platform performance analysis

6 Discussion

The ISE index is used to compare the performance of the system with other proposals. The proposed system is compared with the ISE index of the system changing the ADRC with a PID controller, Fig. 11 shows the evolution of both indexes.. To obtain the PID performance, the controller was tuned with the PID Tuner provided by MATLAB/Simulink and the disturbance of the system was multiplied by several orders of magnitude with the goal of simulating a loaded platform. Therefore, while the inverse kinematics also plays an important role in the performance of the platform, the main source of error comes from the control strategy and the disturbances affecting the system.

Regarding the mathematical model of the platform, the RRS configuration, compared with the RP configuration, exchanges workspace for speed, also, given the non-linearity of the system, for the model to remain relevant, the angle (λ) must not get near 0° or 180°, where the model reaches a point where it fails completely, therefore, maintaining the system within its worspace is vital.

7 Conclusion

The inverse kinematics of a 3-RRS parallel manipulator is presented alongside the modeling of the actuators and the design of an ADRC for position tracking. The system was simulated with MATLAB/Simulink and simscape. The presented mathematical model of a 3-RRS parallel manipulator can follow the reference as long as the angle λ remains approximately between 10° and 170°.

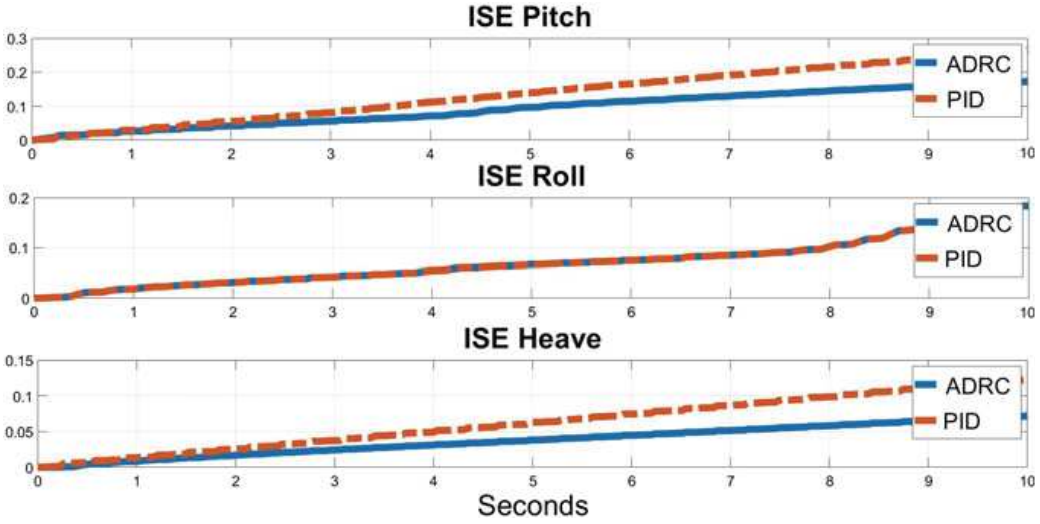

Fig. 11 Platform's performance ISE Index comparison

The simulated system can track the given reference and compensate the intrinsic and extrinsic disturbances caused by the platform weight, inertia, etc. The results obtained in this work are valuable references for the design and implementation of physical driving simulators. Further work for this model needs to be experimentally tested and implemented to measure the performance of the system.

References

1. Slob JJ (2008) State-of-the-art driving simulators, a literature survey. DCT Report. https://doi.org/10.10007/1234567890
2. Fisher DL, Rizzo M, Caird JK, Lee JD (2011) Handbook of driving simulation for engineering, medicine, and psychology. CRC Press
3. Brecht D (2015) A 3-DOF Stewart platform for trenchless pipeline rehabilitation. Sept 2015, pp 1–185
4. Büründcük K, Tokad Y (1999) On the Kinematic of a 3-DOF Stewart platform. J Robot Syst 16(2):105–118
5. Itul T, Pisla D (2009) On the kinematics and dynamics of 3-DOF parallel robots with triangle platform. J Vibroeng 11(1):188–200
6. Li J, Wang J, Chou W, Zhang Y, Wang T, Zhang Q (2001) Inverse kinematics and dynamics of the 3-RRS parallel platform. In: IEEE International Conference on Robotics and Automation, vol. 3, pp 2506–2511. https://doi.org/10.1109/ROBOT.2001.932999
7. Carbajal-Espinosa O, Izar-Bonilla F, Díaz-Rodríguez M, Bayro-Corrochano E (2016) Inverse kinematics of a 3 DOF parallel manipulator: a conformal geometric algebra approach. In: IEEE-RAS international conference on humanoid robots, Dec 2016, pp 766–771. https://doi.org/10.1109/HUMANOIDS.2016.7803360

8. Huang H, Wu L, Han J, Feng G, Lin Y (2004) A new synthesis method for unit coordinated control system in thermal power plant—ADRC control scheme. In: 2004 International conference on power system technology, POWERCON 2004, vol 1, Nov 2004, pp 133–138. https://doi.org/10.1109/icpst.2004.1459980
9. Guerrero-Castellanos JF, Rifaï H, Arnez-Paniagua V, Linares-Flores J, Saynes-Torres L, Mohammed S (2018) Robust active disturbance rejection control via control Lyapunov functions: application to actuated-ankle-foot-orthosis. Control Eng Pract 80:49–60. https://doi.org/10.1016/j.conengprac.2018.08.008

Plataforma Virtual e Interactiva para la Conducción de Vehículo Basado en Modelo 3D de Unreal Engine Mediante Transmisión de Datos TCP/IP

1st Erick Gómez López
Facultad de Ciencias de la Electrónica.
Benemérita Universidad Autónoma de Puebla
Puebla, México
erick.gomezlo@alumno.buap.mx

2nd Roberto Carlos Ambrosio Lázaro
Facultad de Ciencias de la Electrónica.
Benemérita Universidad Autónoma de Puebla
Puebla, México
roberto.ambrosio@correo.buap.mx

3th Ivan Cañedo Farfan
Facultad de Ciencias de la Electrónica.
Benemérita Universidad Autónoma de Puebla
Puebla, México
ivan.canedo@alumno.buap.mx

***Abstract*—En el siguiente trabajo se presenta el desarrollo de una plataforma virtual e interactiva para la conducción de un vehículo. Se establece una comunicación de datos entre LabVIEW y Unreal Engine 4, para la adquisición y envió de información al modelo virtual 3D de un vehículo dentro de un escenario similar a un circuito de conducción. Esta manera de transmitir los datos entre ambos software descarta el uso de archivos dll(Dynamic Link Library), en su lugar se utiliza hardware para lograr un transmisión mediante el protocolo de red TCP/IP, y una conexión serial. El hardware utilizado se basa en el dispositivo SoC(System on Chip) ESP32. El vehículo y el escenario son diseñados y visualizados a través del motor de videojuegos, mientras que el mando de velocidad y dirección del vehículo es gestionado desde el software de automatización y control.**

***Index Terms*—Conducción de vehículo, virtual, Protocolo de red TCP/IP, ESP32, Unreal Engine 4, LabVIEW, Transmisión de datos.**

I. Introducción

Un simulador de manejo es una herramienta que recrea, en un contexto artificial, la situación de conducción de un vehículo[1]; los simuladores de manejo son herramientas fundamentales para estudios regionales, nacionales e internacionales que buscan probar la eficiencia de programas de entrenamiento para conductores, aptitudes para manejar en pacientes con distintos trastornos, efectos en la capacidad de conducción de los individuos que se encuentran bajo los efectos de algún medicamento, impacto de señalamientos en el comportamiento del conductor, analizar estrategias con el fin de disminuir los accidentes, y validar ventajas y desventajas de nuevas tecnologías introducidas al vehículo [2][3]. En el campo de la investigación, los trabajos más comunes sobre simuladores de manejo son principalmente dirigidos hacia el software, varias universidades han realizado sus propias versiones basadas en motores gráficos como Unity, Unreal Engine, etc. En su desarrollo han dejado de lado el diseño de los estimuladores y la comunicación vía remota [4][5][6][7][8].

Por otro lado, un software con enormes capacidades de desarrollo en áreas de control, automatización e instrumentación electrónica entre muchas otras, puede en ocasiones encontrarse muy limitado al realizar animaciones en 3D, puesto que no esta enfocado y especializado en estas tareas. Si se logra una colaboración entre este tipo de software y otro especializado en animaciones 3D, el universo de posibilidades se incrementa. De manera general, cuando pensamos en utilizar dos software distintos que trabajen en conjunto, lo primero en mente es utilizar algún complemento que permita conectar de manera sencilla a ambos. Para lograr la colaboración, la manera mas fundamental sería desarrollar una gestión de archivos dll(Dynamic Link Library) para vínculos dinámicos entre ambos software, lo cual implicaría entre otras cosas, la necesidad de conocimientos avanzados en este tipo de archivos, un amplio panorama de la arquitectura de los software con los que se desea trabajar y un mayor tiempo de desarrollo[9]. Por otro lado existe la posibilidad de adaptar herramientas de software y complementos disponibles para otras finalidades, aprovechando sus características y logrando que trabajen de la manera en que lo necesitamos, lo cual brinda una posibilidad mas rápida para resolver la necesidad, y representa un nivel de desarrollo de software menos exigente.

En este trabajo establecemos una comunicación de datos entre LabVIEW y Unreal Engine 4, logrando que ambos funcionen de manera colaborativa para conseguir conducir un vehículo dentro de un entorno virtual. Tanto la dirección de giro, como el avance frontal, son adquiridos desde LabVIEW, por medio de un mando de Xbox One compatible con windows, posteriormente son transmitidos a la red a través del protocolo TCP/IP. LabVIEW es una plataforma de programación gráfica desarrollada por National Instruments para el diseño de sistemas de pruebas y mediciones, el cual debido a su naturaleza gráfica hace que tareas como la vinculación y transferencia de datos con hardware externo a la PC sean mas sencillas[10].

La animación en 3D del vehículo como de su entorno son diseñados y renderizados desde Unreal Engine 4(UE4),

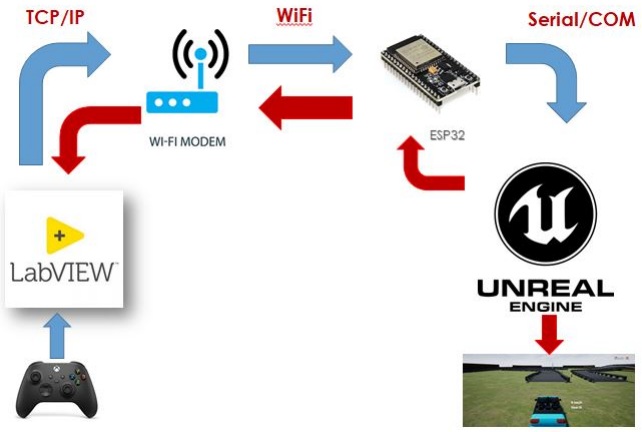

Fig. 1. Flujo de transmisión y recepción de datos a través del hardware para la comunicación entre los dos software.

un motor y editor de videojuegos gratuito desarrollado por la empresa Epic Games, utilizado para la producción de distintos tipos de videojuegos y aplicaciones, el cual tiene la posibilidad de desarrollar proyectos basados en C++, o como es el caso de este trabajo, proyectos basados en blueprints los cuales utilizan un ambiente de scripts visuales, en los que se programan funciones de una manera similar a la programación por bloques[11]. Parte de la comunicación de datos esta apoyada en el uso de UE4Duino un complemento para UE4, el cual permite establecer una conexión serial con un puerto COM. Cuando los datos son recibidos, UE4 se encarga de ejecutar las acciones necesarias para convertirlos en información y así renderizar la animación que obedece la nueva posición del vehículo.

Con el aumento de las necesidades de realizar actividades de manera remota, una transmisión de datos a través de la red utilizando un protocolo TCP/IP se vuelve bastante conveniente. Este tipo de transmisiones de datos se vuelven cada vez mas sencillas de implementar gracias a los avances que han sucedido en dispositivos pensados para proyectos de IoT, como lo es el desarrollo del ESP32, un SoC que involucra un procesador de doble núcleo, altas prestaciones en sus interfaces periféricas, tecnologías de comunicación WiFi y Bluetooth de modo dual, bajo consumo de energía y un bajo costo[12]. Dentro del trabajo que aquí se presenta, el ESP32 tiene la función de recibir mediante su conexión WiFi los datos provenientes de labVIEW que fueron enviados a través de la red, posteriormente los transmite hacia UE4 por medio de una comunicación serial en un puerto COM a través de un cable USB. El flujo de la transmisión y recepción de datos entre los diferentes elementos del sistema puede visualizarse en la figura 1.

II. Desarrollo

A. Adquisición de datos del mando

La adquisición de datos se realiza con base en los estados que se muestra en la figura 2, para comenzar se conecta el mando de Xbox One a la PC, posteriormente dentro de un proyecto se inserta un ciclo while que permita una ejecución continua, fuera de este ciclo iniciamos el reconocimiento del mando insertando la función *Open* que pertenece al grupo de herramientas *Maker Hub interface for Xbox One controller* y permitimos que permanezca en Auto Detect, dentro del ciclo while se anida un *eventstructur* para reconocer cuando sucede un evento en en mando de Xbox, dentro de esta estructura se agrega la función *Read* y por ultimo, fuera del ciclo infinito se cierra la conexión con el mando insertando la función *Close*, esta disposición de los bloques de funciones se muestra en la figura 3.

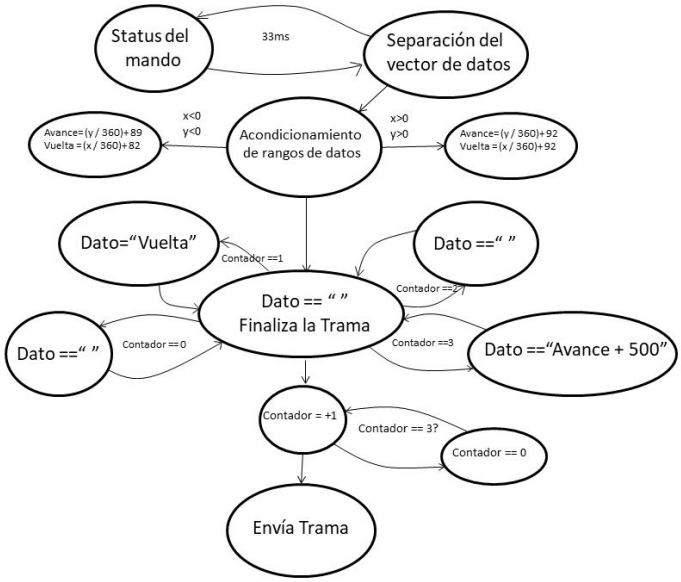

Fig. 2. Diagrama de estados para la adquisición, acondicionamiento y envió de datos.

Las palancas del mando tienen una resolución de 16 bits para cada eje, por lo que los datos de lectura en labVIEW, pueden tomar valores desde -32768 hasta 32768 en cada uno de los ejes y son entregados por medio de un vector de datos que incluye todos los valores de posición, de las palancas de mando, gatillos y D-pad. Se deben separar los datos del vector y deben ser convertidos en información útil. Así que tal y como se observa en la figura 3, al vector de datos se le aplica un primer bloque de función *Unbundle*, lo que separa los datos de la palanca de mando izquierda, un segundo *Unbundle* entrega los datos separados del eje x y el eje y, los cuales brindaran el giro y la aceleración respectivamente. Para obtener la información que será enviada, convertimos los datos del rango de 16 bits a un rango simple de 1 a 180, esto lo logramos con la ayuda de un bloque de función *Case structure*, el cual separa los numero negativos de los

numero positivos, los cuales dividimos entre 360, sumamos la cantidad de 89 para los datos negativos y 92 para los datos positivos, de esta manera y repitiéndolo para cada eje, obtenemos los valores que se envían. Cada eje obtendrá un rango de valores diferentes en labVIEW y y serán utilizados para ser transmitidos hacia UE4(figura 4).

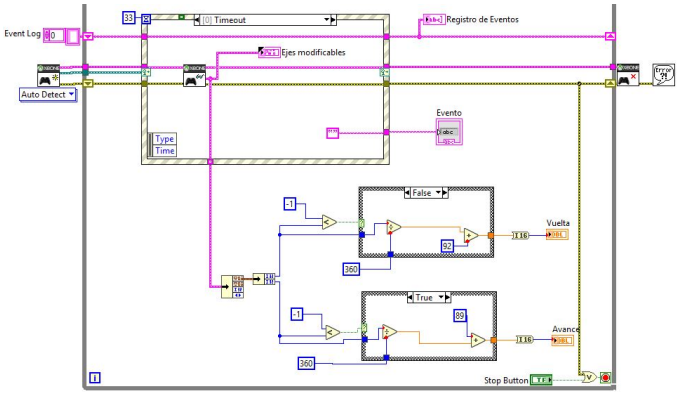

Fig. 3. Disposición de bloques de función para el inicio y fin de la conexión con el mando de Xbox one.

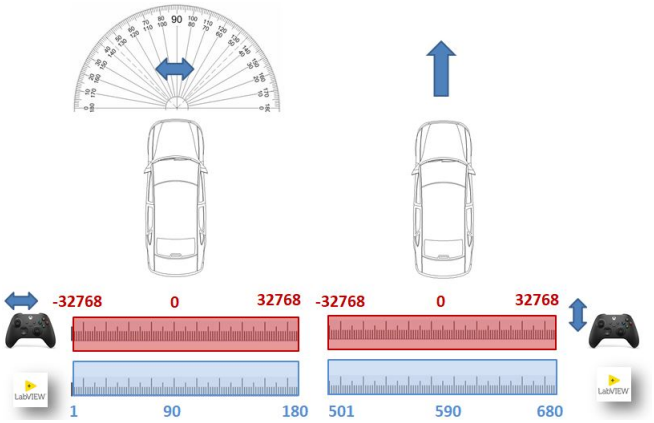

Fig. 4. El eje x de la palanca de mando izquierda brindara la dirección del vehículo, el eje y la aceleración, los datos del mando son convertidos a un rango útil y diferente cada uno, para ser transmitidos.

B. Transmisión de datos en red mediante protocolo TCP/IP

Para transmitir los datos se utiliza el protocolo TCP/IP, para lo cual desde el proyecto en labVIEW se inicia configurando fuera del ciclo while, un bloque de función *TCP Open connection*, en el cual se coloca la dirección IP (192.168.0.13) asignada por el módem de conexión al dispositivo ESP32, un numero de puerto de servicio (8888) que coincide con el que se elige de manera deliberada en el programa que se carga en el dispositivo ESP32, además un valor de 60000 ms para el timeout. Posterior a esto, se coloca un bloque de función *TCP Write* dentro del ciclo while del proyecto, y se cierra la conexión colocando fuera del ciclo el bloque de función *TCP Close connection*, en la figura 5 se puede observar esta configuración utilizada.

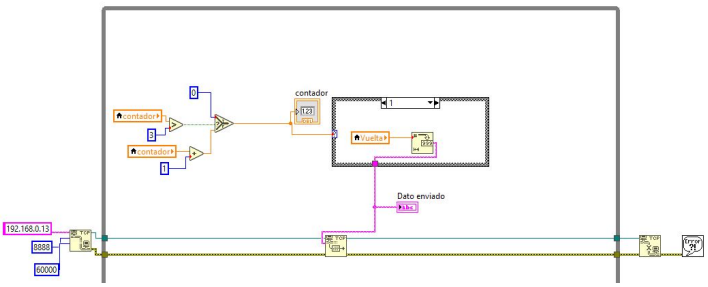

Fig. 5. Disposición de bloques de función para el inicio y fin de la transmisión de datos por protocolo TCP/IP.

La transmisión de datos se realiza cuando Labview da por finalizada una trama, para poder finalizar la trama de información que sera transmitida, es necesario enviar un carácter de espacio vació. Para poder realizar esta tarea se implemento un contador ascendente que va de 0 a 3, a su vez el valor de este contador es conectado a una estructura de control *case structure* de 4 con cuatro casos posibles, de manera que cada ciclo de programa el contador ira variando los casos a ejecutar de una manera secuencial, en los casos 0 y 2, el dato que se envía será un carácter de espacio vació que indicara la finalización de la trama, el caso 1 el dato que se enviara será el valor de la variable local *Vuelta*, en el caso 3 el dato que se envía será el valor de la variable local *Avance* sumando la cantidad de 500, logrando un rango distinto que pueda servir a identificar que datos pertenecen a cada eje. Todo lo anterior se puede apreciar en el diagrama de estados que se muestra en la figura2, la codificación de dicho diagrama corresponde a la disposición de los bloques en la parte superior de la figura 5.

C. Programación de ESP32

Para programar el dispositivo desde el IDE de arduino, es necesario, instalar la biblioteca WiFi.h y el gestor de tarjetas ESP32, una vez realizando lo anterior, procedemos a codificar el algoritmo que se describe en el diagrama de flujo de la figura 6. En la codificación del programa se comienza incluyendo la biblioteca WiFi.h con la que se trabaja, definimos las variables que usaran como también las constantes para el SSID y el Password del módem al que se conectará, de igual manera definimos un puerto de servicio de manera arbitraria que en este caso será 8888, dentro del void setup se configura la conexión serial a 9600 baudios y se inicializa la conexión WiFi. Una vez que el ESP32 se conecta al módem, se le asigna una dirección IP al dispositivo, la cual es necesaria para vincular con LabVIEW, para poder visualizar esta dirección es necesario imprimirla en el monitor serial. Posterior a la configuración inicial el void loop que se estará ejecutando continuamente realiza la tarea de búsqueda del cliente con el que se conectara, una vez

conectado consultara si existe un dato disponible a la entrada, los datos que se reciben se almacenan en una variable, en seguida el valor contenido en esa variable es transmitido por la conexión serial, se detiene la conexión con el cliente y el proceso se vuelve a repetir.

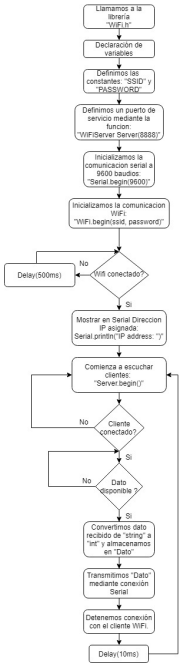

Fig. 6. Disposición de bloques de función para el inicio y fin de la transmisión de datos por protocolo TCP/IP.

D. Comunicación Serial desde UE4

Para realizar la comunicación serial se trabaja con el complemento de distribución libre llamado UE4Duino, el cual permite establecer comunicaciones seriales a través de los puertos COM. Para la instalación de este complemento, se crea una carpeta en el directorio del proyecto la cual albergue los archivos del complemento y se edita el archivo necesario.

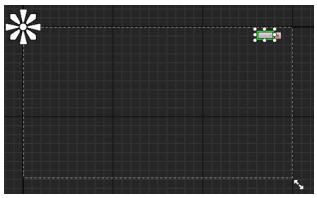

Fig. 7. Diseño del Widget Blueprint que permite al usuario seleccionar el puerto COM.

Una vez instalado el complemento, se crea un *Widget Blueprint* (figura 7) que permita seleccionar el puerto COM por el cual se establecerá la comunicación de los datos, este widget se trata de una ventana flotada sobre el escenario que permite una interfaz con el usuario desde la cual se puede realizar una selección desde un combo box desplegable. La programación de este blueprint se muestra en la figura 8, al existir un evento *On selection*, inicia una rutina para inicializar el puerto COM del numero seleccionado, por otro lado si existe un evento *On Clicked* sobre el boton con la X, la comunicación con el puerto COM debe finalizar y se cierra el juego.

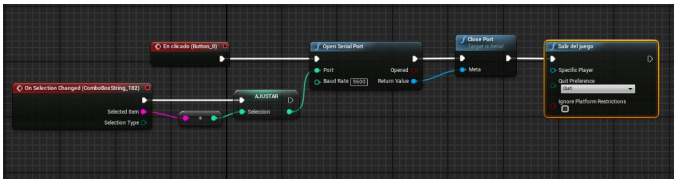

Fig. 8. Programación del Widget Blueprint que permite al usuario seleccionar el puerto COM.

Es necesario programar adecuadamente las tareas que se ejecutaran una vez que se inicia el juego y en que momento se ejecutaran dichas tareas, para esto es importante editar adecuadamente el *Blueprint del nivel* desde el cual se comienza por inicializar todas las variables en un valor de 0, posterior a esto se establece que el widget desde el cual se selecciona el numero de puerto COM se añada al primer plano de la ventana gráfica, además de especificar también que aparezca el cursor del mouse. Lo siguiente es extraer el dato entero seleccionado en el widget para poder inicializar la conexión con el puerto COM adecuado, este dato entero se almacena en una variable llamada *numero puerto*. Ajustado el valor de la variable, se procede a inicializar la conexión con el puerto serial, hacia el cual una vez iniciada la comunicación se envía el valor contenido en la variable salida, después de un breve retardo de 20 ms, se procede a leer los valores que estén siendo recibidos para comenzar a trabajar con ellos. La programación de este Blueprint se muestra en la figura 9.

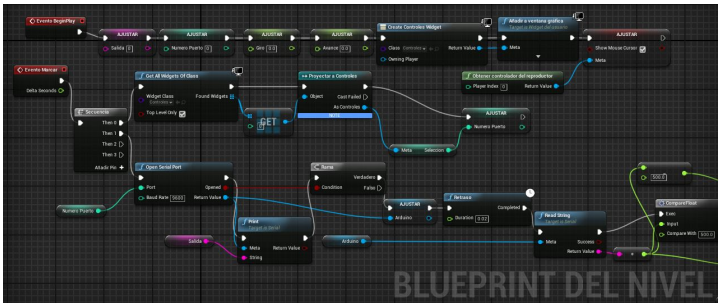

Fig. 9. Programación del Blueprint del nivel.

Una vez que es posible recibir los datos, estos deben ajustarse a la manera en que se trabajara con ellos, lo primero es convertir la cadena en valores de tipo flotante, después de que estos son convertidos se tendrá entonces valores que pertenecerán a dos diferentes rangos, el primero entre 1 y 180 , el segundo entre 501 y 680, entonces se realiza una comparación del valor para distinguir el eje de la palanca de

mando del cual proviene, en el caso del segundo rango, una vez distinguido es posible restar las 500 unidades que se le agregaron para su distinción, hecho esto, se almacenan los datos en sus variables correspondientes y se repite el ciclo. Esto se realiza de la manera que se muestra en la figura 10.

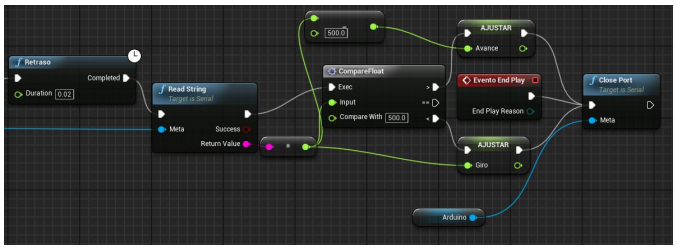

Fig. 10. Distinción de los datos de rangos distintos y su almacenamiento en las variables correspondientes a cada uno.

Los valores de las variables que ya almacenan los datos provenientes desde LabVIEW son utilizadas para dar la dirección al vehículo (figura 11) y la aceleración (figura 12), estas funciones para la entrada del volante y el avance son editadas desde el blueprint del vehículo sedan del proyecto.

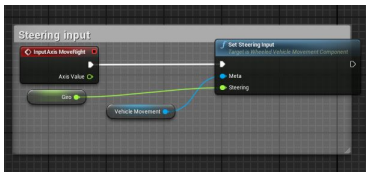

Fig. 11. Entrada para la dirección del volante desde la variable Giro.

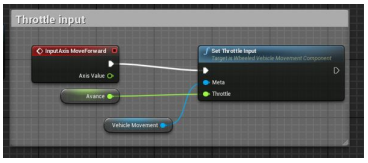

Fig. 12. Entrada para la aceleración del vehículo desde la variable Avance.

E. Diseño de escenario de conducción

El escenario de conducción es estructurado a partir del uso de dos mallas prediseñadas, diferentes texturas y un algoritmo de creación de SPline. Las mallas prediseñadas son las formas básicas para la construcción de una carretera dentro de UE4, tanto la cinta asfáltica como las barras de contención presentan texturas repetitivas que pueden ser replicadas múltiples veces. Con la ayuda de un algoritmo para la creación de un SPline es posible insertar de manera mas fácil estructuras repetitivas como puentes, tuberías y en nuestro caso una carretera, facilitando en gran media el diseño de nuestro entorno de conducción. Las texturas precargadas por el mismo UE4, nos ayudan a dar un entorno mas realista, en este trabajo se agregaron texturas de césped en el suelo y piedra en las paredes que rodean el escenario(figura 13).

Fig. 13. Perspectiva de edición en el diseño para el escenario de conducción.

III. Resultados

Se realizo la adquisición de los datos del mando de Xbox one desde el software de LabVIEW, se acondicionaron los valores de la resolución de 16 bits en rangos de 1 a 180 para el eje X, de 501 a 680 para el y, ambos ejes pertenecientes a la palanca de mando izquierda. La estrategia de utilizar diferentes rangos de números para distinguir entre diferentes ejes de una misma palanca de mando fue adecuada para el envió de la información hacia el software Unreal Engine 4, en la figura 14 es posible observar los datos obtenidos desde el palanca de mando.

Fig. 14. Resultado obtenido al colocar la palanca de mando izquierda situada al extremo derecho en el eje X y en el medio de la altura en el eje Y.

La comunicación de datos a través de protocolo TCP/IP presenta un buen desempeño en la transmisión de información entre ambos software, no se presento un retardo importante entre un movimiento en el mando y la respuesta de la animación en el motor de videojuegos. Además de lograr establecer la comunicación sin el uso de archivos dll, fue posible conducir el vehículo virtual de manera satisfactoria (figuras 15, 16).

Conclusiones

Un trabajo colaborativo entre LabVIEW y Unreal Engine 4, sin el uso de archivos dll y sin complementos específicos para la simulación virtual de la conducción de un vehículo, es completamente posible de realizar, funciona de manera adecuada y es estable, lo cual incrementa en gran medida el universo de posibilidades en animaciones 3D que pueden realizarse en desarrollos de control, automatización entre otros. Parte de los

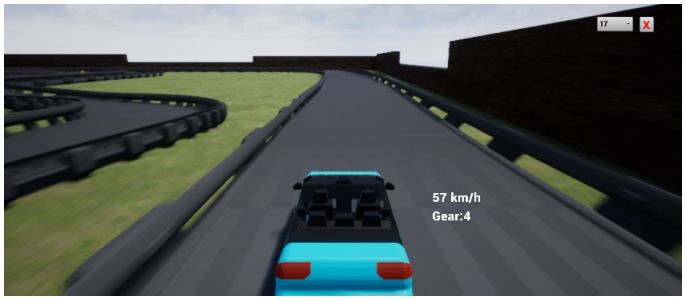

Fig. 15. Conducción de vehículo virtual a través del escenario diseñado.

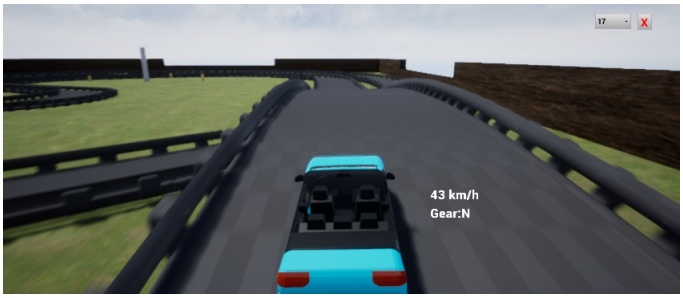

Fig. 16. Conducción de vehículo virtual sobre puente elevado dentro del escenario de conducción.

beneficios en este desarrollo son que el complemento utilizado en este trabajo y el IDE de arduino, son de distribución libre, además de que el hardware utilizado es de bajo costo, y tener la posibilidad de realizar un transmisión de datos que permita un mando a distancia con una implementación rápida lo pueden volver una posibilidad relevante.

REFERENCES

[1] H. Arioui and L. Nehaoua, Driving simulation. London: CPI Group, 2013

[2] F. H. Administration, "Roadway Human Factors and Behavioral Safety in Europe," 2015. [Online]. Available: http://repositorio.unan.edu.ni/2986/1/5624.pdf.

[3] D. L. Fisher, M. Rizzo, J. K. Caird, and J. D. Lee, Handbook of driving simulation for engineering, medicine, and psychology. CRC Press, 2011.

[4] Q. Yao, "A Compact Driving Simulator to Support Research and Training Needs - Hardware , Software , and Assessment," Clemson University, 2015.

[5] E. S. Batalla Goinzález, "Desarrollo de un Simulador 3d de Manejo Para la Capacitación de Conductores Del Sistema de Transporte Colectivo Metropolitano," Instituto Politécnico Nacional, 2012.

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," no. CoRL, pp. 1–16, 2017

[7] O. A. Perez Reyes, "Desarrollo de un Simulador de Tractocaminoes Utilizando un Ambiente Inmersivo 3D," Instituto Politecnico Nacional, 2012.

[8] K. Guldbrand and R. B. Gustafson, "Developing a low-cost driving simulator and the physical components effect on validity," Aalborg University, 2011.

[9] J. A. Hernandez, "Desarrollo de aplicaciones dinámicas dll, en lenguaje visual C++," Proyecto de fin de ingeniería, Instituto Tecnológico de Tuxtla Gutiérrez, Chis, México, 2014.

[10] R. Bitter, T. Mohiuddin, M. Nawrocki,LabVIEW advanced programming techniques, 2nd ed., vol. 1., Boca Raton, Florida: CRC Press, 2007.

[11] A. Cookson, R. DowlingSoka, C. Crumpler, Unreal Engine 4 game development in 24 hours, United States of America: Pearson Education, Inc. 2016.

[12] N. Cameron, Electronics Projects with the ESP8266 and ESP32, Building web pages, Applications and WiFi Enabled Devices, Edinburgh, UK: Apress Media LLC, 2021.

ICASAT 2021
INTERNATIONAL CONFERENCE ON APPLIED SCIENCE AND ADVANCED TECHNOLOGY

Implementation of an ADRC for DC motor angular position in an ARM Cortex M4 MCU using MicroPython

Eng. Iván Cañedo Farfán[1], Dr. Roberto Carlos Ambrosio Lázaro[1], Dr. José Fermi Guerrero Castellanos[1]

[1]**Benemérita Universidad Autónoma de Puebla;**

Abstract

This research presents the implementation of the angular position control of a DC motor using the active disturbance rejection control strategy. The implementation is performed in a ARM-M4 micro controller unit (MCU) with floating point unit (FPU). The programming of the MCU is performed using the MicroPython interpreter. The time required to compute and update the control signal is measured using a logic analyzer with the help of an auxiliary digital signal. The reference signal is generated using LabVIEW and communicated to the MCU with a serial interface. In the same way the overall performance of the system is measured in LabVIEW comparing the reference with the actual position of the motor.

Introduction

The active disturbance rejection control (ADRC) is a control strategy that has the feature of compensating the internal and external disturbances, this control strategy was first introduced in [1], since then, it has been implemented in multiple reported projects [2, 3].

The control of angular position of a brushed, permanent magnet, DC motor using the ADRC strategy is presented.

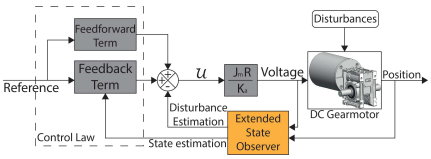

Figure 1: ADRC block diagram.

In Fig. 1, the block diagram of the ADRC is shown, where J_m is the rotor inertia, R is the motor's armature resistance and K_a is the motor-torque constant. The control law is defined by equation (1), where $K_{1,2}$ are control constants, x_d is the desired position, x_1 is the motor's position and x_2 is the motor's velocity. The extended state observer (ESO) is defined in equation (2), where $\hat{x_{1,2}}$ are the estimated states of the system, o_{0-3} are the observer gains and h is the sampling period in seconds.

$$u(t_{k+1}) = K_1[x_1(t_{k+1}) - x_d] - K_2[x_2(t_{k+1})] - z_1(t_{k+1}) \quad (1)$$

$$\begin{aligned} e(t_k) &= x_1(t_k) - \hat{x_1}(t_k) \\ \hat{x_1}(t_{k+1}) &= \hat{x_1}(t_k) + h[\hat{x_2}(t_k) + o_3 e(t_k)] \\ \hat{x_2}(t_{k+1}) &= \hat{x_2}(t_k) + h[u + \hat{z_1}(t_k) + o_2 e(t_k)] \\ \hat{z_1}(t_{k+1}) &= \hat{z_1}(t_k) + h[\hat{z_2}(t_k) + o_1 e(t_k)] \\ \hat{z_2}(t_{k+1}) &= \hat{z_2}(t_k) + h[o_0 e(t_k)] \end{aligned} \quad (2)$$

Methodology

To implement the control strategy it is necessary to measure the motor´s position and control the motor´s direction and speed. To do this, the position of the motor is measured using an AB encoder attached to the motor shaft. To control the direction and speed of the motor a H-bridge is used, where the enable pin is controlled with a PWM signal. The flowchart to implement is shown in Fig. 2.

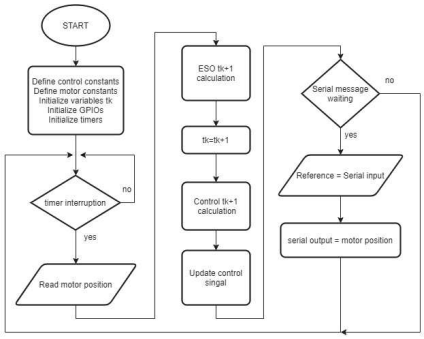

Figure 2: ADRC Flowchart

The sampling rate is defined by the timer, it must be configured to generate an interruption at least $\frac{1}{10}$ of the time that the motor need to reach its nominal velocity. Therefore, the time to calculate the control signal must smaller than $\frac{2}{3}$ of the sampling rate. The algorithm will be implemented in a ARM Cortex-M4 MCU , the schematic diagram is shown in Fig. 3.

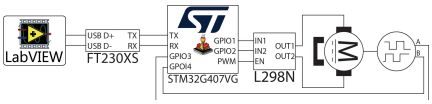

Figure 3: Schematic diagram of the implementation

Experimental Results

To measure the performance of the system, six auxiliary digital signals were used to measure the execution time of each part of the program with a logic analyzer. The time chart is shown in Fig. 4

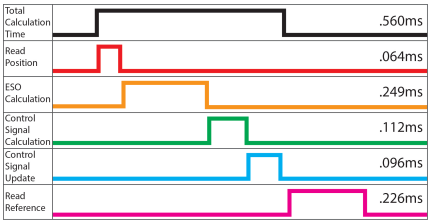

Figure 4: Time charts

The overall performance of the control strategy is shown in Fig. 5. To measure the performance the reference is plotted against the actual position of the motor using LabVIEW to communicate with the MCU by a serial port using a USB-RS232 converter.

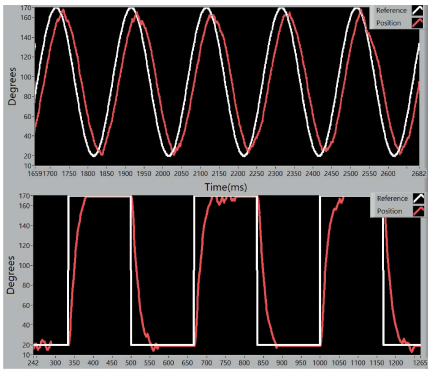

Figure 5: Overall performance of the control

Conclusion

The implemented algorithm has an overall acceptable performance. The MCU is capable of calculating and updating the control signal within the time constraints. The resources of the tested device are enough to perform the algorithm and due to the available timers, a single device can control up to three motors.

As future work, is left the codification of the algorithm in C/C++ to optimize the performance of the MCU, a considerable reduction in calculation time is expected, since the use of the MicroPython interpreter impacts in the execution time.

References

[1] H. Huang, L. Wu, J. Han, G. Feng, and Y. Lin, "A new synthesis method for unit coordinated control system in thermal power plant - ADRC control scheme," *2004 International Conference on Power System Technology, POWERCON 2004*, vol. 1, no. November, pp. 133–138, 2004.

[2] H. Chalawane, A. Essadki, T. Nasser, and M. Arbaoui, "A new robust control based on active disturbance rejection controller for speed sensorless induction motor," *Proceedings of 2017 International Conference on Electrical and Information Technologies, ICEIT 2017*, vol. 2018-Janua, pp. 1–6, 2018.

[3] J. F. Guerrero-Castellanos, A. Pulido-Flores, J. Linares-Flores, S. E. Maya-Rueda, J. U. Alvarez-Munoz, J. Escareno, and G. Mino-Aguilar, "Active Disturbance Rejection Control for Attitude Stabilization of Multi-rotors UAVs with bounded inputs," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1181–1188, IEEE, jun 2018.