Politecnico di Torino

Politecnico di Torino

Master's Degree in Mechatronics Engineering
A.a 2022/2023
October 2023

Model control and low-level interface for an autonomous car prototype

Supervisor:
Prof. Tonoli Andrea

Candidate:
Ferrari Leonardo

Abstract

In this work it is proposed the developing and implementation of Simulink models that enable an autonomous car prototype to communicate between two different control systems, Autec Remote and ROS (Robot Operating System), and all the sensors, actuators, and other components of the vehicle. All by using two distinct communication protocols: CAN (Controller Area Network) and UDP (User Datagram Protocol).
The object of this thesis is to study and analyze a previous Simulink architecture, used in 2018 in the first version of this particularly autonomous car prototype, understanding its working principle and then improving it.
Through the study of the Simulink model all its sections are explored and explained, then, by examining its behaviour and the functionality not present in the model, some improvements are developed in order to guarantee a more complete and safe functionality.
Finally the developments are uploaded in the MicroAutoBox II, that is present in the vehicle, and the autonomous car prototype is being tested to check the proper operation of all the new functionalities.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

This thesis born as an effect of the collaboration between Politecnico di Torino and Italdesign Giugiaro.

This thesis work has as its first objective the study and correct understanding of a Simulink model, developed in 2018 by a team in Italdesign Giugiaro, used for the first version of a project about an autonomous car prototype, called Tech Demo (figure 1.1). That architecture had no documentations or explanations left regarding its logic and functioning, because of that a new document containing all the information has to be developed, to allow a quicker and clearer presentation of the model for future updates.

Figure 1.1: Tech Demo

Subsequently, an analysis is carried out to understand what improvements can be made to the model to add new features to the autonomous car prototype. Finally, the new version of the Simulink model with the new features is being tested on the vehicle itself.
The thesis work is divided into the following chapters explained in the roadmap below.

1.1 Roadmap

The chapters of this thesis are organized in the following way:

- Chapter 1: **Introduction**
  In the initial chapter, a comprehensive introduction is provided pertaining to the realm of autonomous driving, with specific emphasis on Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD).

- Chapter 2: **State of the Art**
  This chapter, on the other hand, delves into the world of autonomous driving with greater specificity. It commences by scrutinizing the latest advancements on a global scale concerning autonomous vehicles, subsequently conducting an examination of the predominant sensors employed in this automotive domain. Furthermore, it provides a comprehensive overview of all the programs and libraries utilized in the course of this thesis.

- Chapter 3: **Current Simulink Architecture**
  In this chapter the Simulink architecture that was originally developed for the preceding version of this project at Italdesign Giugiaro is examined and analyzed. The objective is to gain insight into the intricate process through which the data packets received from the MicroAutoBox, along with their contained signals, are meticulously processed to ensure accurate reception by the vehicle. Due to the substantial size and complexity of the model, our analysis is structured by dividing the Simulink architecture into four overarching macro areas: *ROS Commands by UDP*, *Remote Control by CAN*, *Powertrain communication by CAN* and *ROS Feedback by UDP*.

Each area has its function and they are all interconnected each other.

- Chapter 4: **Simulink Improvements**
  Following a comprehensive examination of the structural aspects of the model employed previously, this chapter endeavors to outline enhancements that can be implemented to refine the original model.

- Chapter 5: **Testing**
  The testing chapter introduces the testing phase of this thesis work, incorporating all the enhancements integrated into the model. In the interest of safety, all modifications are initially subjected to testing while the vehicle is in a suspended state. Subsequently, once it has been verified that all components are functioning optimally, on-road testing is conducted.

- Chapter 6: **Conclusion and future improvements**
  This final chapter provides a comprehensive conclusion to the project, summarizing its key findings and offering a detailed exploration of potential areas for future improvement and development.

1.2 General overview

The automotive industry has witnessed remarkable advancements over the years, revolutionizing transportation and reshaping the way we live. From the invention of the first automobile to the integration of advanced technologies, such as electric propulsion and connectivity, the automotive landscape has undergone significant transformations. One of the most groundbreaking developments in recent times is the emergence of autonomous driving.

The first autonomous car prototype, known as "Electro," was developed in 1977 by Tsukuba Mechanical Engineering Lab in Japan. This development marked a significant milestone in the history of self-driving vehicles. The project's objective was to create a vehicle capable of autonomous navigation through the utilization of cameras and sensors, enabling it to track white street markers and attain speeds of up to 30 kilometers per hour. Over the years, numerous autonomous driving projects have emerged, contributing to the ongoing advancement of this technology.

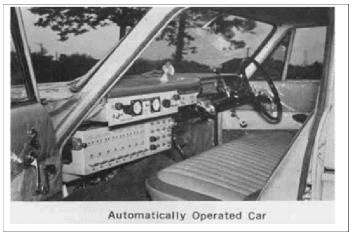

Figure 1.2: First autonomous car prototype

Autonomous vehicles, also commonly referred to as self-driving cars, possess the potential to revolutionize transportation by facilitating vehicle operation without the need for human intervention. This technology has garnered substantial attention in recent years due to its capacity to enhance road safety, improve traffic efficiency, and transform the driving experience for everyday motorists. The implementation of autonomous driving in conventional automobiles holds the promise of mitigating human errors and reducing fuel consumption, thereby enhancing transportation accessibility. By eliminating the human factor from the driving equation, autonomous driving stands to significantly diminish the number of accidents attributable to driver error. Moreover, autonomous vehicles have the potential to optimize traffic flow, leading to reduced fuel consumption and consequently a reduced environmental impact of automobiles within urban areas.

1.3 ADAS and AD

The terms ADAS (Advanced Driver Assistance Systems) and AD (Autonomous Driving) can be easily confused, but they do not refer to the same concepts. The distinction between the two can be summarized by considering the degree of human intervention that the system necessitates for driving a car.

Figure 1.3: ADAS understanding of the surrounding environment

The objective of ADAS is to expand the range of scenarios in which collisions can be prevented. With each distinct ADAS implementation, it is feasible to assess the potential reduction in incidents that can be attributed to it. As a result, all ADAS features are meticulously researched and developed to integrate various technologies aimed at eliminating circumstances in which collisions might occur. There is a multitude of ADAS functions, all of which are in a state of constant evolution, and they can be categorized based on their objectives. For instance, some ADAS functions are designed to prevent forward collisions, such as Automatic Emergency Braking, Pedestrian/Child Pedestrian/Bicycle Detection, and Obstacle Detection. Others are geared towards controlling the vehicle's speed, as seen in Adaptive Cruise Control, or managing the steering wheel angle, exemplified by Lane Keeping Assist or Lane Centering. Additionally, ADAS encompasses features that, while not directly focused on accident avoidance, provide valuable assistance to the driver in their daily routines, such as Intelligent Parking Assistance and Automotive Night Vision.
As for AD (Autonomous Driving), it entails a scenario in which ei-

ther the machine or the vehicle itself assumes responsibility for specific tasks or the entirety of driving functions, effectively replacing the human driver.

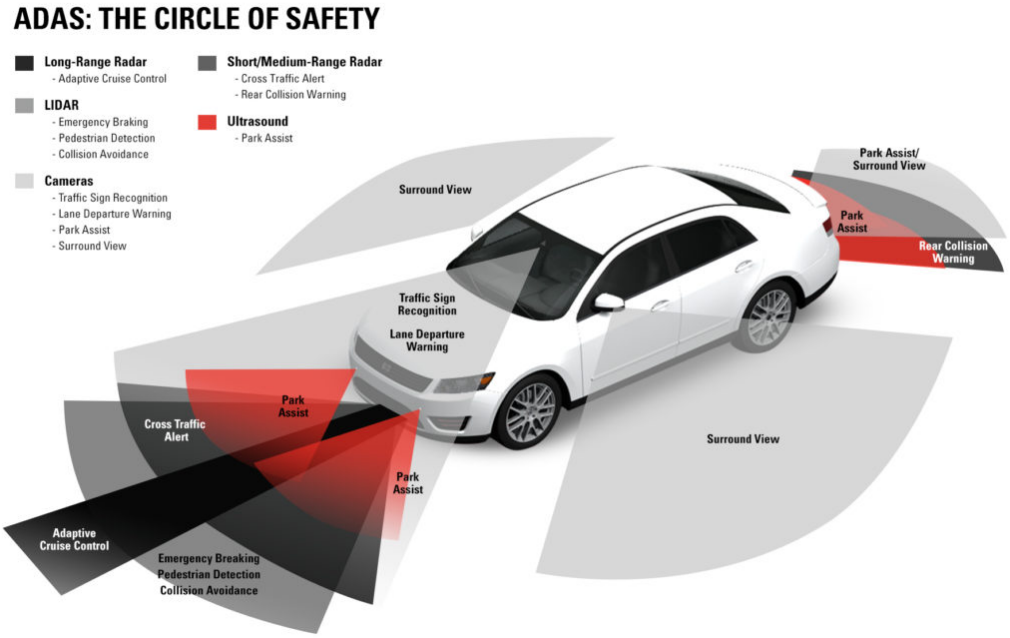

Figure 1.4: ADAS features and correspondent sensors used to develop them

1.3.1 Levels of driving automation

An AD (Autonomous Driving) system is anticipated to execute the entire spectrum of actions essential for the driving process. To achieve optimal performance, it relies on the perception of the surrounding environment through sensors, as depicted in Figure 1.4. The processing of data gathered by these sensors culminates in control decision-making. To classify the level of autonomy attained by an AD system, the Society of Automotive Engineers (SAE) has established a scale comprising six levels, ranging from "no automation" to "full automation". They are resumed in the figure 1.5 in the following page.

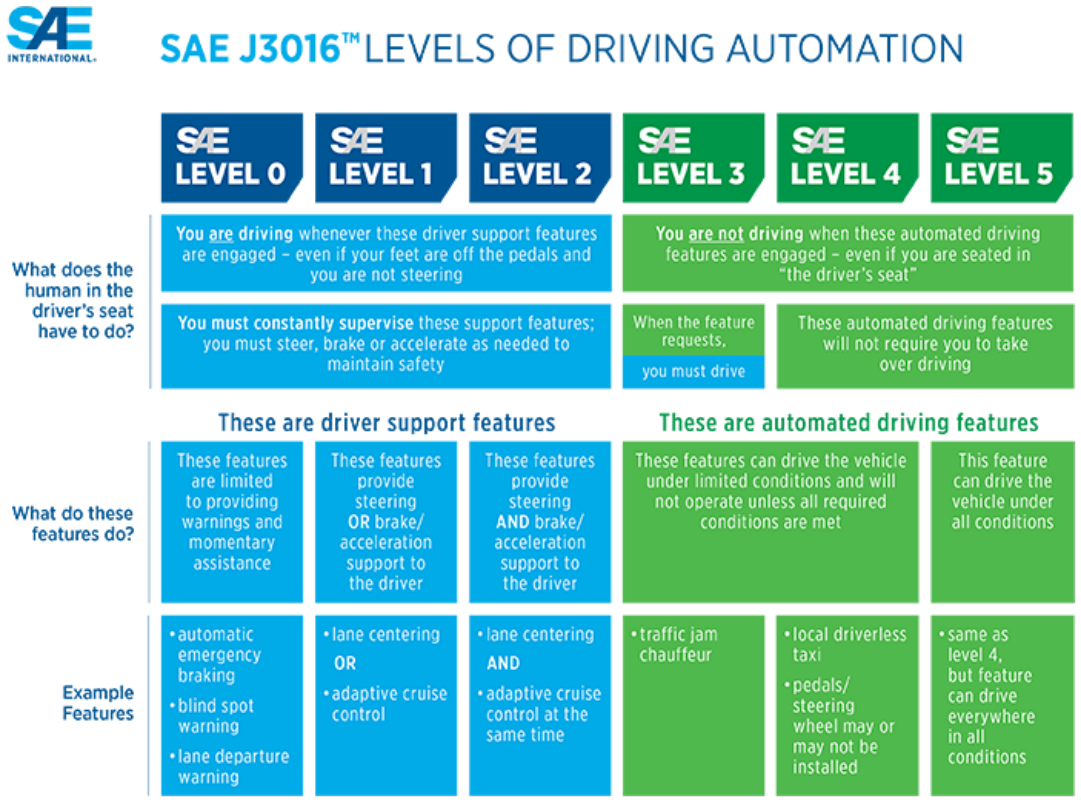

Figure 1.5: SAE International's Levels of Driving Automation for On-Road Vehicle

- `Level 0`: vehicles completely lack any driving automation technology, it may has automated system issues warnings or alerts, so it has no sustained vehicle control.

- `Level 1 ("hands on")`: the automotive system provide assistance with acceleration, steering and brake, the driver and the automated system share the control of the vehicle (the driver always has the decision-making power). The driver must be ready to retake full control at any time. Example of level 1: Adaptive Cruise Control, Parking Assistance, Lane Keeping Assistance...

- `Level 2 ("hands off")`: the automotive system takes full control of acceleration, steering and brake. The driver must be ready to intervene and should no literally takes his hands off (contact between hands and the wheel is mandatory).

- `Level 3 ("eyes off")`: vehicles autonomously handle all driving tasks and the driver can safely turn his eyes away from the road.

The vehicle his able to handle situation that require immediate response as Emergency Brake, but the driver must always be available to take the wheel if required.

- `Level 4 ("mind of")`: no driver attention is required for safety, autonomous vehicle systems are completely responsible for all driving tasks. Level 4 is limited only in certain spatial areas or under specified circumstances, outside these areas or conditions it must be able to safely abort the trip.
- `Level 5 ("passenger")`: no human intervention is required at all, the driving automation systems will operate independently and universally in all weather conditions and roadways.

As of today, none of the automotive companies can offer a fully autonomous system.

Chapter 2

State of the Art

In this chapter, various elements pertaining to this project are comprehensively described. To begin with, self-autonomous cars and the associated concepts are briefly elucidated. Following that, Autoware, recognized as the world's foremost open-source software project for autonomous driving, is discussed in detail. Subsequently, the software environment employed for the execution of this project, which comprises Matlab/Simulink and dSPACE, is expounded upon, with a specific emphasis on the MicroAutoBox II.

2.1 Autonomous Driving

A self-driving car is a vehicle capable of operating without human intervention. The concept of creating a system capable of autonomously driving a car was already envisioned in the previous century. However, during that era, the available technology did not possess the capability to tackle such a complex task. In recent years, significant advancements in the field of computer technology have fundamentally altered the perspective on this technology, making the development of self-driving cars a viable possibility.

Figure 2.1: Waymo driverless taxi

At the end of the last century, some researchers ventured into the realisation of the first autonomous driving architectures developing and testing a few prototypes that were able to drive on real streets. These tests were performed in delimited and protected areas, and the target is to increase these areas. An example is the Waymo One taxi service, a driverless taxi that has been tested in San Francisco in 2021 and it will be tested in the recent future in the street of Los Angeles.
These experiments demonstrate that the final solution is still far away because for example the Waymo experiments is still not ready for an intense traffic, they demonstrated that autonomous driving is getting better year to year and it is not anymore just a dream in the mind of futurists.
Before getting a complete self-driving car able to drive in every street it will still take years but leaps and bounds are made every years. Most of the pioneers in the autonomous driving industry started testing driverless car systems as of 2013, including General Motors, Ford, Mercedes Benz, Volkswagen, Audi, Nissan, Toyota, BMW, and Volvo.
BMW has been testing driverless systems since around 2005, while in 2010, Audi sent a driverless Audi TTS to the top of Pike's Peak at close to race speeds. In 2011, GM created the EN-V (short for Electric Networked Vehicle), an autonomous electric urban vehicle. In 2012, Volkswagen began testing a "Temporary Auto Pilot" (TAP) system that will allow a car to drive itself at speeds of up to 80 miles per hour ($130km/h$) on the highway. Ford has conducted extensive research into driverless systems and vehicular communication systems. In January 2013, Toyota demonstrated a partially self-driving car with numerous sensors and communication systems.
Tesla motors released the first version of its Tesla Autopilots in 2014 on board of Model S. Even if younger than the other companies operating in automotive, Tesla is leading the today market of self-driving cars, and is working to demonstrate a self-driving coast to coast drive from Los Angeles to New York. Waymo started as the Google self-driving car project in 2009. Today it is an independent company which produces one of the most advanced self-driving systems existing.

In addition to the big actors of the automotive industry, more and more startups have ventured into the field of autonomous driving relying on new technologies and highly qualified staff. Through the most successful and established ones we have Zoox, Roadstar.ai, Pony.ai, Aurora, TuSimple, Drive.ai.

Having a look at the figure 2.2 get by google trends it is possible to see an increasing interest in autonomous driving from 2012/2013. This trend is likely to be related with the rapid progress obtained in the computer industry, sensor industry and new paradigms of data analysis like machine learning and deep learning.

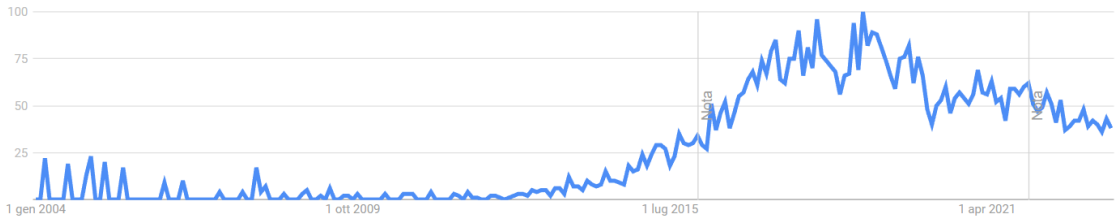

Figure 2.2: Google trends on interest on autonomous driving over time

2.2 Sensors

Here, a concise overview is provided of the sensors commonly utilized aboard autonomous cars. The system within self-driving cars primarily relies on three functional blocks:

- Perception: the process that provides a computer image, divides the image in objects of interest then identifies such objects and finally gives a meaning to the recognized objects, so it represent the environment.
- Decision Making: it analyzes the environment given by perception block and decides which actions to take.
- Actuation: the block that actually put the actions in place.

All the sensors constitute the perception block, and their primary function is to provide a highly detailed description of the surrounding environment. This level of detail enables the decision-making block to select the most optimal course of action.
The principal sensors employed in autonomous cars are detailed in the following subsections:

2.2.1 Camera

Camera sensors are among the most prevalent sensors found in all models of autonomous cars, and they serve as the foundation for implementing nearly all ADAS features. Depending on their placement within the vehicle, cameras are employed for monitoring nearby vehicles, lane markings, speed limit signs, high-beam control, and the presence of obstacles along the vehicle's path. The most commonly utilized locations for installing cameras in cars include the front grille, side mirrors, rear doors, and rear windshield.

This is how a camera for autonomous driving works:

1. Image capture: is the process that provide a computer image, using an array of pixels to capture incoming light converting it into an electronic signal. Each pixel correspond to a specific point and the more pixels a camera has, the higher is the resolution of the image.

2. Pre-processing: deals with noise reduction and detail improvement.
3. Segmentation: divide the image in object of interest.
4. Description: compute characteristic, as dimension and shapes, that are useful to differentiate one object from another.
5. Recognition: the processed image is analyzed by computer vision algorithms to identify objects, is the process that identifies objects such as pedestrians, vehicles and all the relevant elements in the road environment.

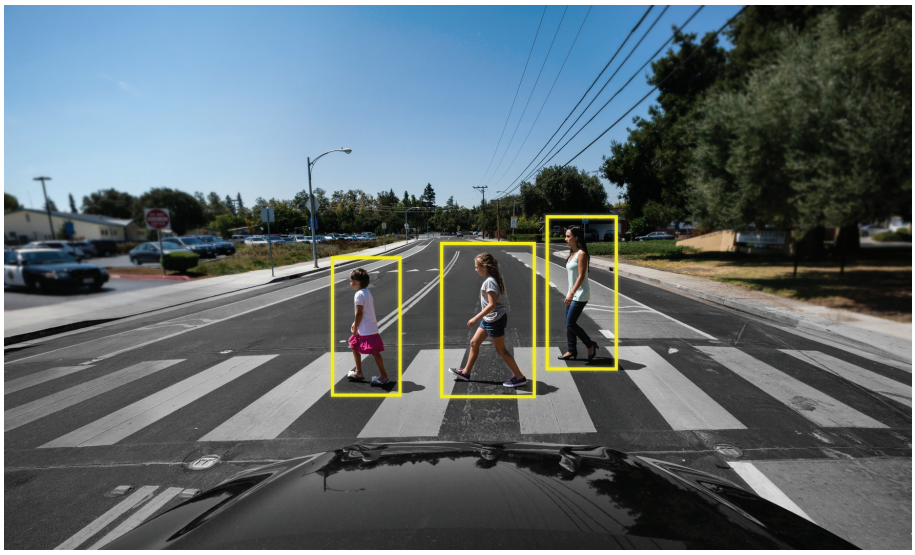

Figure 2.3: Camera able to recognize pedestrians

The primary limitation of cameras is their ability to capture details in the surrounding environment, but they require distance calculation to precisely determine the location of objects. This becomes particularly challenging in adverse weather conditions such as fog and rain.
As a result, relying solely on cameras is insufficient to create a comprehensive map of the surroundings and accurately determine the vehicle's position within that map. Instead, cameras need to complement their data with information from multiple sensors like radar and lidar to overcome these limitations.

2.2.2 Radar

Radar, which stands for "Radio Detection and Ranging," utilizes radio waves to detect objects and ascertain their range, angle, and/or velocity. The fundamental operating principle of radar involves emitting radio waves and subsequently measuring the time it takes for these waves to bounce back after striking an object. Radars are instrumental in detecting objects and precisely determining their distance, direction, and speed.

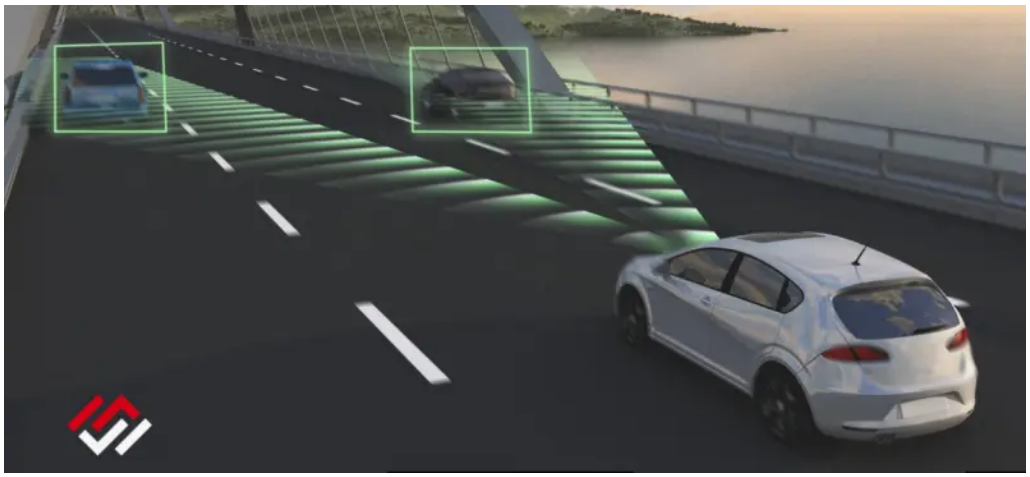

Figure 2.4: Radar sensing technology

One of radar's most significant advantages is its capability to operate effectively in challenging environmental conditions, including nighttime driving and foggy conditions. Radars can complement cameras by providing reliable object detection and distance measurement, even when visibility is reduced. This combination of radar and cameras proves invaluable in situations where cameras alone may struggle to provide a clear and comprehensive perception of the environment.

Radar in self-driving vehicle operates at 24, 74, 77 and 79 GHz, that correspond to different type of radar, short-range radars, medium-range radars and long-range radars.
The short-range radars are used for detecting the environment around the car at low speed, so for example for lane keeping or blind-spot monitor. While the long-range radars cover long distances ($\simeq 200m$) and they are mostly used for distance control and brake assistance.

2.2.3 Lidar

Lidars (Laser Imaging Detection and Ranging) have the same working principle of radars, but with pulsed laser light rather than radio waves. They do not simply detect the objects as radars do, but lidars also describe them by describe all the environment around the objects, this operation repeated million times per second gives a 3D view of their environment, providing shape and depth to surrounding cars, pedestrians and road geography. Like for radars, lidars, using laser light, they work well at night but not in adverse weather conditions such as heavy rain, fog, or snow, the performance of lidar sensor can degrade under such conditions, reducing its effectiveness in certain driving scenarios.

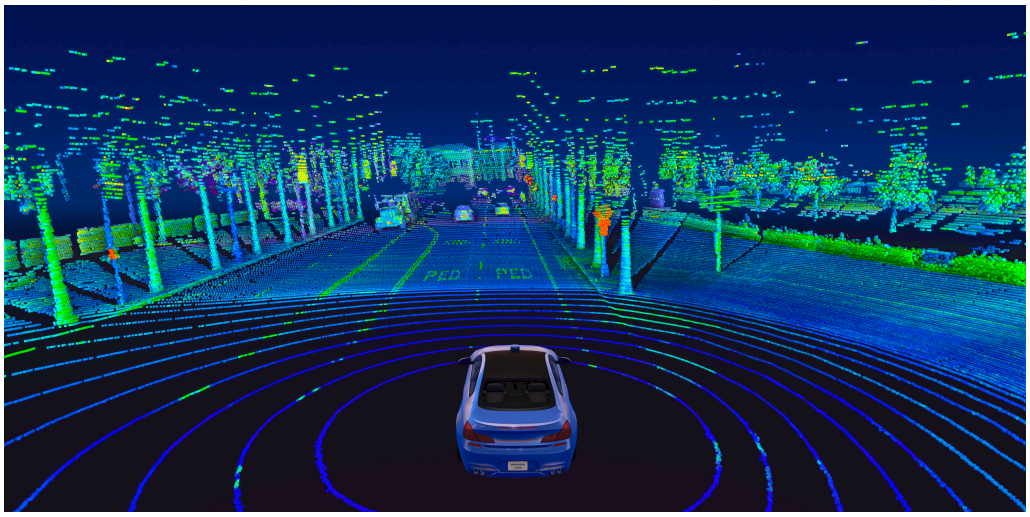

Figure 2.5: Point cloud created by a velodyne lidar

The lidar in a self-driving vehicle has to be placed on the top of the vehicle without obstruction, to guarantee a correct 360° view. The position makes it easy to be damaged and also unaesthetic.
The main drawback of lidars is the cost, high-quality LiDAR sensors can be quite expensive, making them a significant factor contributing to the overall cost of autonomous vehicles. The cost has been decreasing over time, but it still remains a barrier for widespread adoption.

2.3 Autoware Universe

Autoware Universe is an open-source software platform specifically designed for autonomous driving systems. The primary objective of this software is to expedite the development of self-driving technology by offering a versatile and readily accessible platform. It serves as a valuable resource for researchers, developers, and companies engaged in the field of autonomous driving, facilitating advancements in this domain.
The key feature of this software are:

- Community and open source software: it contains a community of developers and researchers that collaborate and contribute to the advancement of the autonomous driving research. The most important resource of any software is a community that shares its knowledge helping each other.
- Modular Architecture: adopting a modular design it allows developers to choose and integrate the components and algorithms that best suit the applications they need. This provide flexibility, reusability and extensibility to the software.
- Sensor integration: to provide the capability of integrate various sensors such as lidar, radar, cameras, and GPS.
- Perception and Mapping: these task give the possibility to analyze all the data acquired by the sensors and understand the objects in the environment generating a detailed map.
- Planning and control: to generate optimal trajectories and control commands for the autonomous vehicle.

2.4 MatLab and Simulink

MATLAB is a widely utilized programming and numerical computing platform employed by countless engineers and scientists. It is employed for tasks such as data analysis, algorithm development, and model building.
Simulink, on the other hand, is a block diagram environment that finds application in designing systems with multi-domain models. It enables the running of simulations before transitioning to hardware implementation and allows for deployment without the necessity of manual code writing. The majority of the work conducted for this thesis has been accomplished using Simulink.

2.5 dSPACE

dSPACE is a prominent global technology leader specializing in simulation and validation solutions. The company serves as a partner throughout the entire innovation process, from the inception of ideas to supporting series production. dSPACE's expertise spans various domains, including the automotive sector, On and Offroad Commercial Vehicles, Aerospace, and the Energy Industry, where it develops solutions to advance and enhance technology across these industries.

2.5.1 MicroAutoBox II

Figure 2.6: MicroAutoBox II

The MicroAutoBox II is a real-time system designed for rapid function prototyping with high-speed capabilities. It operates autonomously, much like an Electronic Control Unit (ECU).

This versatile platform finds application in a wide range of rapid control prototyping scenarios, including powertrain, chassis control, body control, electric drive control, aerospace applications, advanced driver assistance systems (ADAS), and more. In the context of this project, the MicroAutoBox serves ADAS purposes.

Specifically, one MicroAutoBox II unit is integrated into this autonomous car prototype, serving as the system where the Simulink model (in the form of an sdf file) developed within this thesis is uploaded and executed.

2.5.2 Real-Time Interface

The Real-Time Interface (RTI) is a software implementation that facilitates the execution of models on dSPACE hardware. It acts as the bridge between dSPACE hardware and the development software MATLAB/Simulink provided by MathWorks.

RTI offers specialized blocks that incorporate the input and output capabilities of the dSPACE system directly into the Simulink model. This preparation of the model is essential for real-time applications.

Furthermore, RTI includes consistency checks to identify and address any errors or issues during the build process.

Simulink Coder, on the other hand, serves as a C and C++ code generator. It translates Simulink models into executable code. The generated code can be applied to both real-time and non-real-time applications, depending on the specific requirements of the project.

2.5.3 RTI CAN MultiMessage Blockset

This extension, an integral part of the Real-Time Interface, serves the purpose of seamlessly integrating dSPACE systems with CAN communication networks and configuring CAN networks. It is fully compatible with Simulink and supports various dSPACE products, including the MicroAutoBox.
This extension allows for the direct transfer of messages between two CAN buses, either in a unidirectional or bidirectional manner, through the Gateway block, without the need for message or signal manipulation. With this block-set, it becomes feasible to manage, configure, and manipulate a large number of CAN messages from within a single Simulink environment.

In the context of this project, it facilitates communication between the MicroAutoBox II on the Tech Demo and the remote controller utilized for vehicle control.

2.5.4 RTI Ethernet UDP

This extension, an extension of the Real-Time Interface, offers a fundamental UDP (User Datagram Protocol) communication interface within a Simulink model. The RTI Ethernet UDP Blockset enables the exchange of data between a Simulink model operating on the host computer and a real-time application executing on dSPACE hardware, all in real-time.

In the context of this project, this extension facilitates communication between the MicroAutoBox II on the Tech Demo and an external computer.

2.5.5 ControlDesk

ControlDesk is a software tool developed by dSPACE, widely utilized in the automotive and aerospace industries for purposes such as rapid prototyping, testing, and validating electronic control units (ECUs) and other embedded systems. ControlDesk provides a user-friendly graphical interface that facilitates interaction with control systems throughout the development and testing stages. It enables users to monitor, calibrate, and control a wide range of parameters and functions in real-time, contributing to efficient and effective development processes.

Figure 2.7: Screenshot of the graphical interface of ControlDesk

2.5.6 RTMaps Interface Blockset

The dSPACE Blockset for Simulink provides a means for bidirectional and low-latency UDP/IP communication between RTMaps and dSPACE platforms. This blockset allows Simulink applications to connect to the relevant communication blocks through signal buses, enabling data connections to be established with multisensor applications in RTMaps. Moreover, this blockset simplifies the development and testing of perception and application algorithms for Advanced Driver Assistance Systems (ADAS) and automated driving. It serves as a crucial tool in bridging communication between Simulink and RTMaps for efficient algorithm development and testing in the context of autonomous driving technology.

2.5.7 Platform API Package

The dSPACE TargetLink API is a library that enables the downloading and starting/stopping of models. It also provides high-level access to model variables for various operations such as reading, writing, stimulating, capturing, and more.
An API, which stands for Application Programming Interface, defines a set of rules and protocols that enable different software applications to communicate and interact with one another. In the context of autonomous cars, an API is used to facilitate communication with the system and integrate data from various sensors such as cameras, radars, and lidar.
A Platform Package, in this context, refers to a collection of tools and libraries designed to streamline development by offering standardized interfaces and simplifying integration between different subsystems. It aids in creating a cohesive and well-integrated software and hardware ecosystem for autonomous vehicle development.

Chapter 3

Current Simulink Architecture

In this chapter, a detailed description and explanation of the Simulink architecture that was developed for the previous version of the vehicle is provided.
The following model, figure 3.1, was present in the MicroAutoBox II without any explanation on the functionality of the specific blocks and why this type of architecture and layout was used, for this reason one of the objectives of this thesis work is the analysis and understanding of this previously developed model.
Particular focus will be given when analyzing the block regarding dSPACE libraries, that are different from the common Simulink blocks and allow the correct functioning of the communication between the Autec Remote or ROS and the MicroAutoBox II.

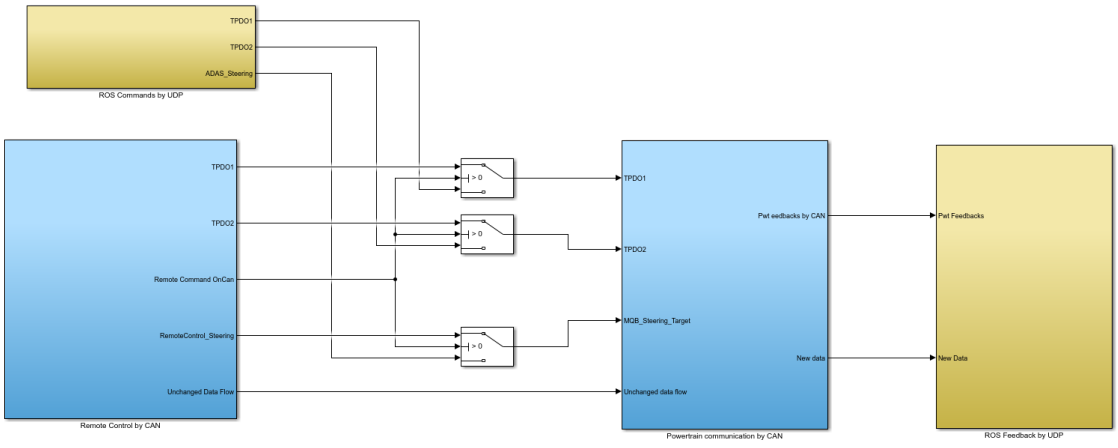

Figure 3.1: General overview of the current simulink structure

The simulink model is composed by four macro areas, each of them has its own specific functions and blocks:

1. ROS Commands by UDP: This area is responsible for receiving UDP packets sent by ROS. Each UDP packet contains multiple pieces of information that need to be extracted and processed individually or in groups. The output of this macro block provides signals structured to match the communication format used via Remote.

2. Remote Control by CAN; This area has the simplest structure among all, mainly because the signal received via CAN from the Autec Remote requires minimal modifications. It possesses a straightforward and clear structure, with all signals internally processed by the remote control.

3. Powertrain communication by CAN; Following the two areas that receive signals via CAN and UDP, this section's role is to consolidate and organize all received signals. Its purpose is to assemble the complete signal to transmit instructions to the vehicle comprehensively. Subsequently, it receives a signal from the vehicle containing all the data regarding its status.

4. ROS Feedback by UDP; This final area is responsible for aggregating all the data received from the vehicle. Its purpose is to package this information and send it via UDP to ROS, enabling the computer to monitor the vehicle's status effectively.

3.1 ROS Commands by UDP

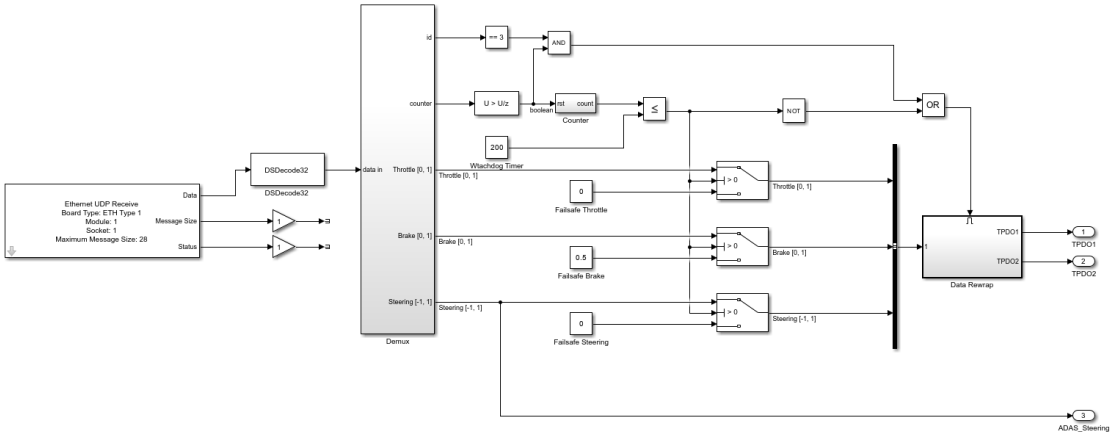

Figure 3.2: ROS Commands by UDP

This macro block encompasses the structure responsible for processing the data received from the computer via UDP. To give a brief description of his working principle, on the left *Ethernet UDP Receive* receives the UDP packets from the PC then they are identified and divided by a *DSDencode* and a *Demux*. This signals are divided into two group:

- `ID` and `Counter` give information about the packet received and the are analyzed to check if the information received are in the correct order, if not "fail safe" procedure occur or the block is disabled.
- `Throttle`, `Brake` and `Steering` are the commands given by the PC and are expanded in *Data Rewrap* to get the two signal *TPDO1* and *TPDO2* with the correct structure.

In the following pages is beign made a more deep analysis on all the block used in this subsystem, starting from the left:

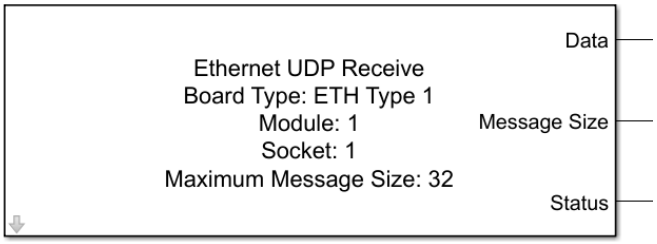

Figure 3.3: Ethernet UDP Receive

This specific block is part of the packet dSPACE RTI Ethernet UDP and his function is to receive and read an UDP signal. The outputs of this block are three: *Data*, *Message Size* and *Status*. For the purposes of this work only the output *Data* is relevant.
The signal *Data* contains all the information given by ROS and to select and divide all the single data a `DSDencode32` is used (visualized in the figure 3.5).

Figure 3.4: DSDecode

DSDencode32 converts a 32-bit word input data into the specified datatype on the output port. The *data in* is then extracted using a demux block to separate the different signals. The representation of the packet sent is visualized on the following page (Figure 3.5).

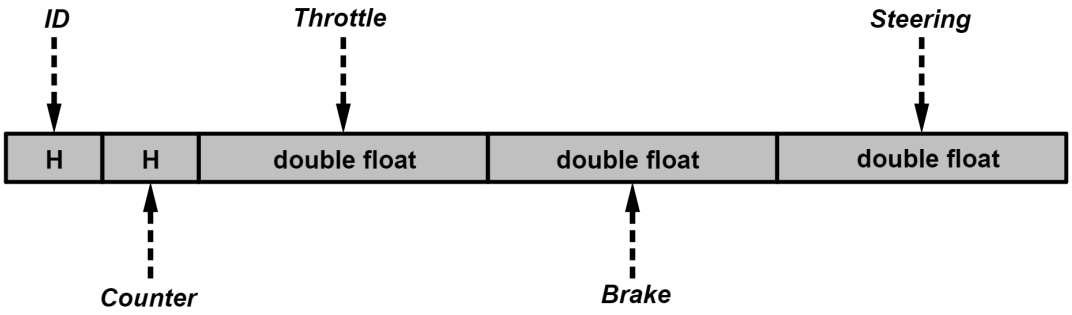

Figure 3.5: Array of received data

▷ **ID** (Unsigned short int (H) - 2B): that's the unique identifier for the UDP packet and it must be equal to 3 in order to work.

▷ **Counter** (Unsigned short int (H) - 2B): it's the value related to the packet counter, indicating when the packet was created by the MicroAutoBox II. It must increase over time to prevent it from being discarded. Consequently, it must be checked every time to ensure that the counter of the current packet is greater than that of the previous one.

▷ **Throttle** (Double float (d) - 8B): represents the value of the throttle, which should be between 0 and 1. It sets the RPM to the chosen value.

▷ **Brake** (Double float (d) - 8B): represents the value of the brake that should be between 0 and 1. It sets the braking pressure to the chosen value.

▷ **Steering** (Double float (d) - 8B): represents the value of the steering that should be between -1 and 1. It sets the steering angle to the chosen value.

3.1.1 Counter

The counter is positioned right after the *detect increase* for the counter data, his function is to increase from time to time if the data received as input is false (so if the counter is not increasing).

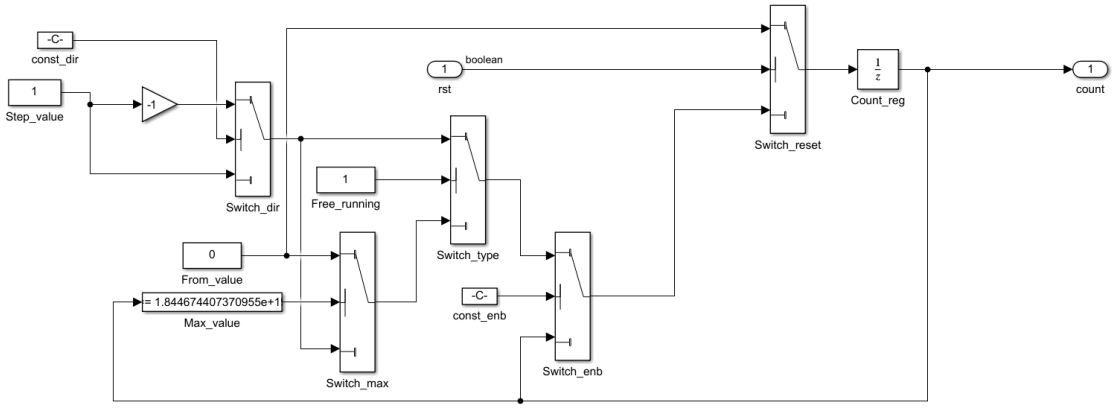

Figure 3.6: Counter

The value entering the counter is a boolean variable:

- If $rst = 1$, so the counter is increased, the output of the switch_reset is 0.
- If $rst = 0$, so the counter is not increased, the output of the switch_reset depend on different factors, if the two constant *const enb* and *const dir* are both equal to 1 the output is also equal to 1.

3.1.2 Watchdog timer

A watchdog timer is a hardware or software mechanism created to oversee the operation of a computer system or device. Its main objective is to identify and recover from malfunctions, crashes, or unforeseen events that may arise during normal system operation. The watchdog timer functions by periodically monitoring the system to ensure it is functioning as intended. It necessitates a regular reset to prevent it from initiating a system reset or executing a predefined action. If the watchdog timer isn't reset within a specified time frame, it assumes that the system has become unresponsive or malfunctioned and proceeds to take corrective measures.

In this system, the watchdog timer is activated when the counter's value reaches a particular threshold, indicating that the counter data hasn't increased for an extended period. However, it resets each time the received counter data is higher than the previous value.

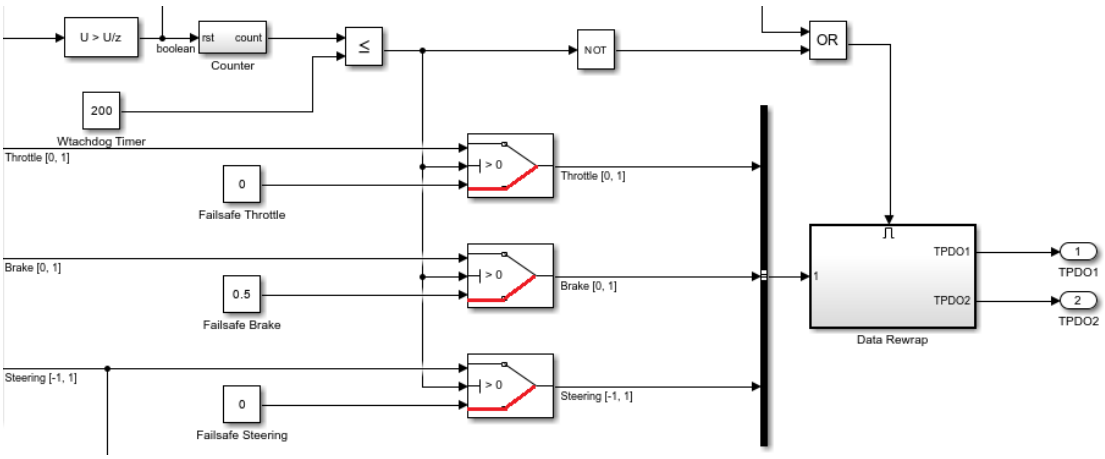

Figure 3.7: Failsafe procedure

The triggering of the watchdog timer will activate a **failsafe procedure**, this procedure activates the switches that influence the value of throttle, brake and steering, giving new values: *Failsafe Throttle, Failsafe Brake* and *Failsafe Steering*. The failsafe values have the purpose to stop the vehicle and because of that throttle and steering are equal to zero while brake is equal to 0.5.

The logic function of the watchdog timer is the following:

- If the detect increase stays false for more than x time the counter, that it is increasing time to time, reaches the watchdog timer and when it passes it the failsafe procedure is activated.
- If the packages sent via UDP has problems with the counter (it stays the same or decreases for long period of time) the system detects that there is a problem and stop the vehicle.

3.1.3 Data Rewrap

Data rewrap is an Enabled Subsystem that processes the three values of Brake, Throttle and Steering and the two outputs of the subsystem are TPDO1 and TPDO2.

Enable subsystem means that an *Enable Port* is present, its function is to check the previous signal and when the conditions are respected and the signal entering the enable port is True (1) it enables this subsystem, while if it receives False (0) the subsystem is not enabled.

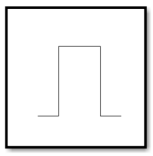

Figure 3.8: Enable port

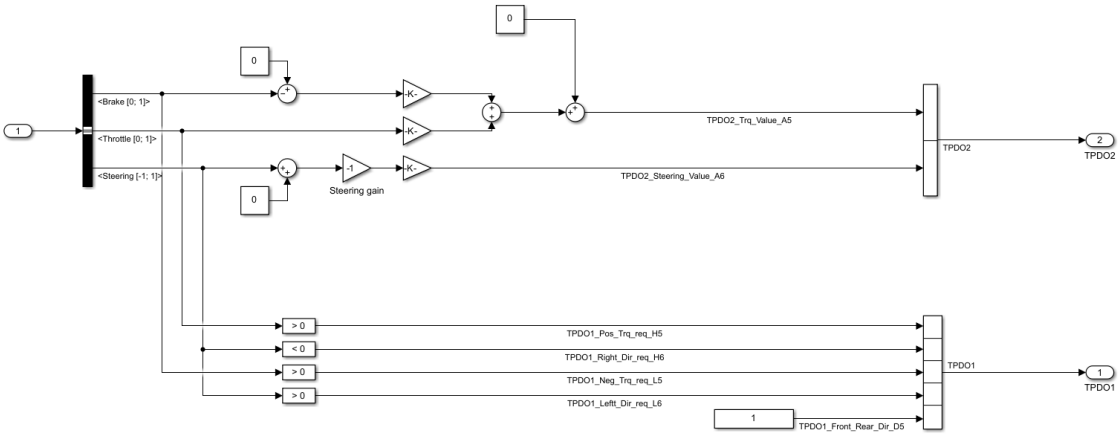

Figure 3.9: Data Rewrap

TPDO1 signal contains all the information about torque and direction:

- $TPDO1\_Pos\_Trq\_req\_H5$, if it is required a positive torque is equal to 1 and if not 0;
- $TPDO1\_Right\_Dir\_req\_H6$, if it is required a right turn (steering<0) is equal to 1 and if not 0;
- $TPDO1\_Neg\_Trq\_req\_L5$, if it is required a negative torque (brake) is equal to 1 if not 0;
- $TPDO1\_Neg\_Trq\_req\_L6$, if it is required a left turn (steering>0) is equal to 1 and if not 0;
- $TPDO1\_Front\_Rear\_Dir\_D6$, it gives the front or rear direction (always equal to 1 so only allowed front direction).

TPDO2 is a contiguous output and it is the concatenation of two data, the first one regarding throttle and brake and the second one regarding the steering.

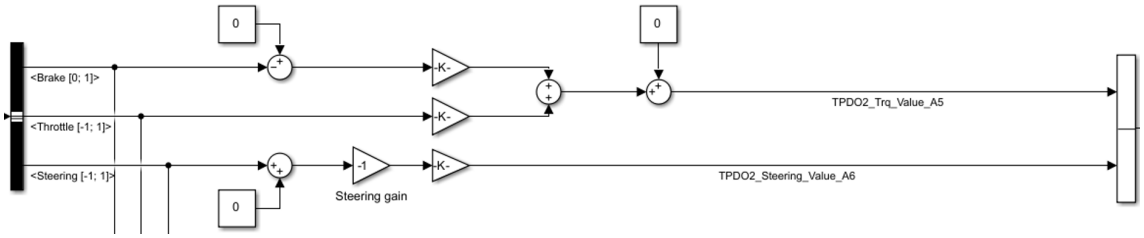

Figure 3.10: TPDO2 signal

The value of brake is reversed and then, after multiplied by a constant K (equal to 100) to get a value between 0 and 100, summed with the value of throttle to get as result the torque value that is requested.
It happens the same for the steering that is reversed and multiplied by K (equal to 65, that is the steering value), to get the steering value.

The gain K used for brake and throttle is different from the one used for steering. For Brake and Throttle we need to obtain a value from -100 to 100, with -100 maximum brake, 0 no brake no throttle and 100 maximum throttle. Otherwise for Steering the value required is between the value of -65 to 65 that is the maximum angle of steering of both directions.

3.2 Remote Control by CAN

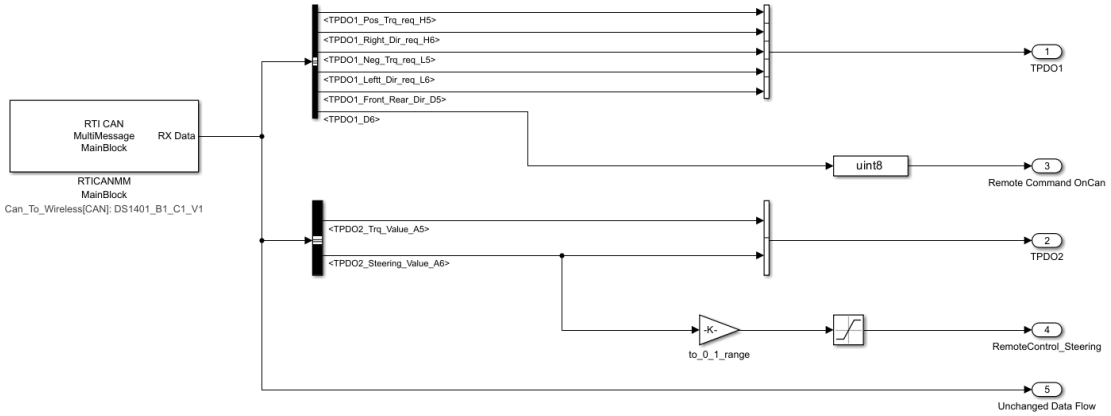

Figure 3.11: Remote control by CAN subsystem

This subsystem receives the data from the remote control via CAN and gives multiple outputs:

- **TPDO1** contains the information about positive/negative torque and right/left direction.
- **Remote Command OnCam** is the switch between the remote control and ROS command.
- **TPDO2** contains the value of torque and steering.
- **RemoteControl_Steering** is the value of steering and it has a gain K ($= -0.01$) to invert it and scale it in a range between 0 and 1. It presents also a saturation block that limits input signal to the upper and lower saturation values (1 and -1).
- **Unchanged Data Flow** all the data received by the remote controller is used again in other subsystems.

3.3 Powertrain communication by CAN

Before entering the subsystems TPDO1 and TPDO2 in both the "Remote Control by CAN" and "ROS Commands by UDP" sections, the signals must pass through switches. These switches are responsible for determining which of the two signals will proceed. This is necessary because the subsystems cannot receive both UDP and CAN packets simultaneously; only one of them can be considered at a time. The decision is made by a third signal, *Remote Command OnCam*, which, based on its value, changes the orientation of the switches.

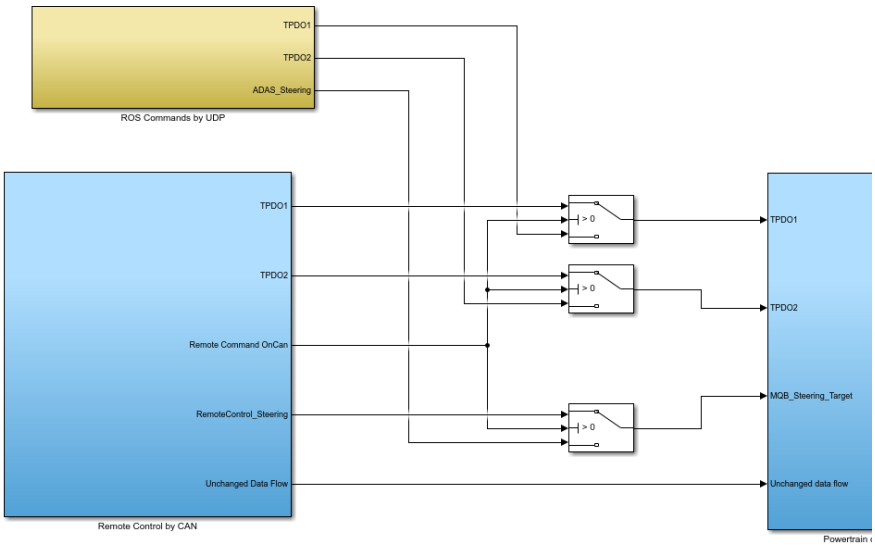

Figure 3.12: Conjunction between two subsystems

Only the "Remote Control by CAN" has the authority to determine which signal to select, while "ROS Commands by UDP" lacks decision-making capabilities. As a result, ROS Commands cannot be considered until the *Remote Commands OnCam* signal provides the opportunity for the signal to pass through the switches.

3.3.1 Powertrain communication

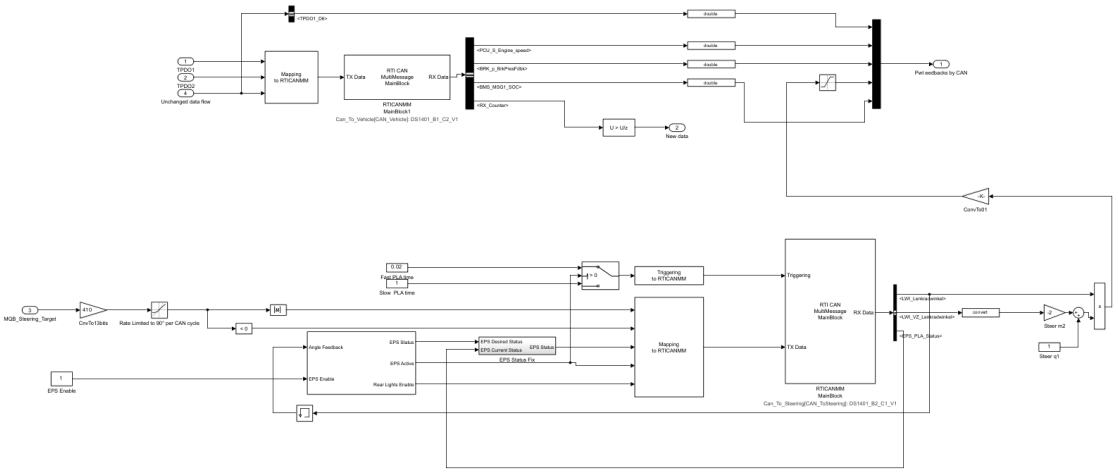

Figure 3.13: Powertrain communication by CAN subsystem

It receives as input the data get from the Remote Control or the ROS Commands, it sends the data to the vehicle via can and receives as output from the vehicle its data. To be analyzed in a more clear way it can be divided in two part: the top part contains the TPDO1 and TPDO2 data which contain information about throttle, brake and steering; the bottom part contains MQB_Steering_Target that it is the pure value of steering from 0 to 1.

3.3.2 Bottom part

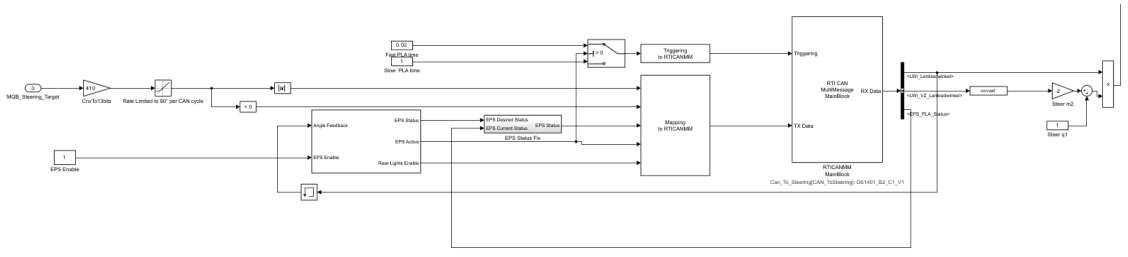

Figure 3.14: Bottom section of Powertrain Communication by CAN

In the bottom part of the block, MQB_Steering_Target is processed, first the data is converted to 13 bits and the rate limited to 90° per CAN cycle with a rate limiter, that limits rising and falling rates of signal by 850 and -850.

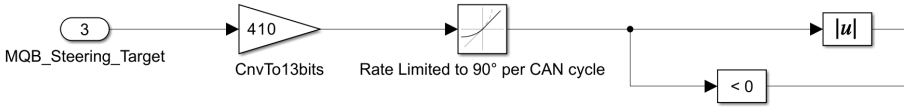

Figure 3.15: Bottom section initialization of the signal

It gives two output: one positive value (absolute value block) and one boolean data (True (1) if negative and False (0) if positive).

These two data are processed in Mapping to RTICANMM.

EPS status fix

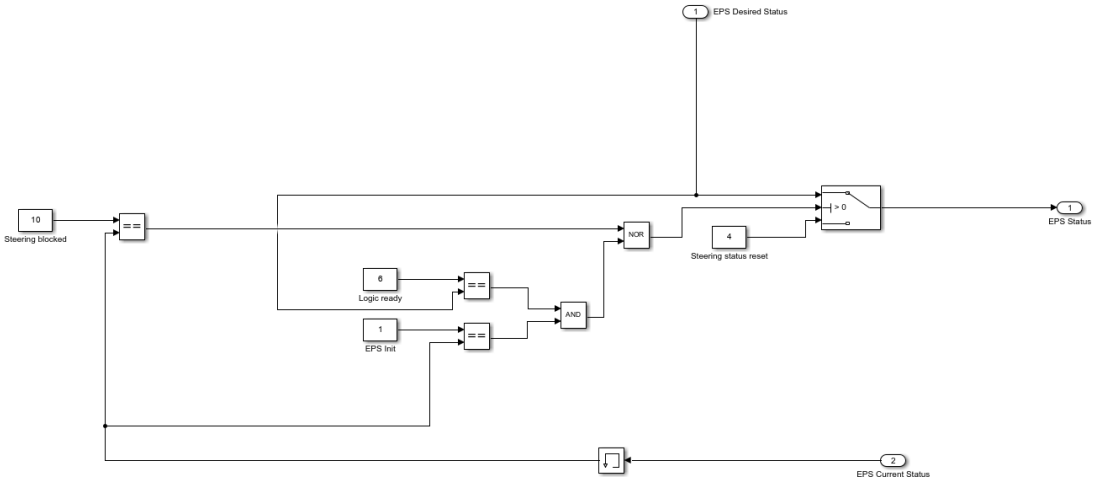

Figure 3.16: EPS status fix

`EPS Status Fix` gives as output the EPS Status comparing the desired status and the current status.

• `EPS Desired Status` is given by EPS Status and calculated by the Angle Feedback.
• `EPS Current Status` is taken directly from the output of the RTI CAN.

The EPS Current Status follow a memory block, that holds and delay its input by one major integration step so the output is the previous input (also the EPS Desired Status present a memory block previously), then is compared with steering blocked and EPS Init.

Steering Blocked → Steering value reset

The EPS Desired Status is compared with Logic ready.

Logic Ready and EPS Init both True → Steering value reset

EPS Current Status	**EPS Desired Status**	**EPS Status**
== Steering Blocked ≠ EPS Init	≠ Logic Ready	Steering value reset
== Steering Blocked ≠ EPS Init	== Logic Ready	Steering value reset
≠ Steering Blocked == EPS Init	≠ Logic Ready	EPS Desired Status
≠ Steering Blocked == EPS Init	== Logic Ready	Steering value reset
≠ Steering Blocked ≠ EPS Init	≠ Logic Ready	EPS Desired Status

Table 3.1: EPS logic table

EPS Status in order to be equal to EPS Desired Status must respect these conditions:

- EPS current state different from Steering blocked;
- EPS Current State equal to EPS Init and EPS Desired State different to Logic ready, or viceversa, or both different.

3.3.3 Upper part

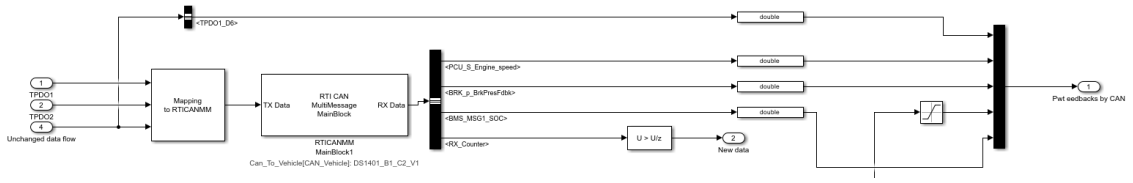

Figure 3.17: Upper section of Powertrain Communication by CAN

In the upper part **TPDO1**, **TPDO2** and **Unchanged data flow** are processed.

Mapping to RTICANMM

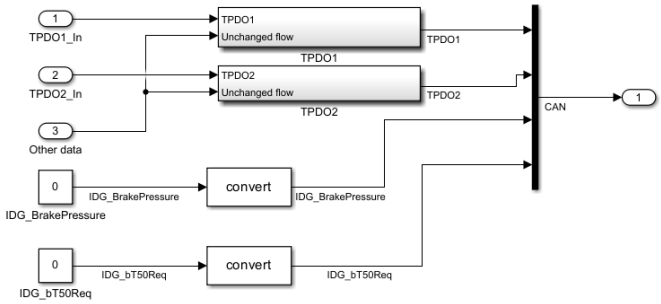

Figure 3.18: Mapping subsection

The mapping block has the purpose to converge all the data into one single signal, TPDO1 and TPDO2 are converged with "Unchanged Data Flow" because there are some signals that are sent only by the remote and not developed by the PC. The output signal is the Transmit Signal that will be sent into the vehicle.

RTICANMM MainBlock

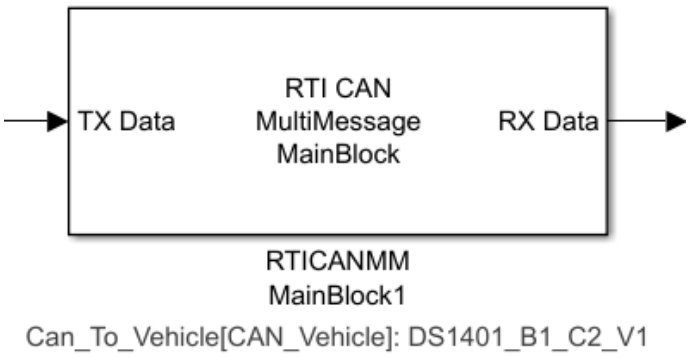

Figure 3.19: RTICANMM MainBlock

This block, differently from the one used in "Remote Control by UDP", has both Tx and Rx, it sends all the signals to the vehicle and receives back the feedback info from it. From this block it is possible to decide which signals are important to be read from the PC in order to ensure proper operation of the controls.
The signal chosen are the following:

- `PCU_status.PCU_S_Engine_speed`, the Current Engine speed [RPM], that can go from 0 to 5000 (in this case it is limited to 100 for safety measure);
- `BRK_Tx_01.BRK_p_BrkPresFdbk`, Current Braking pressure [mBar] applied, that can go from 0 to 50000;
- `BMS_MSG1.BMS_MSG1_SOC`, State Of Charge of the battery [0-100%];
- `RX_Counter`, counter value that with a detect increase is checked if the input is greater or equal of its previous value or not and gives as output of the subsystem a boolean data: New data (it will enter into the Enable Port).

3.3.4 PowerTrain

Finally the powertrain converges all the feedbacks obtained from the vehicle with 5 different data:

- **TPDO1_D6** (UINT16), data obtained from the unchanged data flow that contains the information about which system has the control (ROS or Remote Control);
- **PCU_S_Engine_speed** (FLOAT), current engine speed;
- **BRK_p_BrkPresFdbk** (FLOAT), current brake pressure;
- **STRG_Pct_StrngPstnFdbk** (FLOAT), current steering feedback (obtained from the lower part of this subsystem);
- **BMS_MSG1_SOC** (FLOAT), State Of Charge of the battery.

3.4 ROS Feedback by UDP

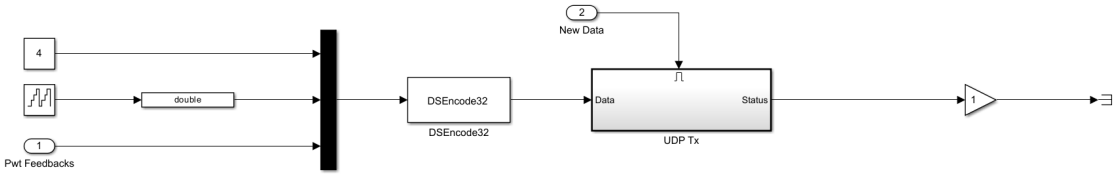

Figure 3.20: ROS Feedback by UDP

This subsystem receives from subsystem "*Powertrain Communication by CAN*" the data acquired from the vehicle via CAN and has the purpose of sending them via UDP to the computer used for ROS Communication.

The *Pwt Feedback* data is multiplexed with other two data, the firt one is the ID and it a constant and the second one is the counter that increase each time its value. The data is then encoded with DSEncode32 (opposite function of what it was done at the beginning with the data get from the UDP receive), it converts an input signal consisting of datatypes into 32Bit WORD data stream on the output port.

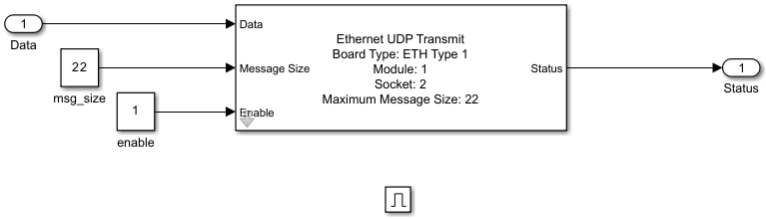

Figure 3.21: UDP Tx

The message size is defined by a constant, which, in this case, is set to 22 bits. This message contains seven different data components, which are transmitted back to the PC to provide real-time information about the vehicle's behavior:

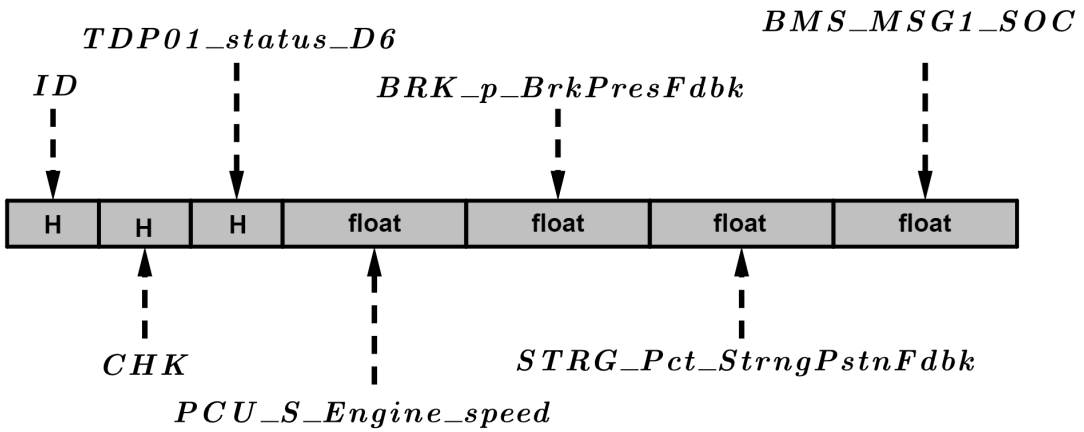

Figure 3.22: Array of transmitted data

▷ **ID** (Unsigned short int (H) - 2B): it is the unique identifier for the UDP packet.

▷ **CHK** (Unsigned short int (H) - 2B): it is the packet counter so it tells the MicroAuto Box II when the packet was created.

▷ **TDP01_status_D6** (Unsigned short int (H) - 2B): it is the flag that tells the vehicle who is in control.
Flag = 0 Remote Command is in control.
Flag = 1 Host PC via UDP is in control.

▷ **PCU_S_Engine_speed** (Float (f) - 4B): it is the current engine speed [RPM].

▷ **BRK_p_BrkPresFdbk** (Float (f) - 4B): it is the current braking pressure [mBar] applied.

▷ **STRG_Pct_StrngPstnFdbk** (Float (f) - 4B): current steering feedback [-1,1].

▷ **BMS_MSG1_SOC** (Float (f) - 4B): current state of charge [0-100].

3.5 ETH and CAN Configurations

Some blocks are needed to allow a correct communication between the MicroAutoBox via CAN and UDP.

3.5.1 ETH communication (UDP)

Regards the UDP communication just one block about the general setup is needed, so the "*Ethernet UDP Setup*" block has to be present in order to allow the UDP communication with the PC.

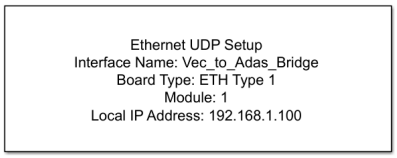

Figure 3.23: ETH confiiguration

3.5.2 CAN communication

In order to allow CAN communication different blocks need to be present, the first one is the "*RTI CAN MultiMessage GeneralSetup*" block, that defines where the generated files for the CAN communications has to be.

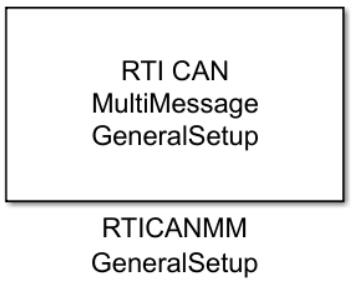

Figure 3.24: CAN MultiMessage GeneralSetup

The other blocks for CAN configurations are the "*RTICANMM ControllerSetup*" blocks, that they describe the hardware and general settings of the physical layout of the CAN protocol.

`CAN To Wireless`: is the block used to receive the signal from the Autec Remote, it corresponds to the block present in "*Remote Control by CAN*" and it receives all the data.

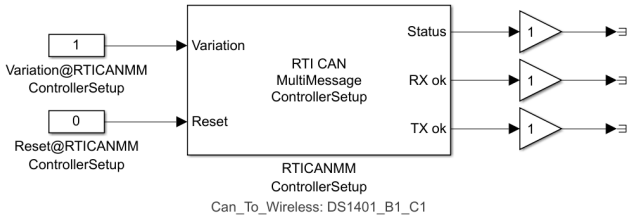

Figure 3.25: CAN To Wireless

`Can To Vehicle`: is the block used to send and receive data to the MicroAutoBox II regarding the Engine Speed, Brake Pressure (also Emergency Brake) and State Of Charge. It corresponds to the block present in the subsystem "*Powertrain communication by CAN*" upper part.

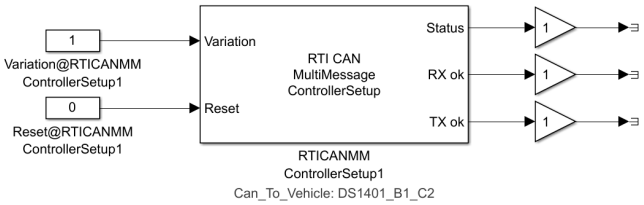

Figure 3.26: CAN To Vehicle

`Can To Steering`: is the block used to send and receive data to the MicroAutoBox II regarding the Engine Speed, Brake Pressure and State Of Charge. It corresponds to the block present in the subsystem "*Powertrain communication by CAN*" lower part.

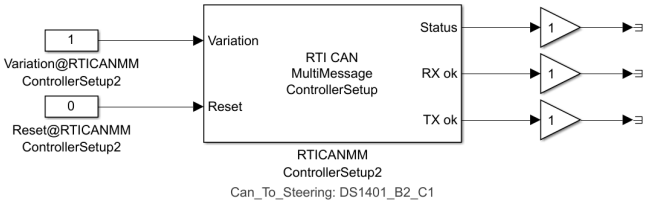

Figure 3.27: CAN To Steering

Chapter 4

Simulink improvements

Upon comprehending the structure and functioning of the Simulink model, an analysis is essential to determine areas where enhancements can be made and sections requiring modification. The majority of these improvements should be focused on the section related to ROS commands, as it possesses less functionality compared to commands via the Autec remote. Therefore, the entire structure needs to be revised to ensure that a greater number of signals can be transmitted via the computer interface while also accommodating an increased volume of signals received via UDP to monitor the vehicle's proper operation and functions.

In this section are explained all the improvements that are been made to the previous Simulink architecture with the aim of making the model more functional and with new functions. Here the Simulink improvements are just described and justified, the testing part where all the upgrades will be checked and where the correct or incorrect functioning will be verified is presented in the following chapter.

4.1 Implementing new signals to be read from the PC

In the simulink architecture studied above only 5 signals are received by the PC regarding the vehicle status, and those are:

- TDP01_status_D6 that is the flag that tells who is in control of the vehicle.
- PCU_S_Engine_speed that is the current engine speed expressed in rpm.
- BRK_p_BrkPresFdbk that is the current braking pressure applied, measured in mBar.
- STRG_Pct_StrngPstnFdbk that is the current steering feedback.
- BMS_MSG1_SOC that is the State Of Charge.

The list of signals that can be chosen using the **RTICANMM MainBlock** is very wide and because of that new systems concerning the vehicle can be detected by the pc.

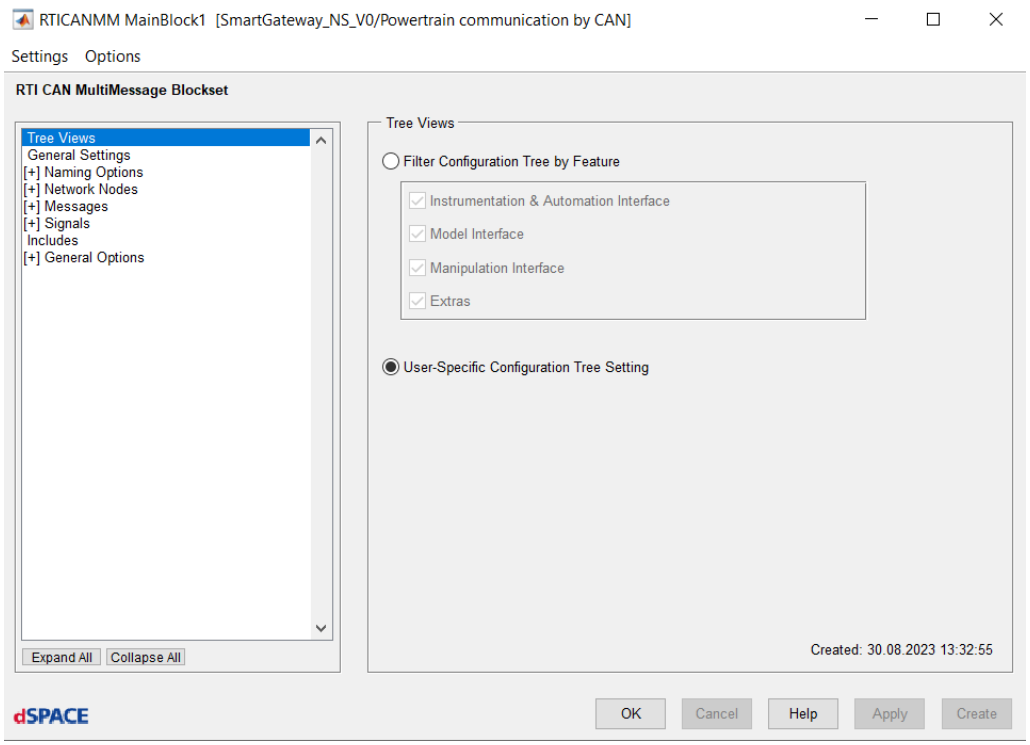

Figure 4.1: RTICANMM MainBlock interface

4.1.1 Signals to implement

Those are the systems that should be detected by finding the right signal to be read:

- **Handbrake signal**, this signal should give information regards the handbrake, ideally with a flag signal that indicate if the handbrake is on with "1" and if the handbrake is off with "0". It is important the implement of this signal because with this feedback it is possible to check the handbrake from the pc without checking it visually and, if the task is the autonomous driving, the fact that the computer know its position is a fundamental step for the correct operation.
- **Reverse signal**, the reverse signal should give information about the direction of the throttle, as studied in the previous chapter the signal regarding throttle is unique for both forward and backward directions, what is different between the two direction is only the signal `TPDO1_Front_Rear_Dir_D5` that acts as a flat between "0" front direction and "1" backward direction. The signal so needs to be able to understand in which configurations the vehicle is on.
- **Charging state**, is already present in the signal the one regarding the State Of Charge of the battery, what it is wanted is a signal that can tell if the vehicle is charging when connected. This signal can help identify some problem that can occur, regards charging situation of the battery.
- **Counter**, a packet counter signal is already present but in that case is a counter created by the Simulink model (with a counter free-running block) and not directly by the vehicle, so in case a better counter directly from the vehicle is present it can be replaced.
- **Motor Temperature**, a signal that indicates the temperature of the motors. It is used to check the correct functioning of the vehicle and to prevent overheating of the motors.

4.1.2 Sending more signals with MicroAutoBox II

To let the MicroAutoBox II sends other signals to the pc it has to be followed a procedure in order to correctly manage and read the signal with a new modified UDP packet.
First of all the new signal has to be selected in the From *RTICANMM Main block* it is possible to visualize all the signals (RX) that are present in the architecture and so the one of interest can be chosen.

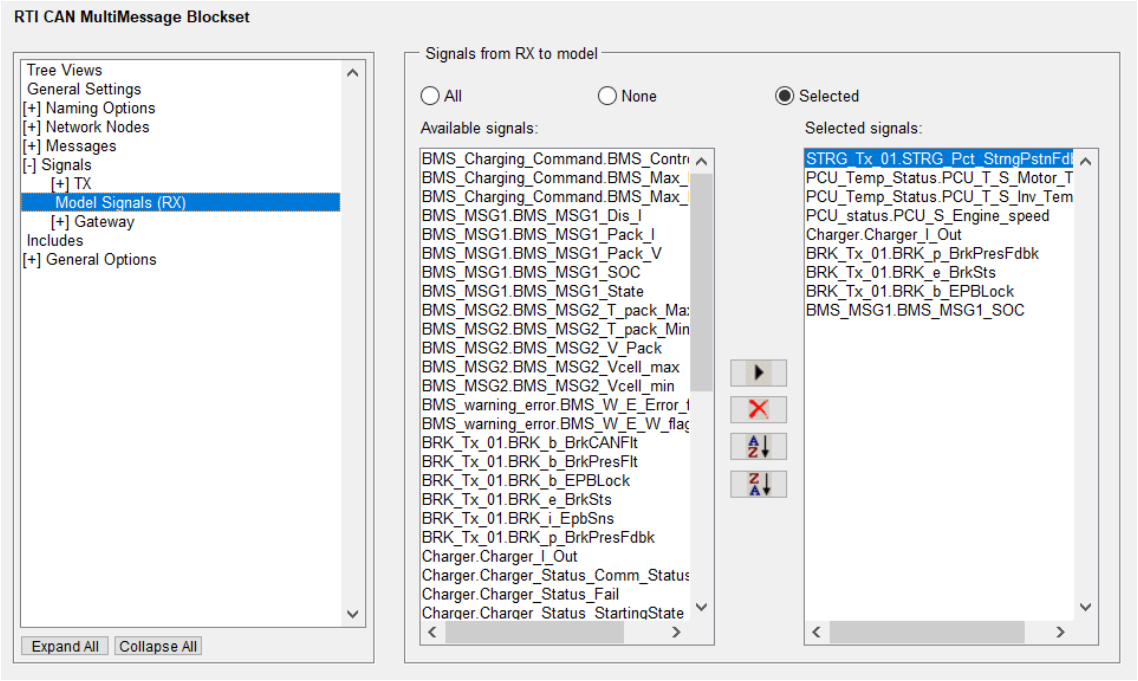

Figure 4.2: RTICANMM MainBlock signals

Then the new signal can be selected from the bus and inserted with the other data. When a new signal is added also some parameters have to be updated in the section "*ROS Feedback by UDP*":

- In `DSDEncode32` a new datatype needs to be added in the structure chosen according to the type of data and its range of values.
- In `Ethernet UDP Transmit` the message size has to be increased based on the datatype chosen previously.

```
Datatype Codes:
BOOLEAN: 1  //  INT8    : 2  //  UINT8   : 3
INT16    : 4  //  UINT16 : 5  //  INT32   : 6
UINT32   : 7  //  FLOAT  : 8  //  DOUBLE: 9
```

Figure 4.3: Kind of possible datatypes

When a signal is chosen it is important to select the right datatype from the one present in the DEDEncode32 (4.3), in the following table all the datatype are resumed with their range of value:

Datatype	full name	**bits**	**bytes**	Range of values
BOOL	boolean	1	1	'True' as 1, 'False' as 0
INT8	Signed 8-bit Integer	8	1	-128 to 127
UINT8	Unsigned 8-bit Integer	8	1	0 to 255
INT16	Signed 16-bit Integer	16	2	$-32,768$ to $32,767$
UINT16	Unsigned 16-bit Integer	16	2	0 to $65,535$
INT32	Signed 32-bit Integer	32	4	-2.1 billion to 2.1 billion
UINT32	Unsigned 32-bit Integer	32	4	0 to 4.3 billion
FLOAT	floating-point number	32	4	range of values with decimals
DOUBLE	floating-point number	64	8	range of values with decimals

Table 4.1: Possible datatype

4.1.3 ControlDesk

Finally, after entering the new data and loading the new Simulink model in the MicroAutoBox, in order to verify the values assumed by the signal two possible path can be followed:

1. Connect the PC with the vehicle and check using the UDP packets the signal added and his changes during the operation of the vehicle;

2. Use the `ControlDesk` program, from which also the new simulink model is uploaded to the MicroAutoBox II. With this program it is possible to control in real time the variable that are present in the Simulink model and so all the signal inserted. This way of checking the variables is simpler and allows for faster checking by not having to go and modify the C++ code as in the previous point.

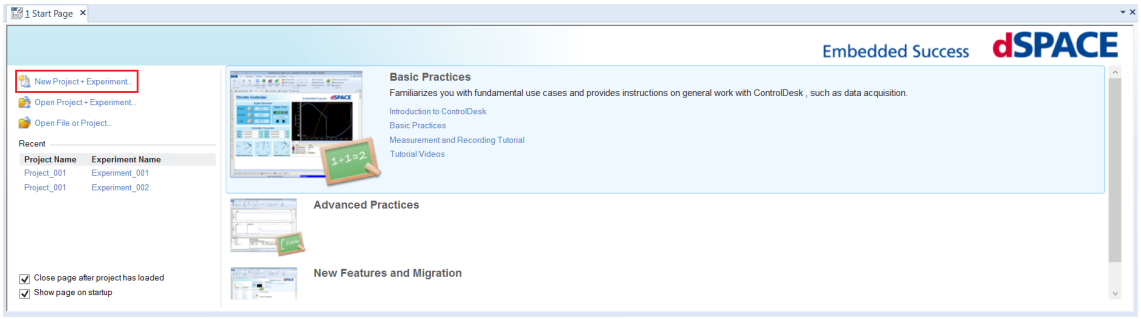

Figure 4.4: ControlDesk interface

4.2 Safety improvement

This improvement is being made for safety measure, because the signals received via UDP by the vehicle have no protection about unwanted values. In case a value outside the range $0 - 1$ in the throttle or in the brake is processed in the simulink architecture, it can cause an undesired behaviour of the vehicle that can may cause, in the worst case scenario, the vehicle to hit some obstacle.
To prevent this problems some saturation blocks are added in the architecture before each value. The working principle of a saturate the signal received between two given values, so for example in case it receive a value of throttle equal to 10 it can be saturated to 1.

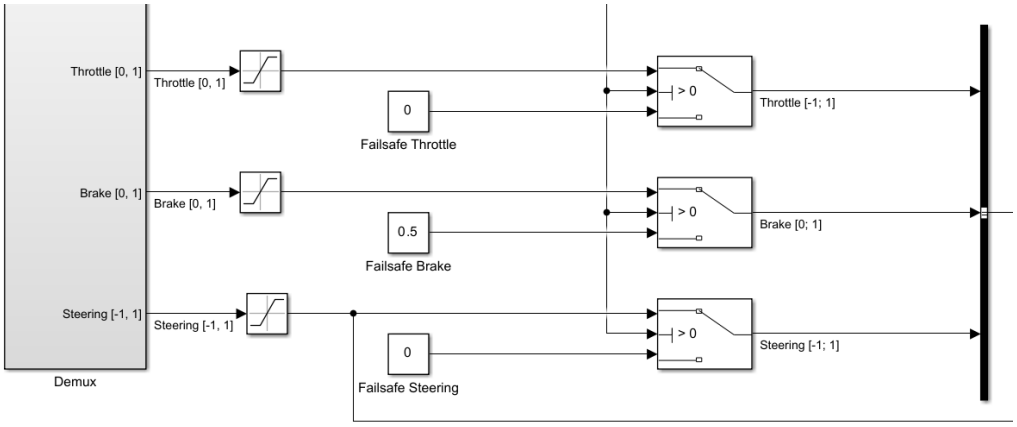

Figure 4.5: Saturation blocks

- Throttle value will be processed by a saturation block from maximum value of 1 and minimum value of 0;
- Brake value will be processed by a saturation block from maximum value of 1 and minimum value of 0;

- Steering value will be processed by a saturation block from a maximum value of 1 and minimum of -1;

4.3 Simplified counter counter

A new counter is being developed in order to simplify the older one that is unnecessary overcomplicated for the function it needs to accomplish. The objective of this kind of counter is:

- To be 0 if the input is True (1).
- To increase if the input is False (0).

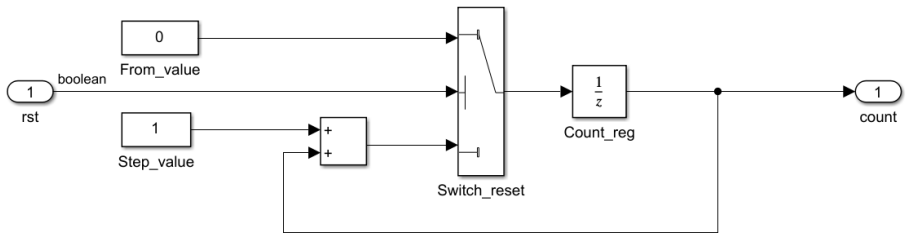

Figure 4.6: New counter configuration

Working principle:
The operation of this switch is as follows: when the input signal (rst) is True (1), the switch maintains its current position, and the output signal (count) remains at 0. However, if the input is False (0), the switch changes its position. Starting from a step_value of 1, the output value incrementally increases as long as the input remains False.

This counter is then compared to a constant value, which represents the watchdog timer. As long as the counter value is less than the constant, no action is taken. However, when the counter value surpasses the constant, it triggers a "failsafe procedure", shown in the following page (Figure 4.7).

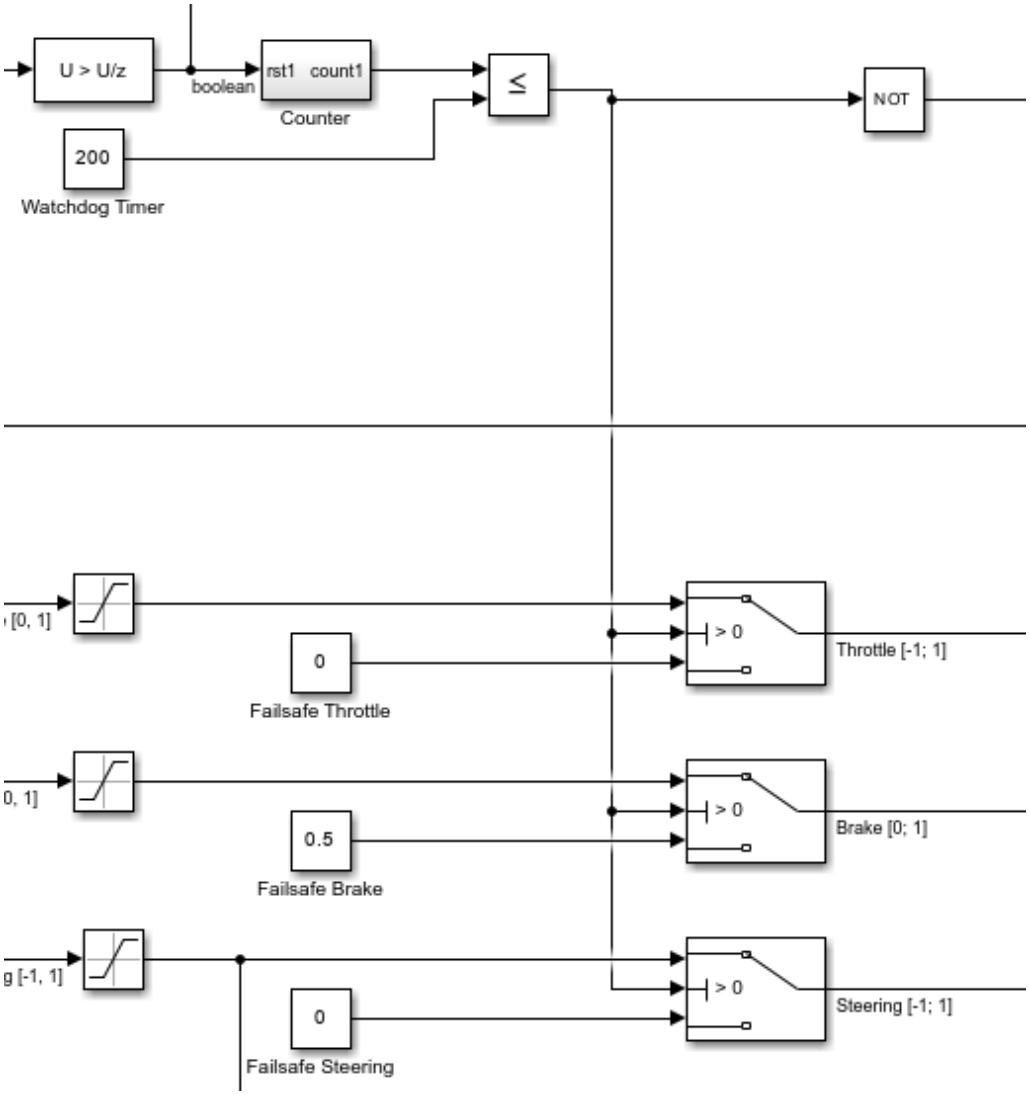

Figure 4.7: Failsafe procedure

4.4 Implement reverse gear on ROS console

The implementation of the reverse gear (in *ROS Commands by UDP*) should be implemented with a similar procedure to how it is already working in the remote control (*Remote Control by CAN*), where a flag signal (0 or 1) TPDO1_front_Rear_Dir_D5 works as a switch to indicate if the throttle signal is for the reverse or not.

The UDP package needs to be increased by 2 bytes, UINT16 is more than enough for a flag signal (it could also be changed with a boolean datatype). This new signal will have only two value:

- 1, the vehicle get a forward acceleration;
- 0, the vehicle get a backward acceleration;

The signal will directly be linked with the Data Rewrap. Therefore it won't be affected by the "failsafe procedure", because that procedure affects directly on the throttle giving it equal to 0, so it won't matter the direction of it.

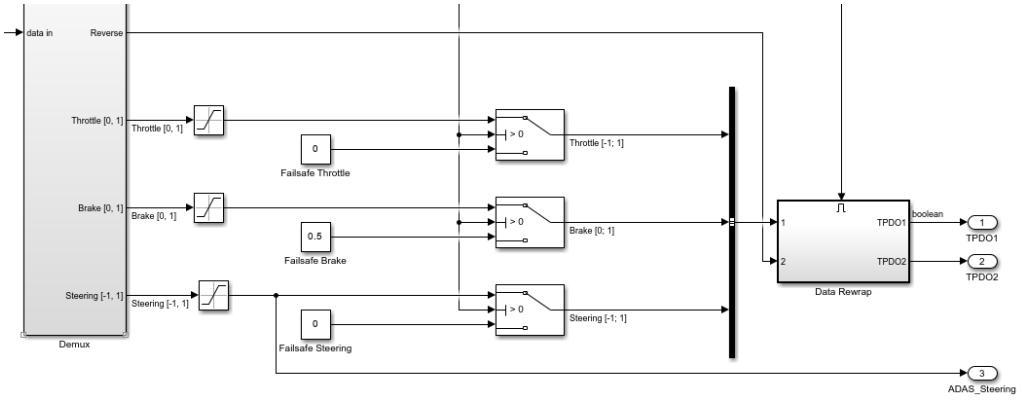

Figure 4.8: Reverse gear

The throttle signal is used for both forward and backward directions but it needs to be considered positive for both direction, only TPDO1_front_Rear_Dir_D5 changes his value to determine if it is a positive or negative throttle.

The Brake signal and the throttle signal will be summed together so it is possible that if we have a low brake value and high negative throttle the vehicle will brake with a value equal to the brake value plus the throttle. The block *"compare to 0"* is just a precaution in case the signal of the reverse is not precisely 1 or 0.

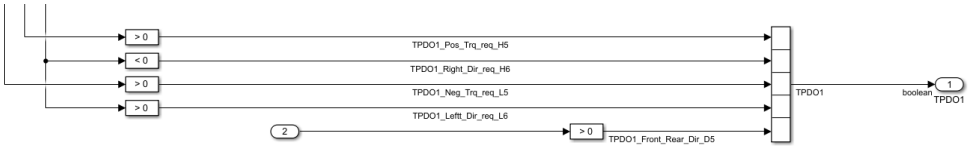

Figure 4.9: Reverse gear data rewrap

4.5 Handbrake command on ROS console

It was not possible with the previous simulink scheme send, via UDP, a command to activate and deactivate the handbrake via ROS console.

To activate the possibility to switch on and off the handbrake the following procedure has to be done:

1. The **Ethernet UDP Receive** must contains one data more (add bytes to the message size, 2 should be more than enough);
2. It needs to add a new datatype to be converted in the **DSDecode32**;

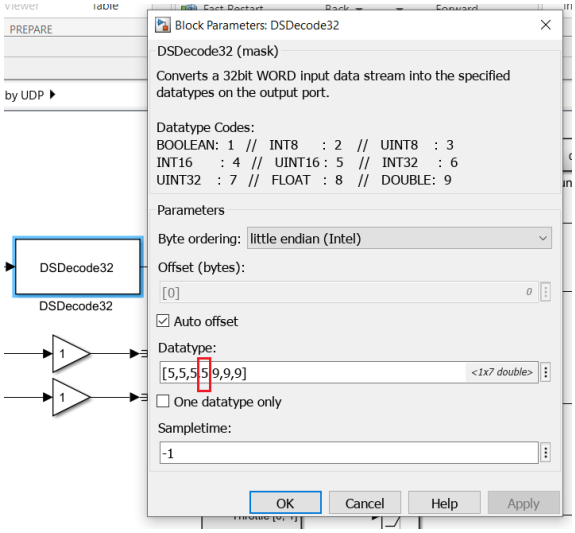

Figure 4.10: DSDecode32 with new signals

3. The **demux** will have one signal more;

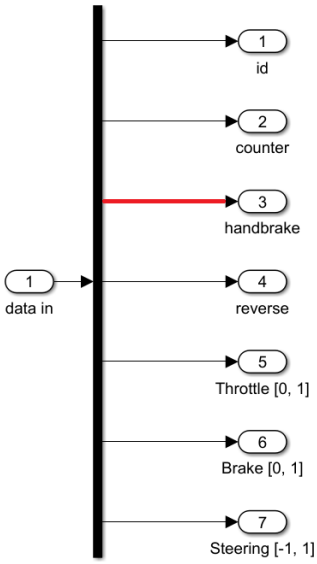

Figure 4.11: Demux with new signals

4. That signal goes directly out the *ROS Commands by UDP* subsystem, that is because it has no utility in the data rewrap but it is used later, and it will enter as input the **Powertrain communication by CAN**; also the command signal will be linked to Powertrain communication by CAN.;

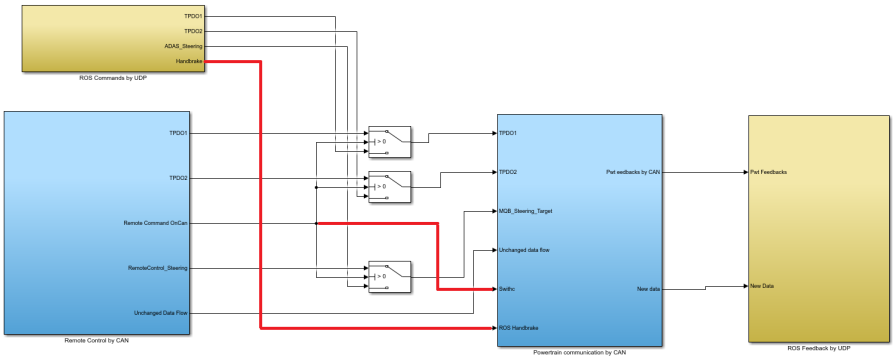

Figure 4.12: Handbrake signals link

5. There will be 2 **Mapping to RTICANMM blocks**, one for the remote control with TPDO1, TPDO2 and Unchanged Data flow and the other for ROS control with the same TPDO1 and TPDO2 and with the new signal of the handbrake;

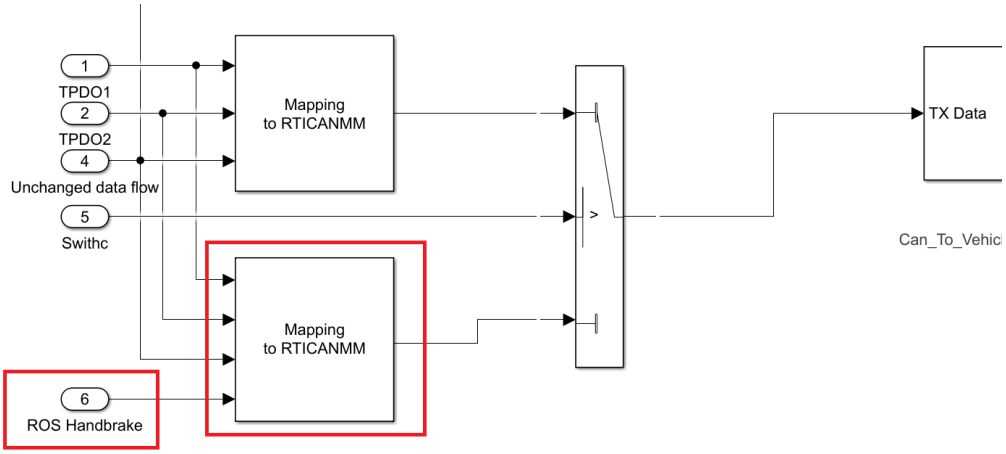

Figure 4.13: Handbrake in Mapping to RTICANMM block

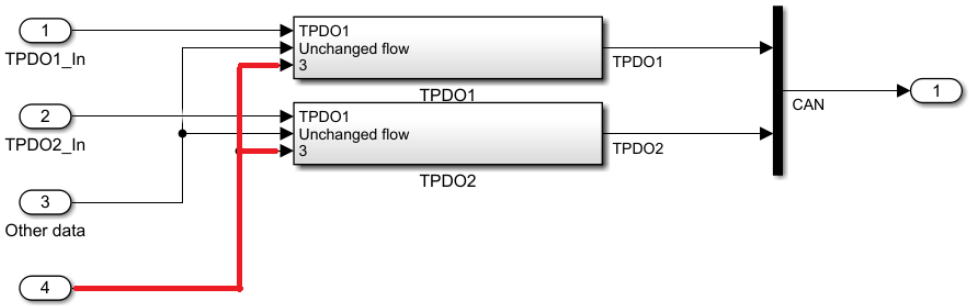

Figure 4.14: Handbrake in Mapping to RTICANMM block

6. The new Mapping to RTICANMM will be the same as the other one with the exception on the TPDO1_D8_Em_Brake, that is taken from ROS handbrake signal.

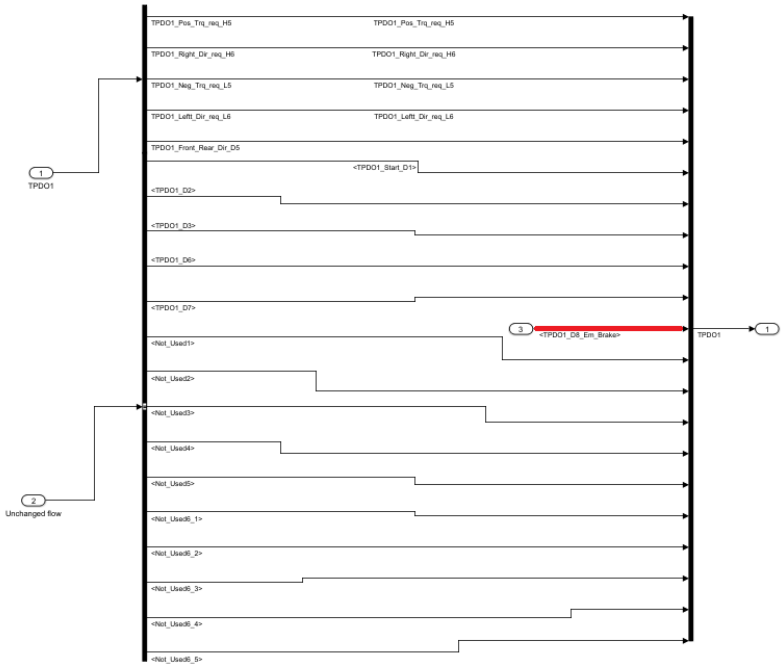

Figure 4.15: Handbrake signal into RTICANMM

7. To decide which data to read a switch will be present, using as condition the data remote Command in CAN.

4.6 Switch between Host PC and Autec Remote

The expansion of functionality also includes an extended capability to switch between the Autec Remote and the Host PC by the Host PC, because previously this functionality was only present in the Autec Remote.

Firstly, the UDP package sent from the PC needs to be increased by 2 bytes, adding a new signal called "ROS command." This signal will be a new output of the subsystem "ROS command by UDP."

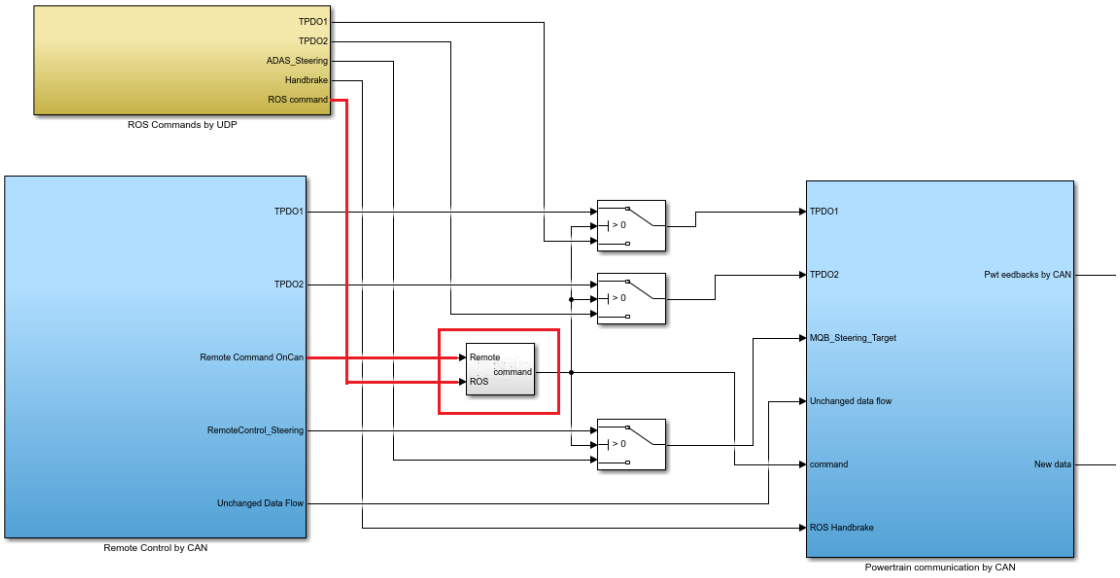

Figure 4.16: Switch block position

The two command data streams, originating from both the PC and the Autec Remote, are funneled into a dedicated subsystem. This subsystem possesses the intelligence to determine the appropriate signal for consideration, making its decision based on the most recent communication method employed. The output from this subsystem is then connected to all the switches, providing the precise indication of which data source should be taken into account for further processing.

This subsystem decide which signal will proceed based on the behaviour of the two inputs:

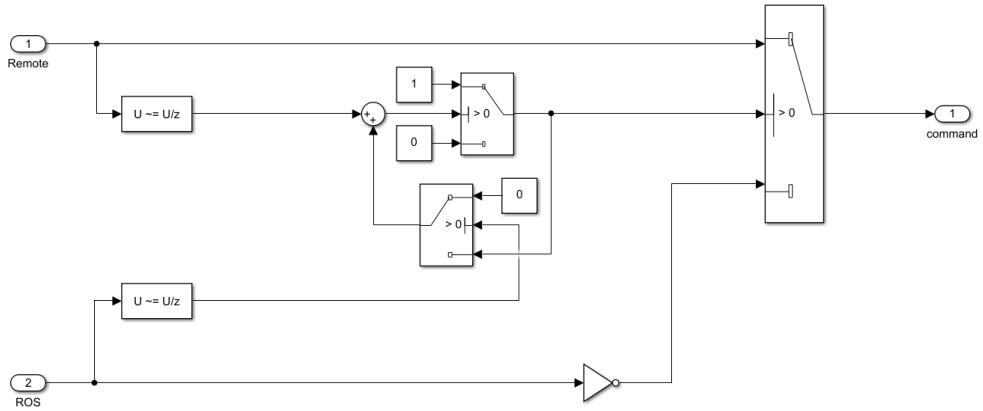

Figure 4.17: Switch block detailed

Inputs:

- Remote, signal from the Autec Remote representing the choice of the Remote on who should command;
- ROS, signal from the PC (UDP package) representing the choice of the PC on who should command;

Output:

- command, the final command that is linked to the control port of the switches that control the commands;

4.7 Improved handbrake control

In the previous simulink architecture in order to activate and deactivate the handbrake a long press of the button was necessary.

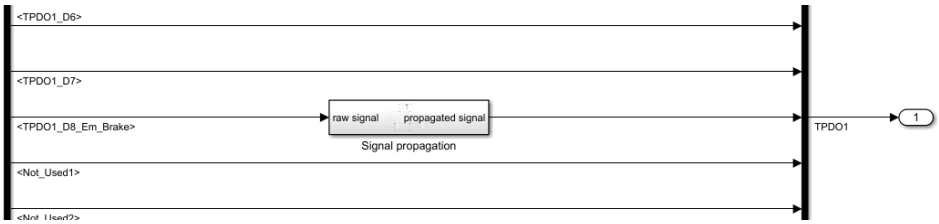

Figure 4.18: Signal propagation of handbrake block

The subsystem below has to be placed in the subsystem Powertrain communication by CAN inside the block Mapping to RTICANMM in the link TPDO1_D8_Em_Brake to increase his signal propagation.

Command given by PC and by Autec Remote have two different Mapping to RTICANMM block and so it is required for both cases.

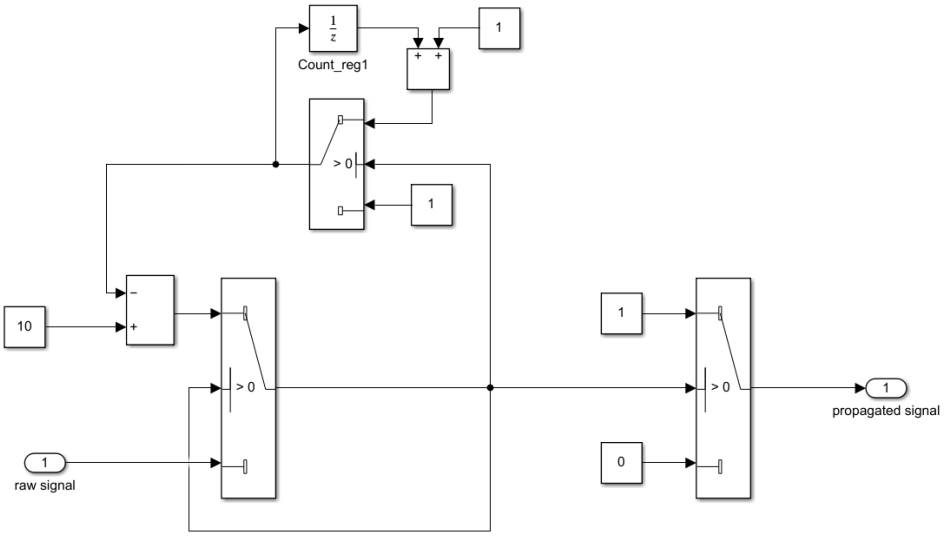

Figure 4.19: Subsystem

Working principle:

1. The input first encounter a compare to constant block that check if the value of the input is bigger than zero;
2. Afterwards a switch is present (the first one encounter by the signal); this switch in a rest condition has as output the input signal and it will change its configuration only when the input change its value (>0) because the control input of the switch is the output itself, when this case happens the output of the switch is maintained bigger than zero for a chosen time;
3. The output of that switch is also linked to the control input of another switch that is just an increasing counter so it increases his output until the value of the control input is bigger than zero;
4. The output of the increasing counter is subtracted from a constant (tuning this constant will increase or decrease the signal propagation);
5. When the output of that add block is equal to zero the output of the first switch is zero and so it changes his configuration in the "rest configuration" until the input changes again;
6. The final switch is present only to have as output of the subsystem a constant signal and not a decreasing one;

The value of the constant needs to be checked because the "signal propagation time" is based also on the sample time and so some tests are needed in order to understand which is the correct value.

Chapter 5

Testing

After completing the study of the previously employed model and developing updates for the new version, this chapter provides a comprehensive explanation of all tests and their outcomes, assessing their utility for the vehicle model.

The initial tests were conducted in a workshop with the vehicle elevated using a mechanical lift, primarily for safety considerations. This precaution was taken to mitigate the potential risk posed by unexpected vehicle behavior, which could result in damage or pose a danger to individuals in proximity.

Subsequently, following the verification that the new model, incorporating the updates, operates correctly and can safely control the vehicle, further tests were conducted outdoors, allowing the vehicle to move freely on its four wheels.

In the forthcoming sections, detailed explanations of the results obtained from individual improvement tests are presented.

5.1 New signals implemented

Throughout the testing phase, numerous signals are meticulously inspected to ensure they align with the intended functions that are to be visualized:

- **Handbrake signal:**
  The following signals were analyzed during the activation and deactivation of the handbrake:
  - *STRG_Pct_StrngPstnFdbk*: the value does not changes during the test.
  - *BRK_Tx_01.BRK_b_EPBLock*: corresponds to 1 when the handbrake is on and to 0 when it is off.
  - *BRK_Tx_01.BRK_e_BrkSts*: corresponds to different value based on the position of the handbrake lever:
    - 1 handbrake switching on;
    - 2 handbrake switched on;
    - 3 handbrake switching off;
    - 4 handbrake switched off.

  Based on the results the signal that is gonna be added in the final model is the *BRK_Tx_01.BRK_e_BrkSts*

- **Reverse signal:**
  In case of the reverse signal none of the signals inside the RTI-CANMM MainBlock is being tested, instead the signal from the Tx data: *TPDO1_Front_Rear_Dir_D6* is used. This signal taken from *'Mapping to RTICANMM'* gives as output the direction of the throttle given by ROS or Autec Remote (depending on the value of *TPDO1_D6*).

- **Charging State**
  All these signals were analyzed by checking their value during charging of the vehicle and subsequent disconnection from the power socket, and vice versa.
  - *BMS_MSG1.BMS_MSG1_State*: it keeps changing his value not depending on the charging state.

- ▷ *BMS_Charging_Command.BMS_Control*: it stays fixed as 0 in both cases.
- ▷ *Charger.Charger_Status_Comm_Status*: It remains fixed at 0 in both cases.
- ▷ *Charger.Charger_Status_StartingState*: It remains fixed at 0 in both cases.
- ▷ *BMS_MSG1.BMS_MSG1_Dis_I*: his value keep changing not depending on the charging state.
- ▷ *Charger.Charger_I_Out*: when it is not charging it is equal to 0 and when plugged into the outlet it stars to increase from 1 to 13.
- ▷ *Charger.Charger_V_Out*: it stays close to 88 increasing a little bit when charging.

Based on the results the signal that is gonna be added in the final model is the *Charger.Charger_I_Out*, with a simple architecture shown in figure 5.1, that with a simple "compare to constant" block will has as output 1 ('True') if the value received is from 1 to 13 and so if it is charging, while it will has as output 0 ('False') if it is not charging.

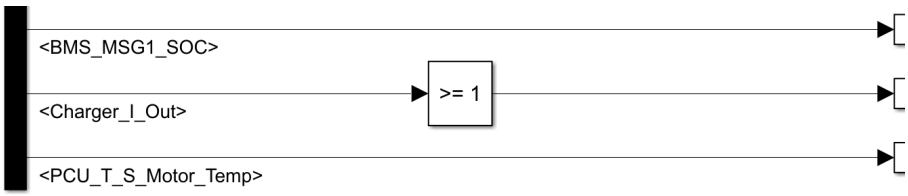

Figure 5.1: Compare to constant in charger signal

- ▷ **Counter**
  All the counter signals tested have a value that remains fixed at 1. Consequently, this signal is no longer added, and the counter created by the Simulink model is retained.

- ▷ **Motor temperature**
  In this case only one signal inside the RTICANMM MainBlock can represents it *PCU_Temp_Status.PCU_T_S_Motor_Temp*. This signal is then tested by stressing the motor and when the vehicle is

moving for a long time his value increase (and otherwise decrease when not used).

All the signals distribution can be seen more clearly in the figure 5.2 and in the table 5.1 below.

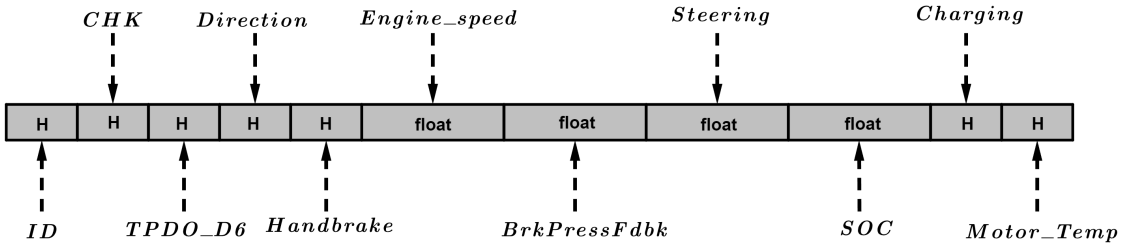

Figure 5.2: Array containing the new signals

Field	Type - Size	Meaning
ID	Unsigned short int (H) - 2B	Unique identifier for the UDP packet.
CHK	Unsigned short int (H) - 2B	Packet counter.
TPDO1_D6	Unsigned short int (H) - 2B	Flag that tells who is in control.
Direction	Unsigned short int (H) - 2B	indicates the direction of movement of the throttle.
Handbrake	Unsigned short int (H) - 2B	indicate the position of the handbrake lever
Engine speed	Float (f) - 4B	Current Engine speed [RPM].
Brake pressure	Float (f) - 4B	Current Braking pressure [mBar] applied.
Steering	Float (f) - 4B	Current steering feedback [-1,1].
SOC	Float (f) - 4B	Current state of charge [0-100]
Charging	Unsigned short int (H) - 2B	Flag that tells if the vehicle is charging or not
Motor Temp	Unsigned short int (H) - 2B	Value that represents the motor temperature

Table 5.1: Tabular of all the signals summarized

5.1.1 Updated architecture

After testing all the possible signals, four of them are added in the update version of the model.
their position was chosen for convenience and for the logic of the signal, placing "*Direction*" and "*Handbrake*" signals at the beginning of the array next to the signal of the system in command, and placing "*charging*" at the bottom of the signal next to SOC and "*Motor Temp*" as the last signal. The new architecture is shown in the figure 5.3 below.

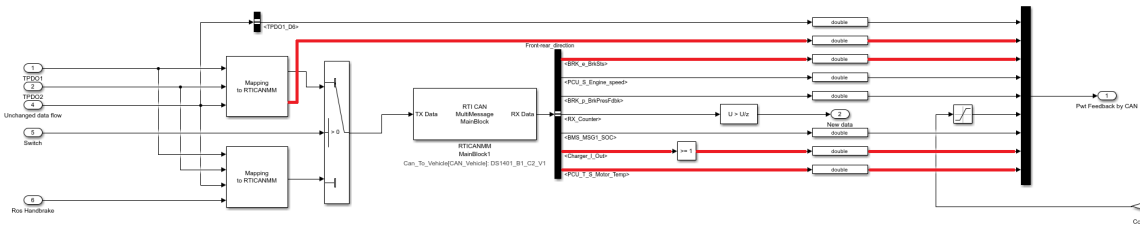

Figure 5.3: Model architecture with new signals

5.1.2 Packets optimization

With the addition of the new information tested above the updated signal will have a dimension of 30 bytes, from the 22 bytes of the original signal. In order to try to optimize the UDP packets received by ROS an analysis on all the datatype has being made to select the "lightest" possible datatype for each signal, checking the table 4.1.
In the table 5.2 the changes are shown from the current datatype (C. Datatype) to updated datatype (U. Datatype).

Data	C. Datatype	**Bytes**	Range of values	U. Datatype	**bytes**
ID	UINT16	2	fixed to 4	UINT8	1
CHK	UINT16	2	Increasing value	UINT16	2
TPDO1_D6	UINT16	2	0 or 1	UINT8	1
Direction	UINT16	2	0 or 1	UINT8	1
Handbrake	UINT16	2	0 to 4	UINT8	1
Vehicle speed	float	4	Decimals needed	float	4
Brake pressure	float	4	Decimals needed	float	4
Steering	float	4	Decimals needed	float	4
SOC	float	4	0 to 100	UINT8	1
Charging	UINT16	2	0 or 1	UINT8	1
Motor Temp	UINT16	2	Range of T	INT8	1
Current packet size:		30	Updated packet size:		**21**

Table 5.2: Packet optimization

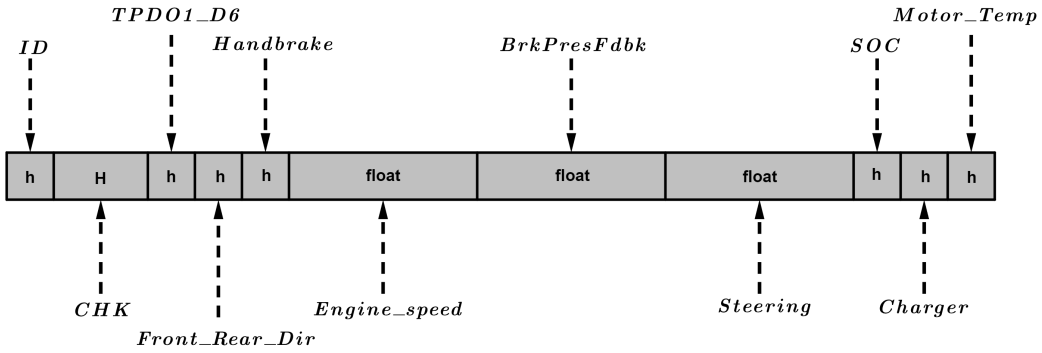

Figure 5.4: Optimized packet array

The optimized packets have a dimension of 21 bytes, 9 bytes less than the current packages and also 1 byte less than the packets of the original model with 4 signal less.

5.2 Safety improvements

About this improvement unfortunately it cannot be tested directly on the vehicle the proper operation of the saturation blocks, that is because the code development for this autonomous car prototype cannot send wrong signals of throttle brake and steering, for this reason it is tested only with a Simulink simulation (figure 5.5).
The saturation block, as their working principle, saturate all the input signals in a range of value predetermined and because of that, in case values of throttle, brake or/and steering received by UDP are outside their limit values, the model would not have to deal with them because they will be converted.

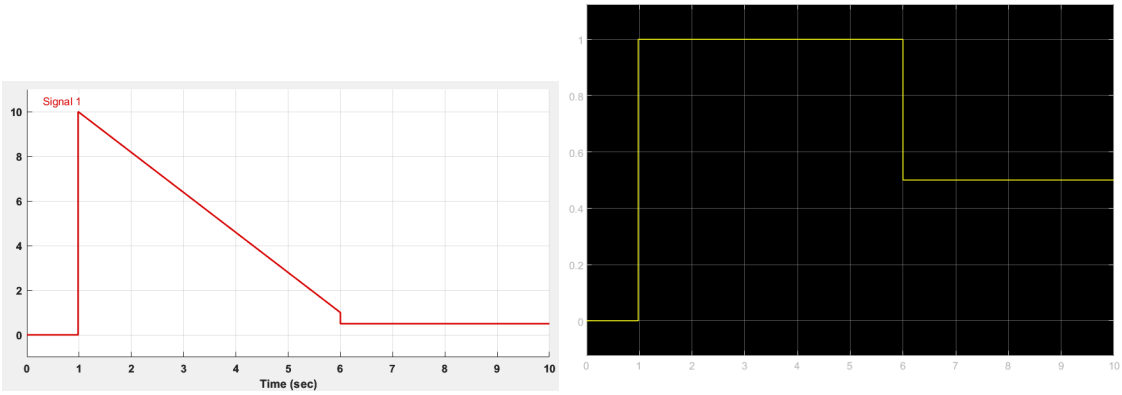

Figure 5.5: Safety simulink test

In the figures above it is possible to see that in case of an unwanted value (in this test equal to 10) the saturation block saturates the value to 1, while allowing the other signals inside the range of saturation to pass unchanged (in this test equal to 0.5).

5.3 Simplified counter

The new counter in the simulation it works fine, unfortunately it was not possible to test the failsafe procedure because the packets coming are set to keep increasing their counter and so it will be activated in exceptional cases.
For this reason the test was made using simulink simulation and the results of the simulation is shown in the figures 5.6 below. The counter when receiving a signal equal to 0 (so when the packet counter is not increasing) it starts to increase his value until it receives a signal equal to 1 (that means order restored) that will reset the counter.

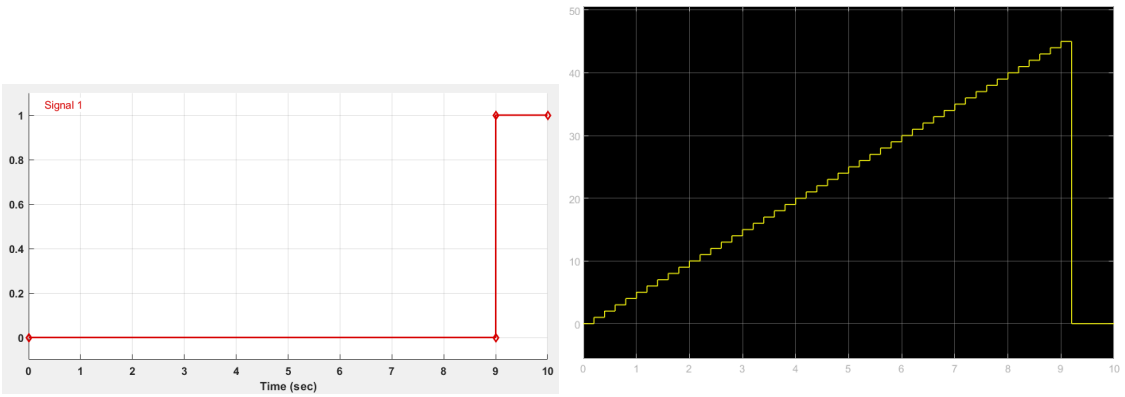

Figure 5.6: Counter simulink test

5.4 Reverse gear on ROS console

The reverse signal functions flawlessly as expected, without any issues. With this signal, it is now possible to activate reverse on the vehicle from ROS via UDP simply by sending a positive throttle and a reverse signal equal to 0.
This enhancement is arguably one of the most significant, as it opens up the opportunity for the autonomous car prototype to commence testing algorithms for parking maneuvers.

5.5 Handbrake command on ROS console

The Handbrake command also works as expected, so all the study made in the previous chapter was correct.
The new structure of the packets received by the MicroAutobox II via UDP is the following:

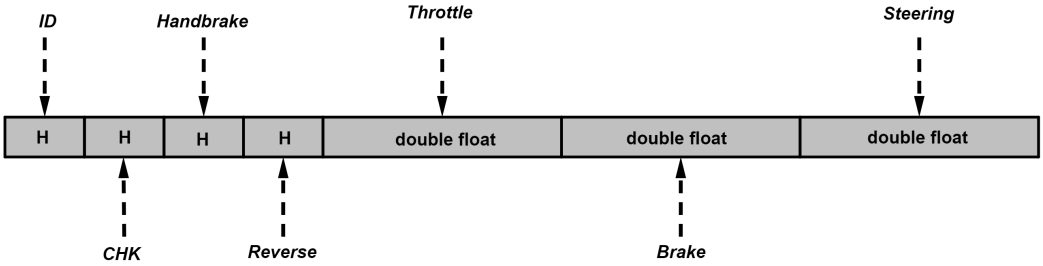

Figure 5.7: Array IN containing the new signals

Field	Type - Size	Meaning
ID	Unsigned short int (H) - 2B	Unique identifier for the UDP packet and it must be equal to 3 in order to work
CHK	Unsigned short int (H) - 2B	It's the value related to the packet counter, tells the MicroAuto Box II when the packet was created.
Handbrake	Unsigned short int (H) - 2B	Flag signal (0 or 1) that represents the handbrake command, when is equal to 1 the handbrake needs to be inserted or removed.
Reverse	Unsigned short int (H) - 2B	Flag signal (0 or 1) that indicates if the reverse gear is on or off.
Throttle	Double float (d) - 8B	Value of throttle that should be between 0 and 1; sets the RPM to che chosen value.
Brake	Double float (d) - 8B	Value of brake that should be between 0 and 1; sets the braking pressure to the chosen value.
Steering	Double float (d) - 8B	Value of steering that should be between -1 and 1; sets the steering angle to the chosen value.

Table 5.3: Tabular of all the signals IN summarized

5.6 Switch between Host PC and Autec Remote

Regrettably, this improvement needs to be removed due to a critical issue. When the signal is switched from the Autec Remote to ROS, the packets sent by the remote continue to be received by the MicroAutoBox II. However, in the reverse scenario, when the signal is switched from ROS to the Autec Remote, the MicroAutoBox II no longer receives the ROS packets. Consequently, it becomes impossible to switch back using ROS alone, and the continued use of the Remote is necessary.
As a result, this block will not be implemented in the final model, and the system will maintain the capability to switch between the two modes of command using the Autec Remote for the autonomous car prototype.

5.7 Improved handbrake

The improved handbrake signal has been implemented in both signals received from ROS via UDP and from the Autec Remote via CAN.
It functions correctly, with the designated value for enabling and disabling the handbrake set at 1000. This value strikes a balance, ensuring that it's not too low to not activate the handbrake and not too high to trigger a secondary activation immediately.
While it's not feasible to demonstrate the physical implementation with the Autec Remote and the vehicle, a simulation conducted in Simulink is illustrated in Figure 5.8 to provide a visual representation of the process.

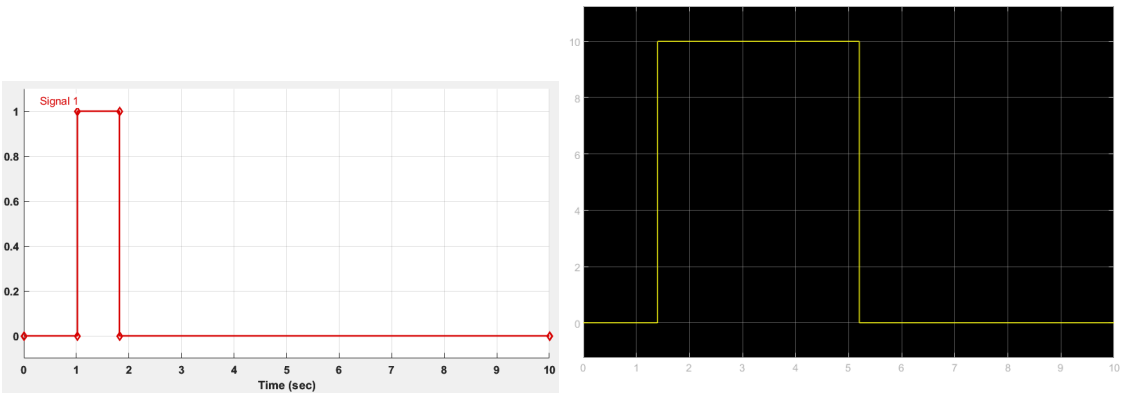

Figure 5.8: Propagation simulink test

This addition significantly enhances the ease of using the handbrake, as it only requires pressing the button on the remote control (or sending

the signal from the PC), even for just a brief moment. This simplification makes activating and deactivating the handbrake an extremely straightforward and immediate process, contributing to an overall improved vehicle control experience.

Chapter 6

Conclusion and future improvements

In conclusion, this thesis project involved a comprehensive analysis of a previously created Simulink model for an initial version of the prototype. This analysis resulted in a detailed documentation of the model's structure, which can serve as a valuable reference for any future modifications.
The study enabled the implementation of several enhancements to the Simulink architecture, facilitating more precise and safe control of the autonomous car prototype. Among these improvements, the addition of the reverse signal stands out, allowing for reverse engagement via ROS. This enhancement initiated the testing and research phase for algorithms related to parking maneuvers, which were previously inaccessible with the older model (due to the absence of reverse capability).
The testing phase of these improvements yielded positive results, with most of them functioning as expected and ready for integration into the final Simulink model.

As the project is still in its early stages, it is likely that subsequent versions of this Simulink model will be developed, incorporating new functions, signals, and possibly alternative communication protocols.

Looking ahead, this project is poised for continuous growth and evolution. It will continue to expand by adding new signals, architectures, and potentially novel communication modes between the MicroAutoBox II and the computer. The future trajectory of the Simulink model promises to be intriguing and dynamic, as it continues to play a crucial role in the advancement of autonomous driving technology.

Bibliography

[1] Review on self-driving cars using neural network architectures, Srinivas Rao P, Rohan Gudla, Vijay Shankar Telidevulapalli, Jayasree Sarada Kota and Gayathri Mandha. www.researchgate.net

[2] ADAS: Features of advanced driver assistance systems, July 1, 2017 by Sam Francis. roboticsandautomationnews.com

[3] SAE Levels of Driving Automation, May 3, 2021. https://www.sae.org/blog/sae-j3016-update

[4] SAE levels of automation in cars, June 9, 2022 by Rambus Press. https://www.rambus.com/blogs/driving-automation-levels/

[5] Waymo https://waymo.com/.

[6] How Does a Self-Driving Car See? April 15, 2019 by Katie Burke. https://blogs.nvidia.com/blog/2019/04/15/how-does-a-self-driving-car-see/

[7] Autoware documentation. https://autowarefoundation.github.io/autoware-documentation/main/

[8] Autoware universe. https://autowarefoundation.github.io/autoware.universe/main/

[9] MicroAutoBox II. https://www.dspace.com/en/inc/home/products/hw/micautob/microautobox2.cfm

[10] MicroAutoBox II RTI. https://www.dspace.com/en/inc/home/products/sw/impsw/real-time-interface.cfm#175_25028

[11] RTI CAN MultiMessage Blockset. https://www.dspace.com/en/inc/home/products/sw/impsw/rti_can_multimessage_blockset.cfm#179_25165

[12] ControlDesk.https://www.dspace.com/en/inc/home/products/sw/experimentandvisualization/controldesk.cfm

[13] RTMaps Interface Blockset. https://www.dspace.com/en/pub/home/products/sw/impsw/rtmaps-interface-bs.cfm

[14] Platform API Package. https://www.dspace.com/en/ltd/home/products/sw/test_automation_software/platform_api_package.cfm#179_25420