POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING

Master degree course
Mechatronic Engineering

Master Degree Thesis

Numerical and Experimental Study of a HIL Test Bench for Electric Work Vehicles

Supervisors
Prof. Aurelio Somà
Prof. Francesco Mocera

Candidate
Carmen Ferri

April 2025

Summary

The increasing use of electrification in both road vehicles and work vehicles has shown the need to use robust, efficient, and flexible testing methods to support the development and validation of electric vehicles (EVs). Today, validation methods involve physical prototyping and real-world testing, which is costly and slow. In this context, X-in-the-Loop (XIL) strategies play a key role in the design and testing cycle, providing tools for the analysis and verification of control units without the need for a physical prototype. Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) are key methodologies for validating complex systems, allowing control software to be tested in a simulated environment and real hardware to be integrated into a real-time simulation.

The aim of this thesis work is to translate the functionalities of a real test bench into a SIL configuration, allowing tests to be performed directly in a virtual environment. The real test bench is the one built by the company Ecothea Srl, developed for the validation of electric powertrains designed for work vehicles (NRMM - Non-Road Mobile Machinery). Peak System's CAN-USB conversion devices are used to interface the test bench with the software on the PC.

A SIL configuration of the test bench was required to test the accuracy of data exchange. Since the latter is the main objective of this work, the numerical model developed in Simulink/Simscape to reproduce the dynamic behavior of the test bench is very simple. It includes electric motors and inertial load to simulate the dynamic of the system, PI controllers for speed control, BUS DC for power supply and energy recovery, and virtual CAN network management for data exchange. The MATLAB Vehicle Network Toolbox was used to configure the transmission and reception of CAN messages. To facilitate interaction with the system, a graphic user interface (GUI) has been developed in MATLAB App Designer,

designed to set test parameters and monitor model variables in real time by managing transmission and reception of CAN messages.
The validation of the SIL configuration was carried out in different stages. After ensuring that the communication between Simulink model and GUI was correct, the model was enriched with strictly numerical models to simulate heavy data traffic on the CAN network, so that the data exchange between ECUs is accurately replicated as in the real test bench. The behavior of the SIL model was evaluated with tests on real hardware using CAN-USB devices to transmit and receive messages after making modifications to ensure consistent data flow in real-time. The tests proved that the simulation and experimental data were consistent, highlighting the robustness of the software used to manage message flow through the CAN network.
In summary, this work illustrates how the switch from a physical test bench to a SIL configuration is an effective way to validate CAN-based control and communication strategies, decreasing development time and costs, and providing a versatile testing platform.
For future development, the test bench model can be improved to accurately reproduce the behavior of the actual system. Integrating the HIL architecture and replacing the Simulink model with a physical test bench will be the next step, allowing direct testing of the control software under real operating conditions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objectives of the research

The increasing use of electrification in both road vehicles and work vehicles has shown the need to use robust, efficient, and flexible testing methods to support the development and validation of electric vehicles (EVs). EVs are made of complex and interconnected subsystems, such as power electronics, battery management systems, and regenerative braking that must ensure safety, reliability, and optimal performance. Today, validation methods involve physical prototyping and real-world testing, which is costly and slow. X-in-the-Loop (XIL) methodologies provide a structured approach to testing by integrating different levels of simulation and hardware interaction. Among these strategies, Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) are key methodologies for validating complex systems, allowing the control software to be tested in a simulated environment and the real hardware to be integrated into a real-time simulation. This approach is particularly useful when there is the need to interface critical physical components with a real-time simulate that replicates dynamic conditions, allowing to avoid physical testing.

One of the most important aspects of HIL testing is the need for robust, real-time communication between the simulated and physical components. Controller Area Network (CAN) BUS enables efficient data exchange between electronic control units (ECU), ensuring synchronous operations and precise data transfer.

The aim of this thesis work is to translate the functionalities of a real test bench into a SIL configuration. The real test bench is the one built by the company Ecothea Srl, developed for the validation of electric powertrains designed for work vehicles (NRMM - Non-Road Mobile Machinery). In particular, the focus area is CAN communication to ensure the accuracy of data exchange. The primary goal of this work is to improve the timing and efficiency of data exchange in a CAN-based system for SIL testing, laying the foundations for future HIL experiments in the real testbench.

1.2 X-in-the-Loop (XIL) Strategies in System Engineering and Product Development.

X-in-the-Loop (XIL) strategies are a set of test methodologies that allow a progressive integration of simulated models and real components during the different stages of development of complex systems. These methodologies are based on an iterative approach, reducing the need for physical prototypes in the early stages of the project and improving the reliability of system before their final implementation. The integration of XIL strategies into the development process follows the V model of systems engineering, which describes the design and validation path of a complex system, as shown in Figure 1.1 . The left part of the diagram (development and design phase) contains the requirement definition, the system architecture and software design; this part is run from the top to the bottom. The rigth part of the model (testing and validation phase) implements tests at different levels, from the software to physical systems, to guarantee that the final product meets the initial requirements, it is scrolled from bottom to the top. The central part of the diagram (implementation) includes the development or physical implementation of the system.

Each left side layer has a corresponding test layer on the right side, which verifies its correctness and compliance with requirements. This model highlights the need for continuous testing and validation, ensuring that each stage of development is supported by appropriate tests before moving to the next stage.

The different phases of the V-model are the following.

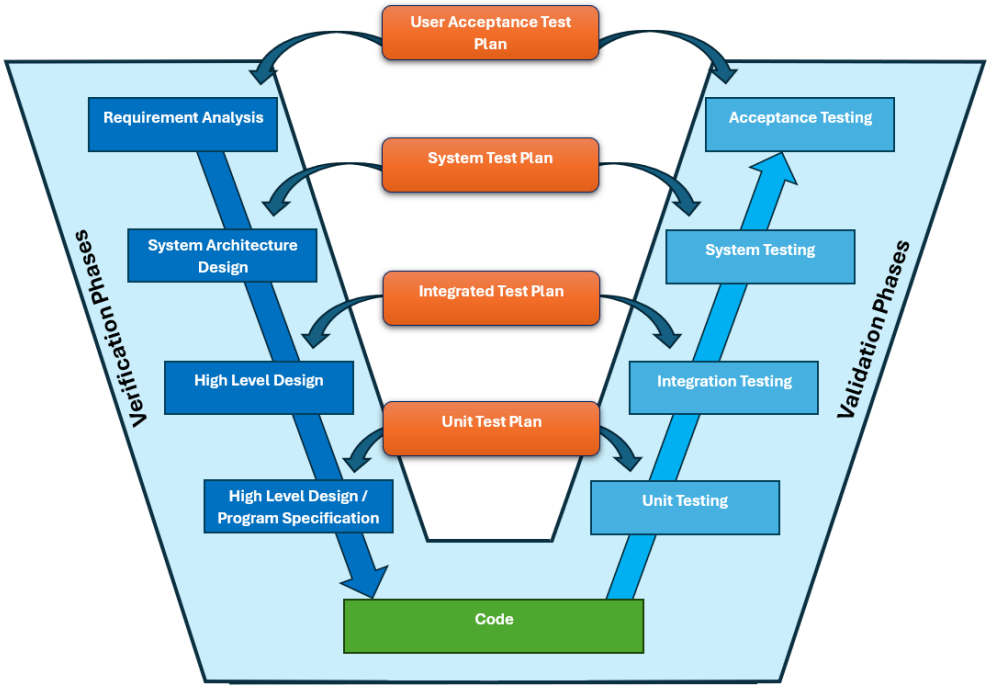

Figure 1.1. V Model.

- **Phase 1:Requirement Analysis**.

  In this initial phase, the requirements that the system must meet are collected and formalized. Both functional requirements, which specify the functions the system must perform (e.g., motor speed control), and non-functional requirements, such as performance, energy consumption, safety, and compliance with applicable regulations, are defined.This phase often involves the drafting of detailed documentation. The involvement of all stakeholders (engineers, customers, regulatory bodies) is essential to avoid ambiguity and ensure that the project is aligned with expectations and applicable constraints.

  Associated Verification: User Acceptance Test Plan. At the end of the development cycle, User Acceptance Testing defines acceptance criteria, test scenarios and use cases that will verify that the final product meets the exact needs of the user. These tests are typically run in an environment that simulates real-world system usage and include usability, performance

and functionality tests.

- **Phase 2: System Architecture Design**.

  Following the definition of requirements, the system architecture is designed, breaking it down into subsystems and modules. Here, the interactions between hardware and software are established, defining the organization of control units (ECUs), communication interfaces (e.g., CAN, LIN, Ethernet), and the involved sensors and actuators. Technological platforms, such as microcontrollers and onboard processors, are selected, ensuring they support the required performance constraints. Furthermore, data and communication management strategies are defined to ensure the system can operate in real-time without critical delays.

  Associated Verification: System Test Plan. This level defines the main components of the system, their interactions and the interfaces between hardware and software. The System Test Plan is designed to verify the overall integration of the system, ensuring that all components communicate correctly and that the system as a whole complies with the required technical specifications and performance. Tests at this level include load, safety and compatibility tests

- **Phase 3: High Level Design**.

  This phase focuses on the detailed design of each system module. From a software perspective, mathematical models and control algorithms are developed, using tools like Simulink for modeling. On the hardware side, electronic circuits, PCB layouts, and physical components are designed and selected. At this point, communication protocols and interfaces between software and hardware modules are also established. Simulations are used to verify that the expected behavior is consistent with the design specifications before proceeding with code and physical part development.

  Associated Verification: Integrated Test Plan. This phase divides the system into modules or subsystems, defining how they interact. The Integrated Test Plan is designed to verify that interfaces between modules conform to specifications and that communications between components are

correct. Both communication interfaces and the cohesion and integrated behaviour of components are tested to prevent incompatibility problems and malfunctions during integration.

- **Phase 4: High Level Design / Program Specification**.

  Here, the system is brought to life: software is written, optimized, and potentially automatically generated (e.g., with Embedded Coder). Simultaneously, hardware is produced, assembled, and integrated with the software. This phase represents the lowest point of the "V," the critical transition between design and verification. It is when the initial challenges of practically implementing designed solutions are faced.

  Associated Verification: Unit Test Plan. In this phase, each module is designed at a very detailed level, including flowcharts, pseudo-code and definition of internal functions and interfaces. The Unit Test Plan establishes test cases to verify that each module functions independently, meeting defined functional requirements and without logical errors. Automated test frameworks are used to isolate the module under consideration and simulate expected inputs.

- **Code: Lowest Part of the Diagram.**

  This phase represents the "bridge" between design and testing. The code is written following the detailed specifications of phase 4. Even if it is not associated with a specific test phase on the diagram, the developed code is subjected to rigorous tests through unit tests, which ensure conformity to the design. Code quality is ensured through code review practices, continuous integration and static code analysis tools.

- **Phase 5: Unit Testing (Testing of Individual Software Modules)**.

  Once the code is implemented, specific tests are performed on each software unit to verify correctness and absence of bugs. Static analysis tools, like Polyspace, are used to identify potential programming errors, buffer overflows, and memory access issues. Unit tests are crucial to identify and correct defects before software is integrated with the rest of the system. This approach reduces the risk of error propagation in subsequent phases.

Connection to Phase 4: IUnit tests verify that each individual module, as defined in the design documentation, performs its functions correctly. These isolated tests help to identify errors at the level of a single function or method, reducing the risk of spreading bugs in later stages.

- **Phase 6: Integration Testing (Integration Testing Between Software and Hardware Modules)**.

  After unit testing, the interaction between software and hardware modules is verified. This phase is crucial to ensure that communications between ECUs, sensors, and actuators occur correctly and that the system functions as an integrated unit. Techniques like Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) are used to verify software behavior on simulation platforms and real hardware, reducing the risk of problems related to incompatibilities or communication latencies.

  Connection to Phase 3: Integration testing focuses on the interaction between modules. After individual modules have passed unit tests, they are combined to verify that interfaces and communications work as intended. This step is crucial to identify interfacing problems or incompatibilities that may not be evident at the unit level.

- **Phase 7: System Testing (Testing of the Complete System in a Simulated or Real Environment)**.

  Here, the complete system is tested in its operating environment, evaluating performance, reliability, and robustness. Test benches and Hardware-in-the-Loop (HIL) platforms are used to simulate vehicle behavior without needing a physical prototype. This phase includes functional tests, safety checks, and simulations in realistic scenarios to identify any anomalies before production.

  Connection to Phase 2: The system as a whole is tested at this stage. The test system is run in an environment that simulates the real operating environment, verifying that all functionality, performance and security requirements are met. The integration between software, hardware and other external components is also checked to ensure that the system works harmoniously.

- **Phase 8: Acceptance Testing (Final Validation and Production Release)**.

  The final phase involves testing on real prototypes or in advanced simulated operating conditions, such as Vehicle-in-the-Loop (VIL) Testing, which combines virtual simulations and track tests to validate autonomous driving and ADAS functions. If the system successfully passes these tests, it is ready for production and integration into production vehicles.

  Connection to Phase 1:

  This is the final stage of the V-Model and is performed directly by users or customer representatives. Tests based on the use cases and real-life scenarios of the system are carried out here. The objective is to confirm that the final product fully meets the initial requirements and is ready for deployment in a production environment.

The X-in-the-Loop (XIL) approach introduces a set of progressive validation techniques that allow testing of a system's software and hardware before its final implementation. Each method is applied at a specific stage of the V-model to ensure that any error can be detected and promptly corrected.

1. Model-in-the-Loop (MIL).

   This is the first level for the verification, in which the control of the system is tested in a completely simulated environment, without real hardware. The aim is to verify the correctness of the mathematical model from a logical and functional point of view. The model is run within the simulation tools such as Simulink, allowing the analysis of the system's behavior in different scenarios before to development of the actual software yet.

2. Software-in-the-Loop (SIL).

   With the SIL the mathematical model is translated into an executable code (automatically generated or manually written), replacing the mathematical model with an executable implementation. The purpose is to verify that the model is consistent with the MIL model. The software is run in a simulated environment on a host pc, maintaining the remaining part of the system in

a simulation environment. The results are compared with those obtained in MIL, this is known as the Back-to-Back Testing.

3. Processor-in-the-Loop (PIL).

   In the PIL test the plant is co-simulated with the code of the controller to analyze its actual performances. The aim is to validate the code in a realistic environment, verify execution times, memory consumptions, and compatibility with the target hardware. The software is loaded on the actual microcontroller or processor, while the rest of the system is still simulated. during this phases are used Embedded Coder to generate optimized code.

4. Hardware-in-the-Loop (HIL).

   This is a critical step in the validation process, as it introduces real hardware into the simulation loop. The purpose is to test the software in realistic operating conditions, verifying teh correct interaction with real sensors, actuators, and ECUs. The software runs on embedded hardware (such an ECU) while the surrounding environment (sensors, mechanical load, driving conditions) is simulated in real time through a HIL platform like dSPACE. HIL allows for the testing of the behavior of the system without a complete physical prototype. Reduces the risks of vehicle testing, allowing you to simulate critical conditions without danger.

5. Vehicle-in-the-Loop (VIL).

   The VIL approach extends the concept of HIL by including the real vehicle in the test process, but with controlled and simulated driving scenarios. The vehicle is equipped with a system that injects synthetic data (e.g. simulated traffic, variable weather conditions) while the ECU software processes the information as if it came from real sensors. This allows repeating tests in identical scenarios to validate vehicle behavior and reduces costs compared to real road tests.

1.3 Overview of Hardware-in-the-Loop (HIL) and Software-in-the-loop (SIL) test benches for electric work vehicles

Hardware in the Loop testing (HIL) is a critical tool to develop and validate complex control systems for electric work vehicles, particularly in the non-road mobile machinery (NRMM) area. Unlike passenger vehicles, electric work vehicles,including construction machinery, agricultural equipment, and heavy-duty utility vehicles operate under demanding, variable environmental conditions and carry unique performance and safety requirements. HIL test benches permit one to simulate and rigorously evaluate these operational scenarios in real time, creating a controlled environment in which virtual models of real environment are combined dynamically with actual hardware components.

In HIL test benches for NRMM applications, physical control units (such as motor controllers, hydraulic actuators, and battery management systems) are connected to real-time simulators that mimic the demanding operational conditions specific to NRMM. For example, load simulators and powertrain models replicate the mechanical stresses and duty cycles these vehicles experience in real-world conditions, such as heavy lifting, uneven terrain, or continuous operation in extreme climates. This approach allows to verify the performance of critical systems, such as power distribution, under controlled conditions, including factors such as thermal stability, power consumption, and fault tolerance. Moreover, by simulating various load conditions, HIL test benches permit to have details about energy efficiency, reliability, and safety of NRMM control systems before these vehicles are deployed in field operations.

Real-time communication protocols, such as CAN-BUS are necessary to provide data exchange between the simulated environment and physical components. Real-time communication ensures that HIL simulations can accurately replicate rapid changes in vehicle states, such as load shifts, motor responses, or braking force adjustments. This precise synchronization between the physical sensors and the simulation model is essential for the accurate testing of advanced control algorithms, such as traction control, antilock braking, and power management

strategies.

1.4 Thesis structure

This thesis work is organized in six chapters, each focused on specific aspect of the research, gradually expanded to build up the objectives laid out in section 1.1. The following is an overview of the next chapters and their key components.

- Chapter 2: SIL and HIL Test benches for electric work vehicles. This chapter explores the architectural design of HIL test bench, and the essential elements for the HIL tests, highlighting how their modular framework supports the testing of electric and hybrid powertrain in simulated environments. Literature case studies provide practical insights, such as tests oh hybrid tractor [1] and multimotor drivetrain evaluations [2].

- Chapter 3: CAN-Bus Communication in the test bench. The second chapter of the thesis explores the role of the CAN-Bus protocol to have real-time data transmission within the test bench environment.

- Chapter 4: Design and development of the GUI. This chapter discusses the design of the graphical user interface (GUI) using MATLAB App designer.

- Chapter 5: Validation of the GUI. In this chapter, it is explained how the SIL test has been performed to validate the user interface. All the steps with the model configuration are evaluated with numerical data for the performance evaluation of the software.

- Chapter 6: Conclusion and future work. The last chapter of the thesis resumes all the objectives achieved, emphasizing improvement in real-time communication and data consistency. Suggestions are presented for future research, such as expanding to the HIL setup to incorporate more complex configurations.

Chapter 2

Software-in-the-Loop and Hardware-in-the-Loop Test Bench for Electric Vehicles

In the development of complex systems, a crucial issue is the need for efficient, accurate, and safe methods to validate control systems and embedded software. Hardware-in-the-Loop (HIL) and Software-in-the-Loop (SIL) systems are becoming essential techniques in order to validate and verify system performance in different phases of system development. They provide powerful tools for the simulation, validation, and testing of control systems across various domains, including electric vehicle (EV) powertrains, hybrid tractors, and complex Non-Road Mobile Machinery (NRMM) applications. This chapter explores the main principles, structures, and applications of HIL and SIL systems, in particular how these are applied in a test bench environment.

2.1 Concepts of SIL and HIL systems

SIL testing consists of a virtual model to replicate the system to be analyzed, executing the system's control algorithms and software in a simulated setting with the aim of evaluating and verifying software functions without requiring physical

devices.

It is a simulation-based technique, this means that the system's software interact with a virtual model of the plant, allowing to ensure that the software behaves as expected in different scenarios, including fault condition. With this technique, it is possible to identify and correct issues in the control logic or algorithms at the early stage of the development cycle. The main advantages for this testing technique are: high flexibility, allowing modifications to the simulated environment as needed; low-risk procedure, as testing is performed in a controlled virtual environment; cost-effective testing, as it does not have the need for hardware prototypes. For example, SIL has been widely used in EV powertrain modeling [18] and hybrid electric cooling systems [3] to ensure software robustness in scenarios such as regenerative braking and variations in thermal load.

On the other hand, HIL testing integrates real hardware components with a virtual simulation of the environment. Unlike SIL, HIL includes physical hardware in the test loop, bridging the gap between simulation and real-world deployment. It allows a real-time interaction: physical hardware (e.g., controllers, sensors, actuators) interfaces with a real-time simulation of the environment, ensuring realistic system responses.
HIL tests are essential to test the interaction between control software and actual hardware components. The main advantages are: realistic testing conditions, as the physical hardware interacts with simulated real-world inputs; reduced development time by allowing parallel development of hardware and software; enhanced confidence in system reliability and performance before deployment.

Modern development workflows often employ both SIL and HIL methodologies in a complementary manner. SIL is typically used in the early stages for initial validation, while HIL is used later to verify hardware-software interactions. This combined approach ensures comprehensive testing, reduces development risks, and shortens time-to-market.

The Figure 2.1 shows the topology of Hardware in the Loop. When the plant model is deployed on a HIL simulator, the controller acts as if it is controlling real hardware, when in fact it is connected to a simulator. The most important challenge in developing plant models for HIL systems is that they must run in real-time and have constraints including accurate representation of the plant physics,

controller rates, and HIL hardware computational power.

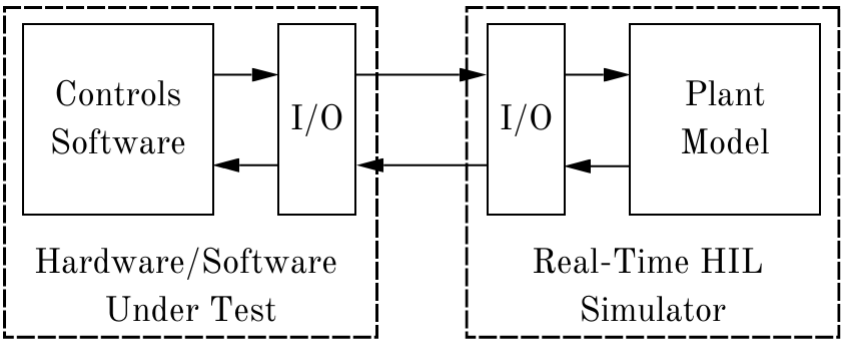

Figure 2.1. Hardware in the Loop Topology.

2.2 Architecture and configuration of HIL test bench for non-road electric vehicles

In the field of powertrain testing, various types of test benches are employed to meet different experimental needs and simulation conditions. Test benches generally fall into two main categories: Dissipative Power Test Benches and Power Recirculation Test Benches. Dissipative Power Benches are designed to inject energy into the system through an electric motor. The energy is transmitted as torque along the motor shaft, often passing through a mechanical transmission that links the motor to the component under test. In cases where the system under evaluation produces energy (such as an internal combustion engine, a whole vehicle, or even the electric motor itself) the motor becomes the energy supplier. A key element of dissipative benches is the energy-dissipating system, typically implemented using a magnetic brake. This brake not only measures the output energy (or the input energy, depending on the test configuration) but also quantifies the power dissipated by the transmission, providing critical data on system performance.

Power recirculation benches are designed with energy efficiency in mind, enabling a significant portion of the energy to be recycled back into the system. They

can be subdivided into two types: Mechanical Power Recirculation Benches and Electrical Power Recirculation Benches. Mechanical Power Recirculation Benches are widely used in industrial settings, these benches are ideal for testing gear transmissions, shafts, couplings, and bearings. The typical setup includes an electric motor to provide the initial motion and two reducers arranged in series to create a closed-loop energy circuit. This configuration allows the motor to supply only the minimal torque needed to overcome friction and initiate motion, with external power consumption representing just about 4–5 percent of the total circulating power. The downside is that the reliance on reducers can impose a certain rigidity in terms of system flexibility and application.

Electrical Power Recirculation Benches start with a similar configuration to dissipative setups—an electric motor connected to a mechanical transmission—but diverge by incorporating an electrical current generator at the output. Instead of dissipating the energy, the generator converts the mechanical torque back into electrical energy. This energy is then fed back into the battery system, which in turn supplies the power needed at the input. The primary advantage here is the minimal external energy requirement; the system only draws additional energy to compensate for losses incurred during the test. The trade-off, however, is a more complex control logic, which is essential for accurately managing the energy flow and ensuring precise performance measurements, especially when testing electric motors.

The test bench used to develop this study is configured as a power recirculation bench: the electric brake, mounted on the fixed part of the bench, does not dissipate power. It converts the kinetic energy into electrical energy and redirects it into the battery system. These batteries then provide the necessary electrical power to the electrified portion of the powertrain under test. This creates an energy loop, where the batteries provide only the net amount of energy that is dissipated during the bench tests, as schematized in Figure 2.2.

This modular and versatile configuration not only enhances energy efficiency but also offers high adaptability, allowing the bench to be reconfigured for testing various components or entire systems. Such a design is particularly valuable in modern research settings where precision and sustainability are paramount.

Test benches are designed to reproduce as accurately as possible the working

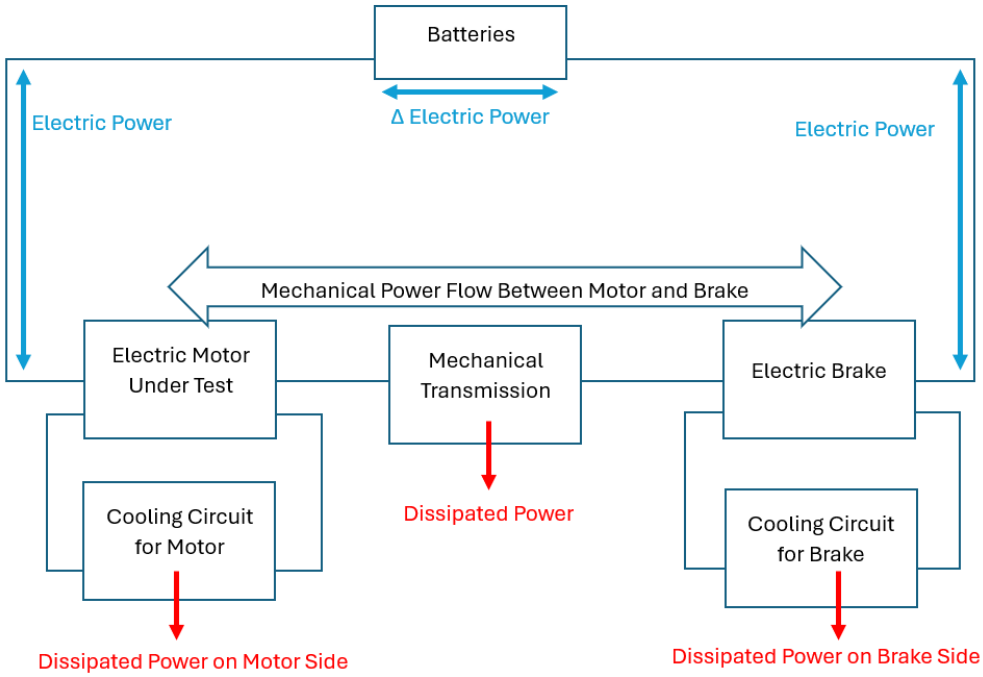

Figure 2.2. Block diagram of the test bench which gave rise to numerical insights at Ecothea Srl.

conditions to which the products to be tested will then be subjected. This type of tests is classified as Hardware in the Loop test. The company Ecothea developed a full scale (1:1) test bench to reproduce the most realistic scenarios in which hybrid and electric powertrains of the vehicles analyzed, in this case NRMM, performing HIL tests in which the physical hardware (that will be mounted in the real vehicle) is tested within a software simulation loop of the electronic control system.

Figure 2.3 shows how the test bench perfectly meets the requirement of a HIL simulation bench. This integrated HiL approach, which combines physical hardware (red section), a virtual simulation environment (orange section), and a physical load simulation (blue section), enables the testing of drivetrain components in a wide range of simulated operating scenarios without the need for a complete vehicle prototype. Within the HiL test bench, the Onboard Control Unit (HCU) interfaces with the Control Software running on a PC. This software simulates vehicle dynamics and generates the desired torque and speed set-points, which are sent to the HCU. The HCU then sends these commands to the motor drive and the

brake drive. Specifically, the motor drive actuates the electric motor (representing the physical hardware), while the brake drive -part of the Physical Load Simulation (blue section) controls the electric brake to emulate external load conditions.

At the same time, the mechanical transmission (including joints and gearboxes) transmits the torque and speed generated, and the torsimeter continuously monitors these parameters in real time. This feedback allows the HCU to fine-tune its commands, maintaining precise control over the system.

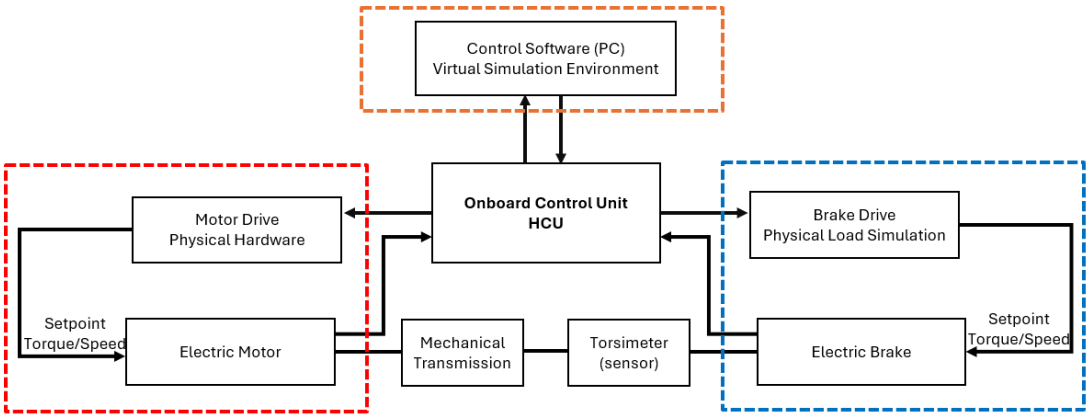

Figure 2.3. Hardware in the Loop of the Real Test Bench.

2.3 Simulink Test Bench model for SIL test

The implementation of a SIL system, as said before, allows to test and validate control software without the need for physical hardware, reducing costs and improving the reliability of the simulations. The design of the SIL system is then followed by the modelling of the test bench, which is a controlled environment for testing the efficiency of electric motors nd transmissions in future HIL configurations.

Software-in-the-Loop (SIL) is a simulation approach that allows control software to run in a virtual environment, testing the system response without using real hardware. In MATLAB/Simulink, a SIL system is based on a model of the physical plant, representing the electric motor, transmission and other components; a PI controller to optimize the system response, and a simulation loop, in which

the controller interacts with the plant model and provides data for performance analysis. The Figure 2.4 shows a block diagram representation of the simplified system, that is coherent with the real configuration showed in the previous section, in Figure 2.2.

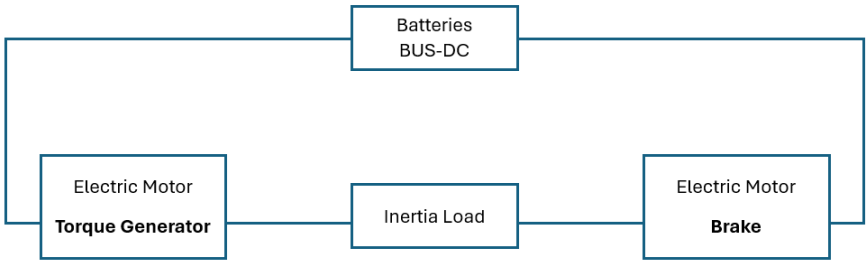

Figure 2.4. Block Scheme of Simplified Model of the Test Bench.

Since the main purpose of this thesis work is to analyze the aspects related to data transmission, a simplified model of the test bench has been developed in Simulink with the Simscape library. The model is the virtual representation of the real test bench and it includes: two electric motors acting as motor and brake, an inertia to model the rotational dynamics of the system and an electrical part composed of a DC BUS that supplies the two servomotors in common. To obtain a dynamic response that is as close as possible to the real plant it would be useful to integrate advanced models to replicate the behaviour of electric motor in different operating conditions as illustrated in [4].

The simplified model of the test bench developed in Simulink environment used for this thesis work is represented in Figure 2.5. This model replicates a system where an electric motor generates torque to rotate an inertia, the second motor acts as a brake, absorbing energy from the system. Both motors are powered by a battery (DC Bus), and two PI controller regulate the system's angular velocity that is then measured and converted into a Simulink signal for output visualization.

Looking at the Figure 2.5, it is possible to see three different parts of the model:

- The Green lines represent the Mechanical Circuit, developed with Simscape Driveline. The motor generates torque and drives the system. The brake

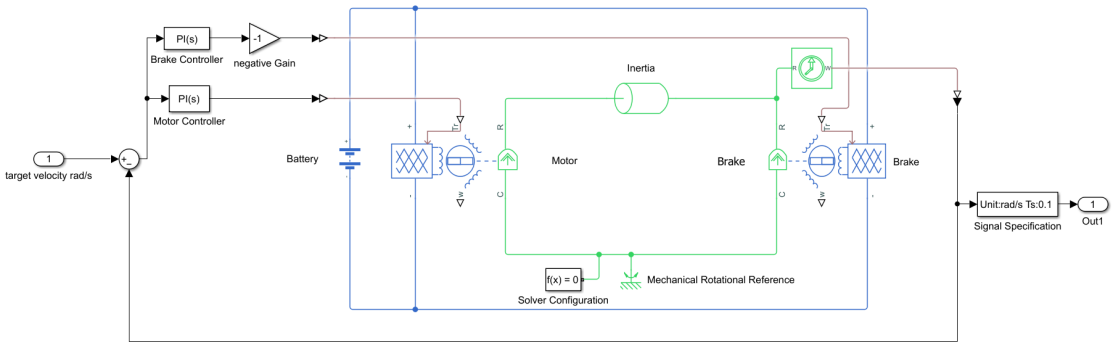

Figure 2.5. Simplified Simulink model of the test bench.

applies a negative torque opposite to the motor. The inertia models the rotating mass connected to the shaft. The Rotational motion sensor is necessary to measure the actual value of the angular velocity and send it back to PI controllers for feedback regulation.

- The Blue lines represent the Electrical Circuit, developed with Simscape Electrical. A Battery provides DC power for both the motor and brake, as illustrated in the block diagram in Figure 2.4.

- The Brown lines represent the System Control, which interface Simscape with Simulink through PS-Simulink Converters. PI Controllers (Motor Controller and Brake Controller) regulate the motor and brake operation. The brake PI controller regulates the brake to oppose motion, a negative gain (-1) inverts the brake signal, applying opposite torque. The Reference signal (Target velocity rad/s) defines the desired speed.

- The blocks Simulink-PS and PS-Simulink are used to convert signals between Simulink (discrete domain) to Physical Signals (PS) for Simscape and vicev-ersa.

The system is powered by a DC battery that supplies electrical energy to both

motors. The first electric motor converts this energy into mechanical power, generating torque to rotate the inertia. The inertia models the resistance of the test bench to changes in angular velocity acting as a flywheel, storing and releasing kinetic energy as the system accelerates and brakes. When the brake motor is activated, it applies a resistive torque opposite to the rotation. Instead of simply dissipating the energy as heat (as in a conventional friction brake), for the actual configuration of the test bench, the brake motor acts also as a generator, converting mechanical energy back into electrical energy. This process is known as regenerative braking.

During regenerative braking, the electrical energy generated by the brake is sent back to the battery through the electrical circuit. This feedback mechanism allows the system to recover part of the energy instead of losing it as heat, improving overall efficiency. The direction of current flow reverses, and depending on the control logic, this recovered energy can be stored or used to power other components. To better highlight this aspect, for future improvement of the model, it will be good to add a current measurement system to monitor power consumption.

The controller logic used for this model is simple: the input sets the desired angular velocity, the value is compared to the actual velocity in a sum block to calculate the error. Two PI controllers are used, one for motor control, and the one for brake control. The error signal is fed into PI controllers to compute the necessary actuation. For this purpose PI controllers have been used because the model is really simple, so a control on instantaneous error (Proportional term) and a correction for steady state deviations (Integral term) is sufficient to ensure smooth and stable velocity tracking. The output of the brake PI controller is passed through a negative gain to ensure that brake applies a negative torque, opposing motion. It is also possible to use a PID controller to optimize the stability of the system, improving the system time response, as demonstrated in [19].

To test the stability of the controller, the main strategies are:

- Step response: Analysis of the engine response to sudden inputs.
- Load variation: Simulation of variable load conditions to verify that the controller is effective.
- Fault injection: Failure condition test, such as low battery, over-temperature.

To evaluate the dynamic performance of the model, a step response test was made. The test involved applying a step input signal of 120 rad/s, that represents the target speed, then the system response has been evaluated. The Figure 2.6 shows the input step signal and the system response. From the simulation, an inertia value of 0.5 $Kg*m^2$ has been selected. This value is reasonable for a heavy electric vehicle test bench, as it approximates the rotational inertia of typical powertrain components under testing conditions. By tuning the Proportional and the Integral terms of the controllers related to the motor and to the brake, the results indicate that the system exhibits a good dynamic behavior.

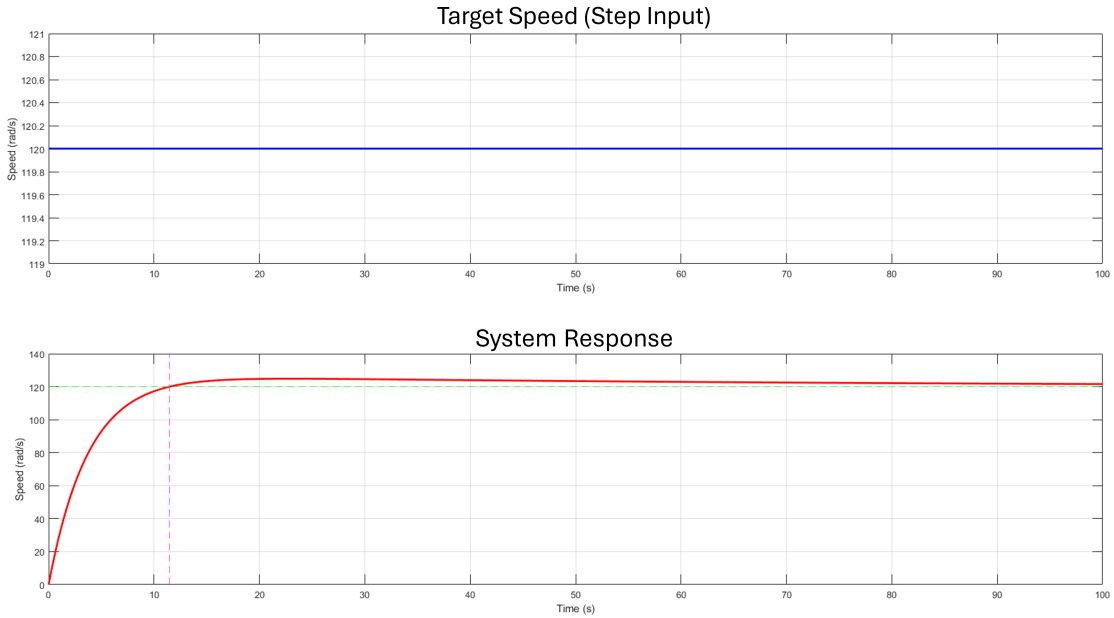

Figure 2.6. Step response of the Simulink model.

The overshoot is below 5%, demonstrating the system's ability to handle sudden changes without excessive oscillations. Moreover, the rise time was measured at 11.47 s, and the settling time at 5% was 8.8 s, indicating the system's rapid stabilization after the step input. These values confirm the model's stability and responsiveness to sudden inputs, aligning with the design objectives for the test bench.

To assess the system's robustness, additional tests were conducted by varying

the input signal at different time intervals. In the first scenario, 2.7 ,the input signal was changed every 60 s while in the second case, 2.8 the variations occurred every 30 s . The system demonstrated consistent and reliable behavior in both cases, responding promptly to each change in the input without any instability.

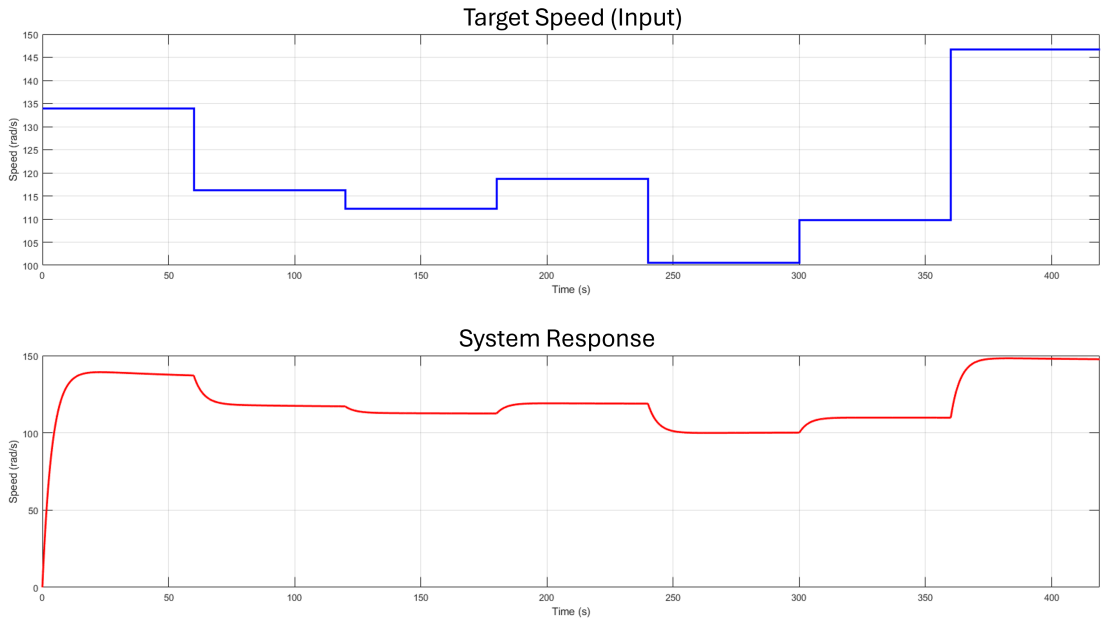

Figure 2.7. System response to input changes every 60 s.

Regardless of the interval, the system maintained its performance, characterized by minimal overshoot, fast rise times, and stable settling behavior.

Another test that has been done to assess the system's robustness: varying load conditions, conducted with different inertia values. For small variations in load inertia, ranging between 0.4 $Kg * m^2$ and 0.6 $Kg * m^2$, the system response was not affected. The controller effectively compensated for these changes, maintaining stable behavior with minimal impact on settling time or overshoot. When the load inertia was increased to higher values, the system exhibited a noticeable change in dynamics. For example, with an inertia value of 1.5 $Kg * m^2$, the settlement time extended to approximately 50 s. This indicates that while the controller requires more time to stabilize the system under heavier loads, it successfully achieves a

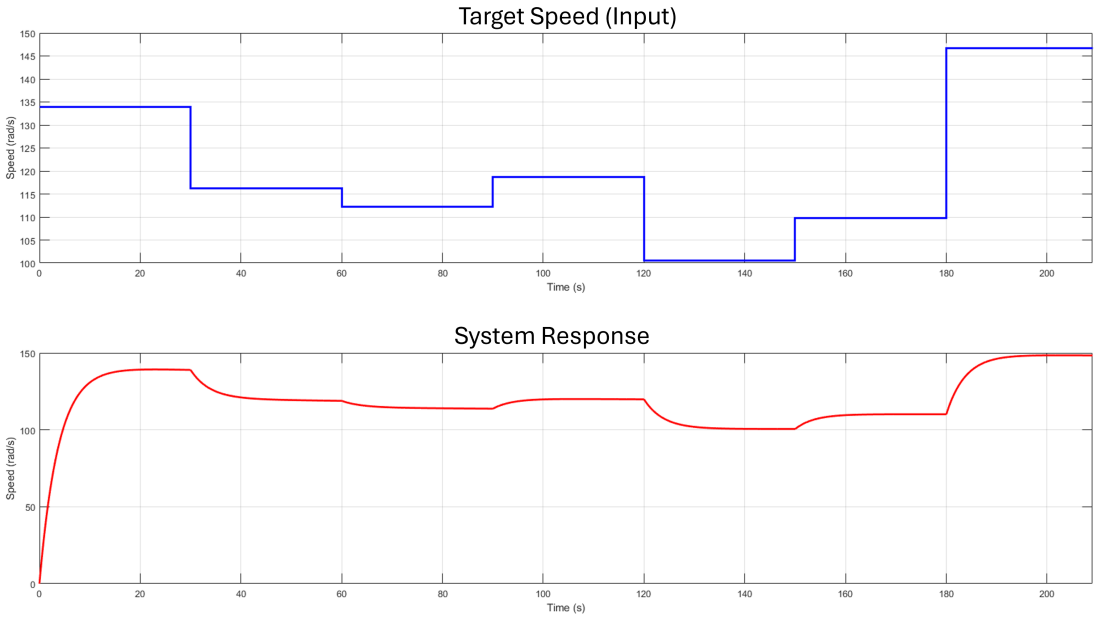

Figure 2.8. System response to input changes every 30 s.

stable final value without instability or excessive oscillations. These results confirm that the controller is capable of managing a wide range of load conditions, ensuring stability and adaptability even under challenging scenarios.

It is interesting, also, to focus on the last point, fault injection in control systems. This is a fundamental strategy in control system validation to assess the robustness of the system under non-normal operating conditions. This approach involves introducing faults or anomalies into the system to observe its behavior, detect potential failures, and validate safety mechanisms. Common fault injection scenarios include power supply fluctuations, sensor failures, overheating conditions, and communication problems. These tests are particularly relevant in Hardware-in-the-Loop (HIL) and Software-in-the-Loop (SIL) simulations to ensure that the system can handle real-world uncertainties.

From a general point of view there are different types of Fault Injection Tests. Power supply faults are made simulating a gradual voltage drop, this helps evaluate whether the system can maintain operation with limited energy.

If the system also involves a thermal part, it would be useful to test thermal

faults, such as overheating and thermal protection. Increasing the simulated temperature of the power electronics or motors beyond safe thresholds ensures that the system properly engages cooling mechanisms or reduces power output.

It is also important to test the behavior of the system when thermal sensors fail. For example, a speed sensor failure can lead to incorrect velocity estimation, affecting the control loop. Introducing random noise or drift in sensor signals assesses how well the system filters or compensates for erroneous readings.

Another main issue is the injection of errors into the communication. The system used for this work rely on Controller Area Network (CAN) communication for real-time data exchange between components. Faults in CAN communication can lead to severe performance degradation or complete system failure.

In the following list, some key fault scenarios are reported, including the ones that have been tested, and the ones needed for future development. It includes:

- Message Loss or Bus Off Condition.

  One of the most critical scenarios is the loss of communication between nodes, that can occur due to message loss or a bus off condition. It is useful to simulate a loss of communication by disabling or delaying CAN messages from critical nodes. This is useful to test how the system reacts when a motor controller loses its command input due to communication failure. When critical messages, such as speed commands, were lost, the motor could either stop unexpectedly or remain stuck at incorrect values. The solution adopted in this work is that the the system save the last received value to avoid sudden stops for the motor. A possible solution to be implemented when the physical system will be included for HIL testing is the following: if a message is not received within a set time, the system enters limp mode with reduced torque output so that the motor maintains predictable behavior even in case of message loss, preventing sudden stops. Regarding the Bus off condition, this is managed by a control strategy in the software: it is important to guarantee that for any new simulation the bus is not affected by data from previous simulations, avoiding that possible errors are accumulated by making the Bus off.

- High Bus Load and Message Collision.

Another potential failure mode occurs when the CAN bus experiences excessive traffic, leading to message delays or even collisions. A high bus load can negatively impact motor control by delaying critical messages such as torque or braking commands. This test consists in injecting a high volume of messages on the CAN bus to simulate bus congestion. An excessive number of CAN messages caused command delays, negatively affecting motor control. For this thesis work also with a lot of messages the correctness of message flow is guaranteed through a good messages management in the developed software. In some real situations, with a real HCU, a solution to avoid problems in case of high bus load is to optimize the CAN ID priority: critical messages, such as torque and braking commands, can be assigned the highest priority IDs to ensure their transmission even under heavy bus load.

- Corrupted Messages and Bit Errors.

  Communication errors, such as bit errors and corrupted messages, can also compromise system reliability. These errors may be introduced by external interference, hardware malfunctions, or software bugs. To be sure that the system is reliable, is important to evaluate if redundant communication mechanisms (e.g., retransmissions, error detection codes) work as expected. This can be an additive test to be done for future developments of this work. The expected behavior is that if a critical value changes unexpectedly, the system discards it and waits for the next valid transmission.

- Node Failure and Silent Node Simulation. A particularly dangerous scenario arises when a critical ECU stops responding (silent node). If the motor controller or battery management system (BMS) becomes unresponsive, the entire powertrain could be compromised, leading to unsafe conditions. To address this, a fail-safe mechanism should be implemented. If the system detects that a critical node is no longer transmitting data, predefined safe fallback values will be applied. In extreme cases, the motor may be automatically disabled to prevent potential hazards. Additionally, recovery strategies should be evaluated to determine how the system behaves when the failed node comes back online after a temporary disconnection. This test is necessary for the test with real hardware.

In this study, two fault injection tests were conducted to evaluate the robustness of the CAN communication system and its ability to handle critical failures. These tests simulate real-world problems that could compromise the correct functioning of the control system and allow the development of effective countermeasures.

Chapter 3

CAN-Bus Communication protocols in the Test Bench

3.1 Introduction to the CAN-Bus protocol

The Controller Area Network (CAN-Bus) protocol, standardized as ISO 11898, was developed by Bosch in the 1980s to provide reliable high-speed communication for automotive applications. This protocol enables real-time data transmission between electronic control units (ECUs) in vehicles, which is essential for controlling and monitoring critical subsystems such as engine management, braking, and safety systems.

Regarding the physical layer, it is made of two simple wire bus: CAN-H and CAN-L, terminated with a resistor of 120 ohm as shown in Figure 3.1. The resistors are placed at both ends of the CAN bus to prevent signal reflections; without termination signals would bounce back and forth, leading to bit errors. Nodes are connected to the main CAN bus through short stub lines, which should be as short as possible to keep the signal intact, avoiding communication errors. The main reason a high-speed CAN bus consists of two wires (instead of one) is related to noise immunity: two wires allow a differential transmission; this means that information is encoded in the difference between voltages. Differential transmission is unaffected by common-mode noise because it affects both wires in the same way, balancing each other out. The CAN Bus uses two wires to transmit information,

CAN-H (high) and CAN-L (low), corresponding, respectively, to recessive and dominant states. The logic function used from the CAN bus to the Wired-AND function that uses logical level 1 for recessive and 0 for dominant. The Figure 3.2 shows the truth table of the Bus logic.

To better understand how it works the Figure3.3 represents data on the physical layer using differential voltage signaling. The picture shows the CAN_H (red) and CAN_L (green) signals, their voltage levels, and how they correspond to logical bit values. The first row represents the actual bit sequence that is actual transmitting. Below, the CAN_H and CAN_L signals illustrate the voltage variations for dominant and recessive states. The last waveform represents the driver logic, showing when the CAN transceiver is actively driving in a dominant state (low signal) and when the driver is not activated, the bus returns to the recessive state due to the pull-up and pull-down resistors.

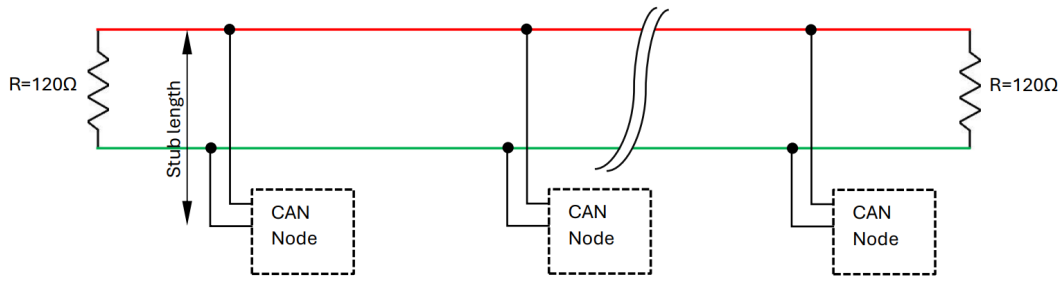

Figure 3.1. High-speed CAN bus. ISO 11898-2 Network.

As said before, information are transmitted based on the difference between the high signal (CAN-H) and the low signal (CAN-L), both signals with an amplitude of 1V are centered in 2.5V, in such way the only possible results of the difference are 0V or 2V. The logic is shown in Figure 3.4. To guarantee synchronization, instead of using a clock (that requires additional wires) it is used the bit stuffing to understand where starts and ends the bit sequence that is being transmitted.

Bits are encapsulated in pre-defined messages, called CAN frames. The data frame objective is to send data over the network in the correct way. Referring to Figure 3.5 each frame contains:

Node 1	Node 2	Node 3	BUS
D	D	D	D
D	D	r	D
D	r	D	D
D	r	r	D
r	D	D	D
r	D	r	D
r	r	D	D
r	r	r	r

Bus in dominant state

Bus in recessive state or idle

Figure 3.2. CAN Bus truth table. The two states are: r for recessive (1) and D for dominant (0) .

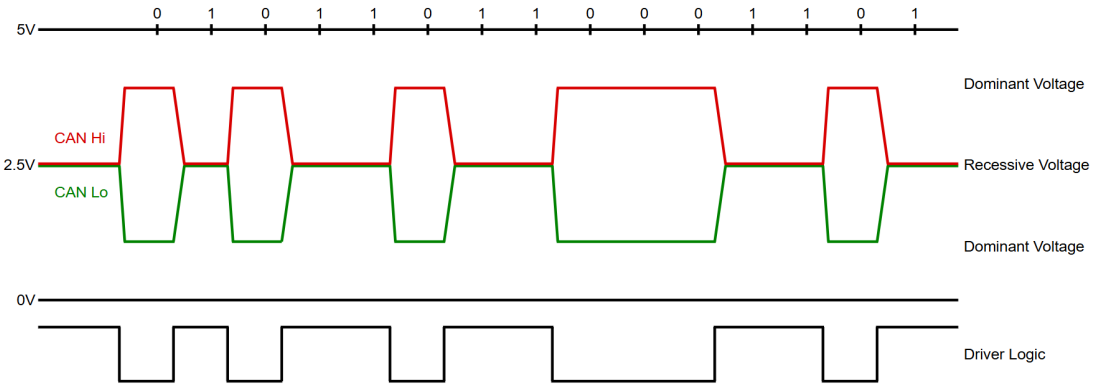

Figure 3.3. CAN Signaling.

- Start of Frame (SOF): consisting of a recessive-to dominant transition (CAN-H=3.5V and CAN-L=1.25) from the idle bus status (CAN-H=CAN-L=2.5V).
- Arbitration Field: this field set message priority and contains:
  - ID: determines message priority: lower ID values have higher priority. In extended CAN format, a 29-bit identifier can be used instead of the standard 11-bit identifier, providing more unique IDs for complex networks.
  - RTR Remote Transmission Request: it is used to distinguish between data frame (actual data transmission, dominant 0) and remote frame

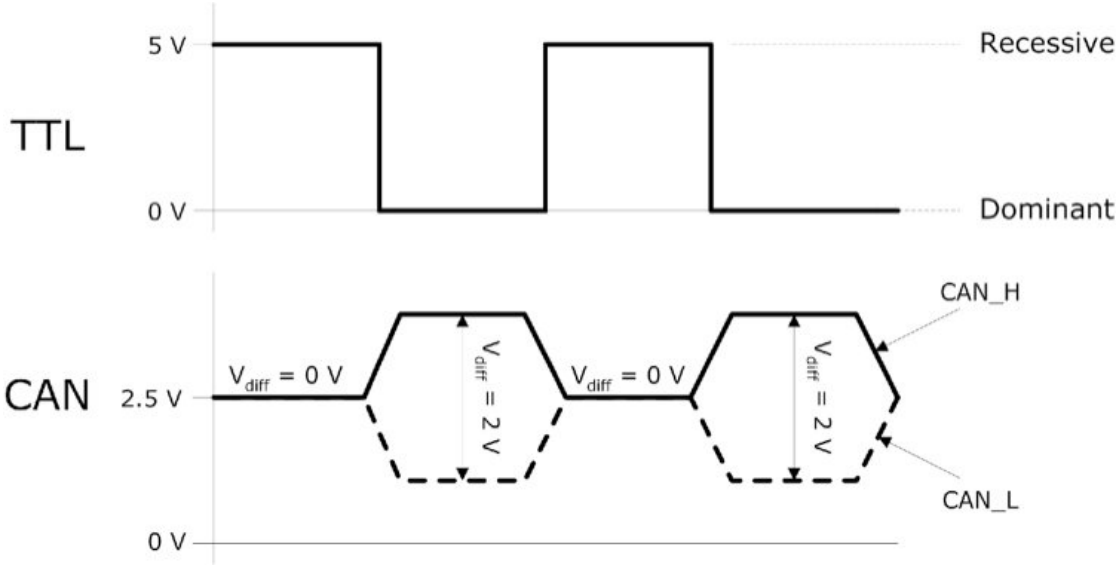

Figure 3.4. CAN logical BUS states and transistor-transistor logic.

(request for data from another node, recessive 1)

– r0 Reserved Bit: must be dominant and is reserved for future protocol extensions.

- Control Field: this field manages frame length and control information.
  - DLC Data Length Code that specifies the number of data bytes in the DATA field.
- Data field: contains actual payload data, such as sensor values or commands. The number of bytes is set by DLC field.
- CRC Cyclic Redundancy Check: it is used to detect errors in the transmission.
- ACK Acknowledgment Field: this field confirms successful data reception.
- EOF End of Frame: all the bits must be recessive (1) and it marks the end of a valid CAN message, ensuring that the bus remains idle before a new transmission.

It is interesting to highlight error and fault handling in CAN-Bus. The CAN protocol features advanced error management mechanisms. Each node maintains

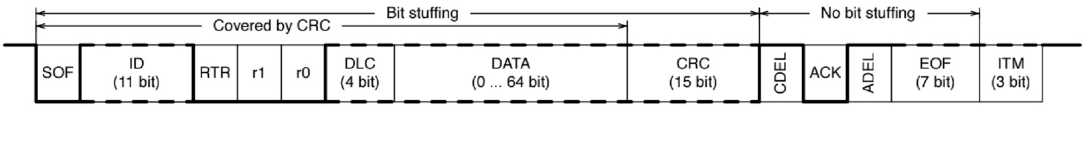

Figure 3.5. CAN frame.

a Transmit Error Counter (TEC) and a Receive Error Counter (REC) to track communication errors. The counters are incremented or decremented based on errors: a detected error increases the respective counter,a successful transmission reduces TED, a successful reception reduces REC. A node can be in three states based on its error counters:

- Error-Active: if the ration between TEC and REC is lower than 127 the nodes operates in normal conditions
- Error-Passive: if tec is higher than 127 or rec is lower than 277 the node still transmit but does not send active error flags to avoid further bus disturbance.
- Bus-off: if TEC exceeds 255 the node is disconnected from CAN network, meaning that it stops transmitting completely so the node is isolated the network to prevent further disruptions.

When a node is in Bus-off state it is necessary re-enable it: this requires manual intervention, or a recovery mechanism based on network logic. This behavior is particularly relevant in the context of this thesis, where improper message handling in the application code resulted in a Bus-Off condition due to TEC thresholds being surpassed. The communication system has been optimized once this issue has been identified and corrected, performing a software-based re-initialization.

3.2 Integration of CAN-Bus at different working levels.

Using CAN communication has different meaning depending on the development flow, which can be divided into two main parts: verification phases and validation

phases, as shown in the picture.
During the verification phases must be performed the network design and hardware design. This means that in this step is defined the number of CAN nodes, how long the network is going to be, what embedded hardware is needs (CPU).
The validation phases are usually done on a test bench and include the following:

- Test at node level: verify that each node communicates correctly.
- Test at network level: verify that all nodes communicate correctly.

From the software point of view the communication is based on coding where is defined how CAN transmission is managed (message-based or signal based) and how CAN reception is managed (polling or interrupt, unfiltered or filtered).
Regarding hardware considerations, to have better modularity and cost control, since higher performance means higher costs, CAN communication is usually based on two separate hardware components: the CAN protocol controller and CAN transceiver.
The first one is typically embedded in a MCU (microcontroller unit) and handles all the data-link layer aspects of the protocol, such as frame transfer and reception, arbitration and error handling. As shown in picture 3.6, the CAN protocol controller communicates with the MCU through a register-based interface, used a dedicated RAM to buffer incoming/outgoing messages and it interfaces also with the CAN trans receiver, that will be explained more in detail later on.

Since CAN is a broadcast protocol, meaning that each node receives any transmitted message, also those they do not deed, the RX matching avoiding overloading the processor with the unneeded CAN messages. The working principle is the processor tells the CAN protocol controller which are messages of interest, upon CAN message reception the protocol controller verifies if the message is relevant or not and only relevant messages are buffered and notified to the processor. The matching rules can be specified (for message ID or message payload).

The CAN transceiver is outside the MCU and manages all the physical layer aspects of the protocol. The same CAN controller supports several different physical layers. The CAN transceiver converts digital signals into differentials required by the CAN physical layer for transmission on the CAN bus.

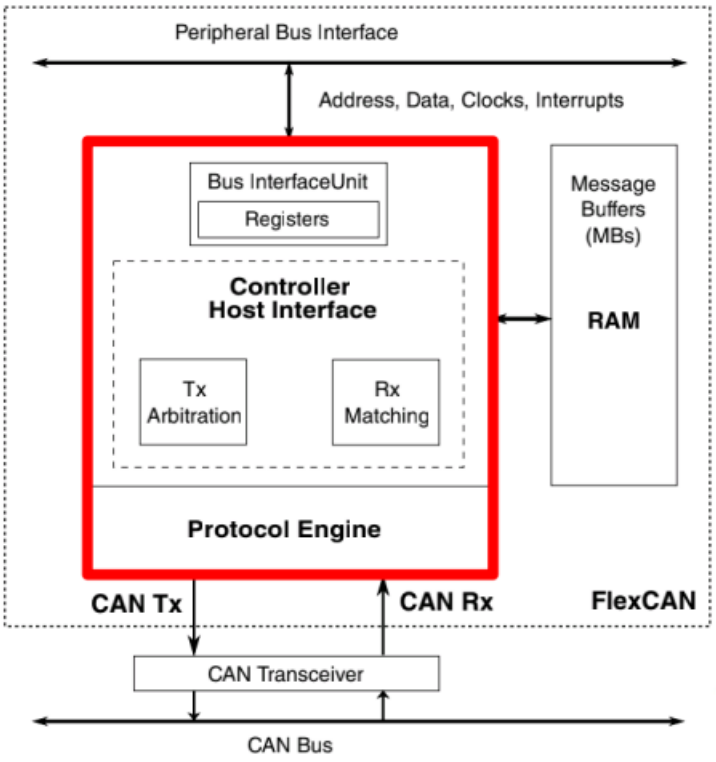

Figure 3.6. CAN protocol controller.

Regarding software considerations, it is normally organized in layers. To better understand the structure of CAN communication, it is useful to divide it into different abstraction levels. Each level has specific responsibilities and interacts with both hardware and software components.

The Physical Layer manages the electrical and signaling characteristics of the CAN bus. It ensures that the transmitted data is physically delivered to all nodes on the network.

The Data-Link Layer is responsible for organizing data into frames, managing bus access through arbitration, and handling error detection and correction mechanisms.

The Application Layer defines how messages are structured and processed by the software running on the microcontroller, implementing communication services needed by the user application.

The Figure 3.7 diagram illustrates these three levels and their corresponding hardware and software components. Now each level will be explained in detail starting from the lowest level.

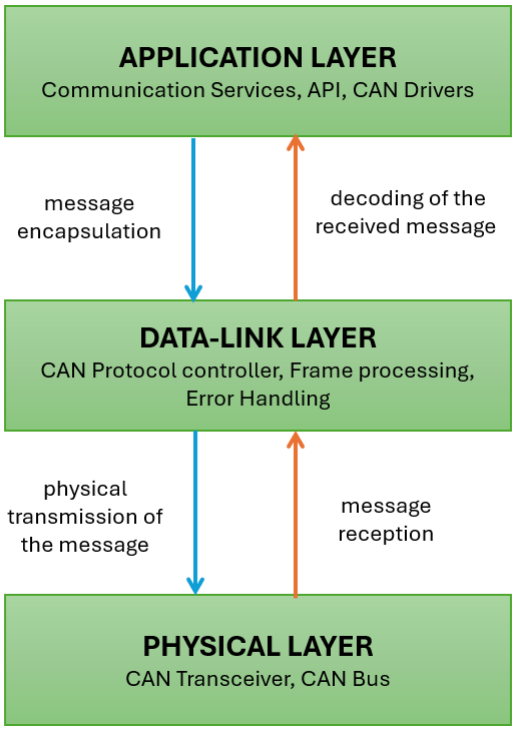

Figure 3.7. Block diagram of the CAN communication layers with the interaction between the Application, Data-Link, and Physical layers.

1. Physical Layer.

   The Physical Layer defines how CAN messages are transmitted over the bus in terms of voltage levels, bit timing, and signal encoding. The main components of this layer are:

   - CAN Transceiver: Converts digital signals from the microcontroller into differential signals required for CAN communication. It also filters and amplifies received signals before passing them to the protocol controller.
   - CAN Bus: The physical medium that connects different CAN nodes. It is typically a twisted-pair cable reducing electromagnetic interference and supporting high-speed communication.

2. Data-Link Layer.

   This layer ensures reliable communication between CAN nodes and manages message transmission and reception. The key components are:

   - CAN Protocol Controller: Handles low-level communication tasks such as message framing, arbitration, and error handling. It is typically embedded within the microcontroller (MCU) ensuring efficient data handling.
   - Frame Processing Unit: Buffers messages and applies acceptance filters to ensure that only relevant messages are passed to the processor.
   - Error Handling Mechanisms: Detects transmission errors and ensures message integrity through automatic retransmission or error flagging.

3. Application Layer.

   This layer defines how the software interacts with the CAN network. It provides APIs for message transmission and reception and structures data based on the application requirements. It includes:

   - Communication Services: Implements the interface for sending and receiving CAN messages. It supports two communication models:
     - Message-based communication where API is provided to send CAN frames given ID, data length and payload bytes.
     - Signal-based communication where API is provided to send higher-level objects, that represent application-related data. The set of signals and messages that application exchange is defined by the network architecture, building a CAN network database.
   - Driver Layer: Includes the Transceiver Driver (for CAN transceiver control) and the Protocol Controller Driver (for managing the CAN controller).

3.3 Implementation of the CAN-Bus communication within Simulink environment.

The CAN-Bus protocol has been used in the SIL test, to link the GUI with the Simulink model of the testbench. The integration of CAN communication ensures real-time bidirectional data flow for control and monitoring. The system configuration can be divided into 3 steps:

- Simulation Layer: The CAN Transmitter and Receiver blocks in Simulink are configured to send and receive data frames based on control signals and simulated feedback.
- Real-time Communication: The GUI interfaces with the Simulink model via a virtual CAN channel, replicating the behavior of physical nodes in a HIL environment.
- SIL test with real CAN peak: The GUI interfaces with the Simulink model via a real CAN network, with real nodes.

The first step in integrating CAN communication within the SIL test is the simulation layer. Here, it is established the communication between the Simulink model of the virtual model and the CAN blocks to allow the transmission and reception of CAN messages in a simulated setup. To do that, the MathWorks Vehicle Network Toolbox has been used. It is a powerful tool for simulating and managing CAN communication and other and other automotive network protocols such as CAN FD (Flexible Data-rate), J1939 and XCP. This toolbox provides a set of Simulink blocks, MATLAB functions and hardware interfaces allowing the simulation, the test and the development of real-time communication system with the test bench. The VNT provides a library of blocks to configure CAN communication. The Figure 3.8 shows the main blocks used:

- CAN Configuration Block: it is essential for configuring the communication parameters of the CAN bus. It defines aspects such as bus speed (e.g. 500 kbps), acknowledge mode (Normal, Silent) and selection of the hardware interface to be used (e.g. Vector, PEAK, MathWorks Virtual).

- CAN Pack/Unpack Blocks: These blocks are used to pack and decompose CAN messages. The CAN Pack block puts together input signals in a CAN message, with a specific ID. In the opposite way the CAN Unpack block extracts signals from a received CAN message. Input data can be chosen as raw data, manually specified signal and CANdb specified signal. For this thesis, the second kind of data has been used, since to use CANdb specified signal it is necessary to have a CAN dbc file. For future developments it can be interesting to realize this file, which structure will be explained later.
- CAN Transmit Block: Used to transmit a message over the CAN network. It is configured to send messages at a specified frequency, using the CAN bus and hardware interface configuration defined in the CAN Configuration block.
- CAN Receive Block: Allows to receive CAN messages in real time. This block is essential to collect data from CAN messages, which can then be post-processed. Also, for this blocks it is possible to set the sample time and if required a filter in the ID to be received.

As said before, a crucial aspect of CAN communication can be the use of a DBC file, which defines the message structure, signals, and their encoding. Each CAN message contains a unique ID and a set of signals. The signals within CAN messages could represent physical variables, such as speed or temperature. The encoding of signals, so how they are represented in the CAN message, is specified in the DBC file. This file could be directly imported in the CAN unpack block.

3.4 Simulink model for SIL testing.

To communicate with the on-board control unit, complete software commands are required in the current test bench configuration. In addition, the acquisition of the data during actual tests is done in a semi-manual way because the parameters to be extracted must be recorded manually after reading them from the software. For these reasons, it was then thought to create a Graphical Interface (GUI), which does not show the user the work done by the control software: this GUI will, in fact, present a series of buttons/sliders that allow the operator to decide how to

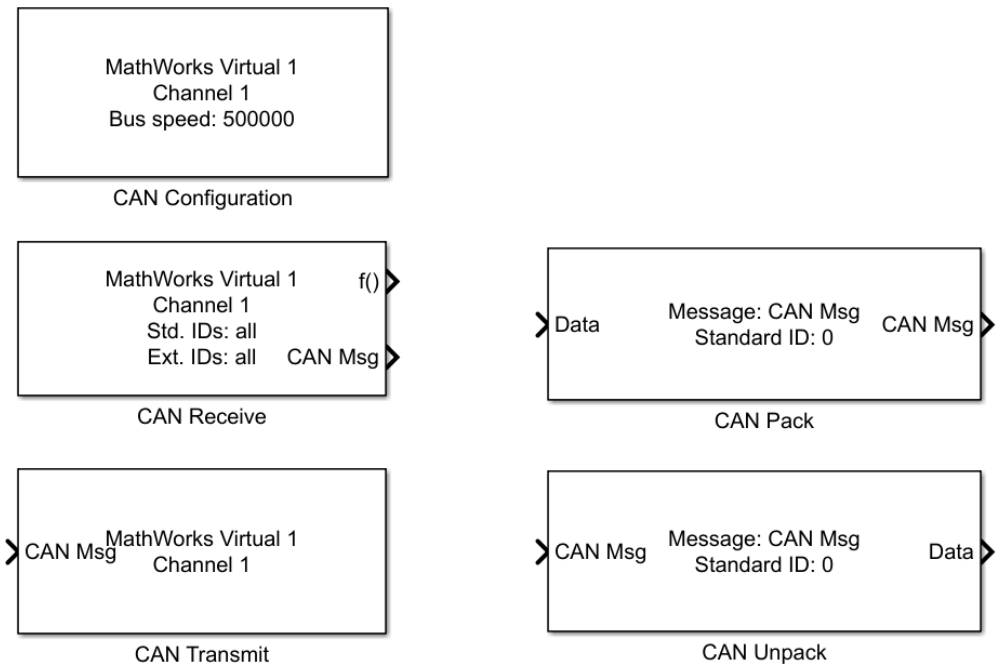

Figure 3.8. VNT Simulink blocks used in the model.

control the engine in test and brake bench (for both is allowed speed control and the torque control). These input sets will then be transmitted to the control software, which will be in communication with the HCU mounted on the test bench. At the same time on the screen of the interface should be mapped the trends of the main quantities of interest.

The main challenges are having real-time communication and maintaining data consistency. The first model has been realized in Simulink, and it includes a model of the graphical interface, to see if the data were transmitted and received correctly.

This first model developed to understand the data flow is shown in Figure 3.9 and is completely within the Simulink environment. From this representation is clear the data exchange between the GUI and the model of the test bench.

This diagram illustrates the data exchange between the Virtual GUI (the light blue part) and the Test Bench Model (the green part) within the Simulink environment, using a virtual CAN bus for communication. It represents the architecture

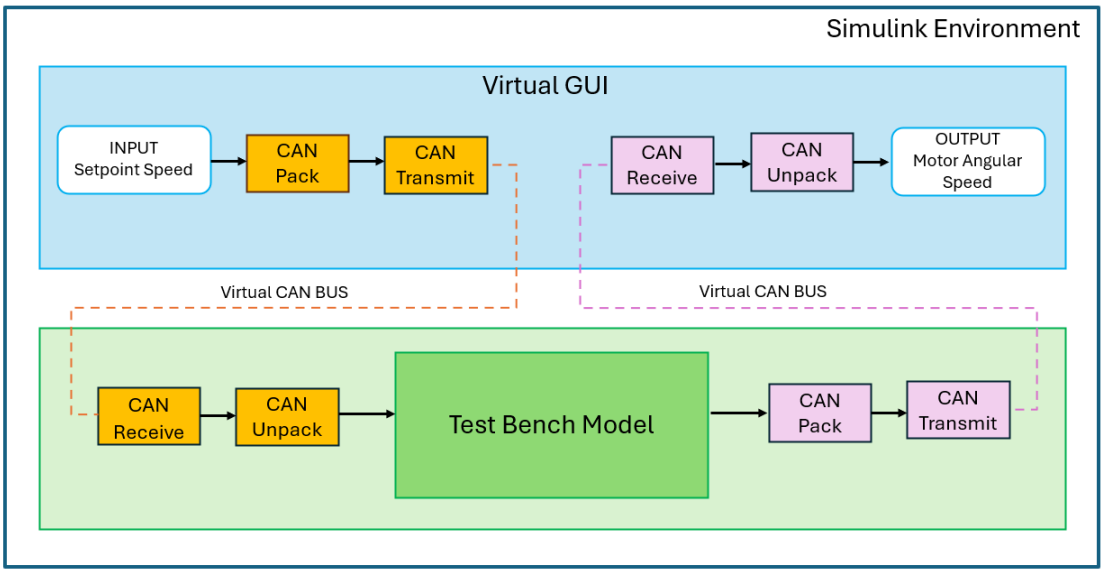

Figure 3.9. Data exchange between the virtual GUI and Simulink model.

of the thesis project, where the GUI interacts with the test bench model through CAN protocol, simulating a real-time control system.

- Step 1: The User Inputs a Setpoint Speed. The process begins with the user defining a setpoint speed directly within the GUI. This value represents the desired angular speed of the motor in the test bench model. However, before this numerical data can be sent to the model, it needs to be properly managed for transmission over the virtual CAN bus.

  This is where the CAN Pack block comes into play. The function of this block is to take the numerical value provided by the user and convert it into a structured CAN message, ensuring it follows the necessary protocol format. Once the message is correctly packed, it is passed to the CAN Transmit block, which is responsible for sending the data onto the virtual CAN network.

  At this point, the message is now in transit, traveling through the virtual CAN bus like a real CAN frame would in an actual hardware implementation.

- Step 2: Receiving and Processing the Data in the Test Bench Model.

On the other side of the communication loop, the Test Bench Model is constantly listening for incoming messages. As soon as the CAN message containing the setpoint speed reaches the test bench, the CAN Receive block picks it up.

However, the received message is still in its packed format, meaning it cannot be directly used by the test bench model. That is why the CAN Unpack block is necessary, it extracts the actual numerical value from the structured CAN frame, making it available for processing inside the model.

Now, the test bench model can use the setpoint speed as an input, the model simulates the behavior of the system, as illustrated in chapter 2 Figure 2.4. One of the key outputs of this simulation is the actual motor angular speed, which represents how the system reacts to the given setpoint.

- Step 3: Sending the Computed Motor Speed Back to the GUI.

  Once the motor angular speed is determined, the next step is to send this result back to the GUI so that the user can monitor the system's response in real time. To achieve this, the CAN Pack block is used to encode the computed speed into a CAN message.

  The packed data is then transmitted via the CAN Transmit block, sending the response back over the virtual CAN bus toward the GUI.

- Step 4: Displaying the Output to the User.

  Back at the GUI, the process mirrors what happened earlier. The CAN Receive block captures the incoming message, containing the computed motor speed. But, just like before, the data is still in packed format, so the CAN Unpack block is needed to extract the actual value.

  Finally, the unpacked motor speed is displayed in the GUI, allowing the user to see how the system reacts to the setpoint provided.

The last step completes the cycle of data exchange, starting from the user's input, traveling to the test bench model for processing, and then returning as an output to the GUI for real-time monitoring. After making sure that the logic works and that data are transmitted correctly, the GUI interface has been realized

through the MATLAB tool, MATLAB App Design, the code will be explained in detail in the next chapter. The final step for this thesis work was then replace the real can bus (referring to Figure 3.9, the orange and the pink dotted lines) with the devices CAN-USB provided by Peak System that are currently used on the real test bench, in order to perform the SIL simulation with real communication hardware.

Chapter 4

Design and Development of the Graphical User Interface (GUI)

The Graphical User Interface (GUI) plays a critical role in the operation of the Software-in-the-Loop (SIL) testing, enabling efficient real-time control, data visualization, and diagnostic capabilities. The tool used for the development of the User Interface is MATLAB App Designer.

MATLAB App Designer is an integrated development environment for designing and implementing graphical user interfaces (GUIs) in MATLAB. It offers an intuitive drag-and-drop interface combined with programmatic control, allowing users to create professional applications with integrated computational capabilities. App Designer generates applications using object-oriented programming principles, where user interface components (UI components) are defined as properties within a class. Event-based callbacks manage user interactions, ensuring responsiveness and dynamic behavior.

In the context of CAN bus communication, App Designer can be used to develop monitoring and control interfaces that interact with real or simulated devices. Integration with the Vehicle Network Toolbox (VNT) allows you to configure, send

and receive CAN messages in real time. Through the VNT, the application can interface with CAN hardware, read messages received from the network and display them in real time within the graphical interface. In addition, advanced features such as message filtering, data logging for later analysis and sending specific commands to test the behavior of network nodes can be implemented.

Thanks to these capabilities, App Designer is an effective tool for creating interactive applications for monitoring and debugging CAN communication, facilitating the testing and validation of HIL (Hardware-in-the-Loop) and SIL (Software-in-the-Loop) systems.

The GUI developed in MATLAB App Designer allows interfacing with the Simulink model of the test bench through a virtual CAN communication. Compared to the solution shown in the previous chapter, where the GUI was implemented entirely as a Virtual CAN Interface in Simulink, the current implementation uses the capabilities of MATLAB to manage the CAN protocol directly through code.

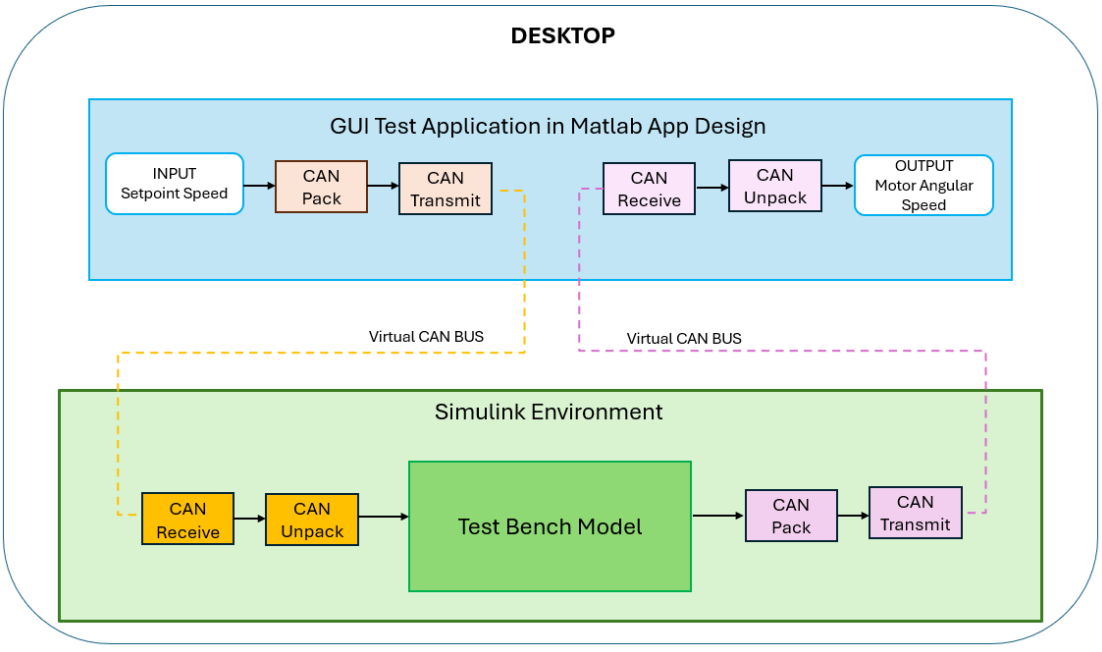

Figure 4.1. SIL configuration with MATLAB App Designer

The old solution was limited from the flexibility point of view of the GUI itself

because the transmission and reception logic were bound to the structure of the Simulink blocks. In the new implementation, these blocks are no longer needed: data pack, transmit, receive and unpack operations are now handled by MATLAB functions within the GUI code. This approach allows more control over message management and clearly separates the GUI logic from that of the Simulink model.

Figure 4.1 illustrates the new architecture: the GUI developed with MATLAB App Designer is responsible for sending and receiving CAN messages, while the Simulink model processes data and returns results. The data flow takes place via a virtual CAN channel provided by MathWorks, which simulates real communication between the two environments.

This chapter explores the design methodology employed, and its functional features for managing real-time data exchange and monitoring.

4.1 Design methodology using MATLAB App Designer

The developed GUI has been designed to provide an intuitive and user-friendly interface for managing the communication between the test bench model and Simulink via CAN bus. The interface aims to ensure ease of use, efficient data visualization, and a clear separation between input settings, system monitoring, and results analysis. The design follows a structured approach to enhance usability and facilitate debugging during testing operations.

- Tabs: each section is organized in a specific tab, each with a specific function, facilitating navigation and separating the different operations. This separation ensures that the user can easily switch between configuring inputs, viewing raw numerical outputs, and analyzing real-time system behavior. The tabs 1,2,3 have been used for testing the application, instead the tab illustrated in the Figure 4.2 contains the input and output of the simplified model of the test bench, as explained in chapter 2.
- Start button: this button starts the communication via CAN bus by executing several operations. First, clears any existing CAN channel object to ensure that there are no previous configurations that could cause conflicts.

Then, it initializes the CAN channel with the function setupCANChannel, setting up the communication parameters such as the CAN device, channel number, baud rate. After that, all the input fields are reset, ensuring the user starts from a clean state, and creates the different CAN messages that will be transmitted. Afterwards the Simulink model is started to allow real-time data exchange.

- Stop button: this button interrupts communication performing the following operations. It interrupts data transmission and reception, ensuring that no unwanted messages are sent. disconnects from the CAN bus, preventing further interactions until restarted, and then stops the Simulink model.

- Edit Fields: Allow users to input numerical values that will be used in the CAN message transmission. These fields provide real-time updates to mirror the current system state.

- Drop-down menu: allow the user to select the refresh interval of the plot, as shown in Figure 4.2.

- Plots: Display key system variables in real time, helping the user monitor trends, detect anomalies, and validate system behavior. The refresh rate can be adjusted to optimize performance.

- Signal lamps: have been used during the validation tests to provide immediate feedback of the status of the system. These indicators can be extended to show additional operating states, such as communication errors, current activity or CAN connection status.

The following section provides an overview of the algorithm implemented in MATLAB App Designer to manage the virtual CAN channel. The application is designed to simulate data transmission and reception of CAN in a controlled environment, allowing users to interact with the system through a graphical interface. To facilitate understanding, flowcharts are used to explain the algorithm's logic and the corresponding functions of the tool Vehicle Network Toolbox (VNT) in App Design.

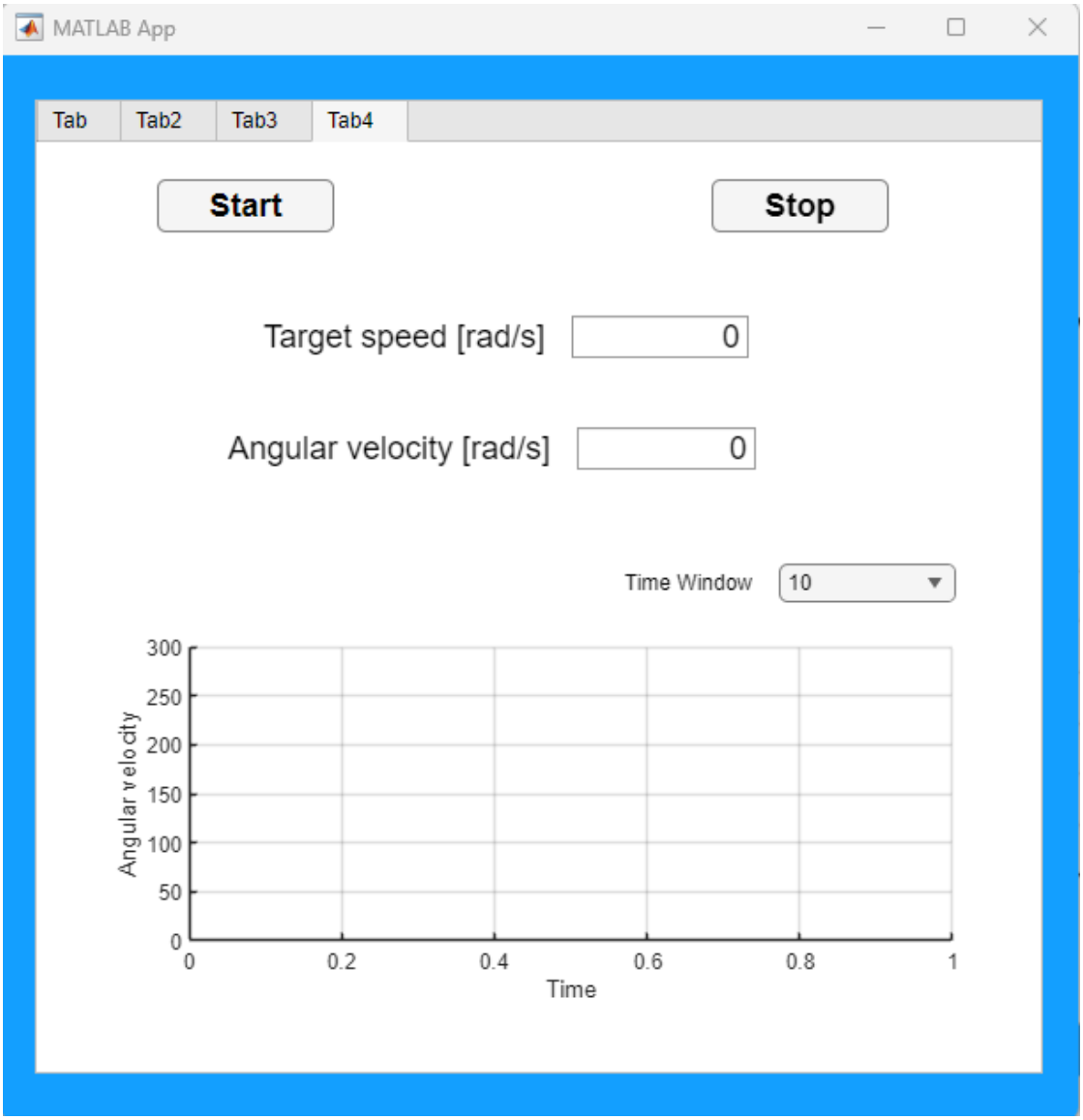

Figure 4.2. Tab of the GUI for motor control.

The Figure 4.3 provides a representation of the operations when the Start button is pressed within the graphical user interface. It describes the main steps and key functions involved in setting up the virtual CAN channel, to get ready the system for data transmission and reception. The logical flow is as follows: the UI inputs are reset to their default state, ensuring a clear start of the system. Any pre-existing CAN channel object is deleted to avoid conflict or errors during the

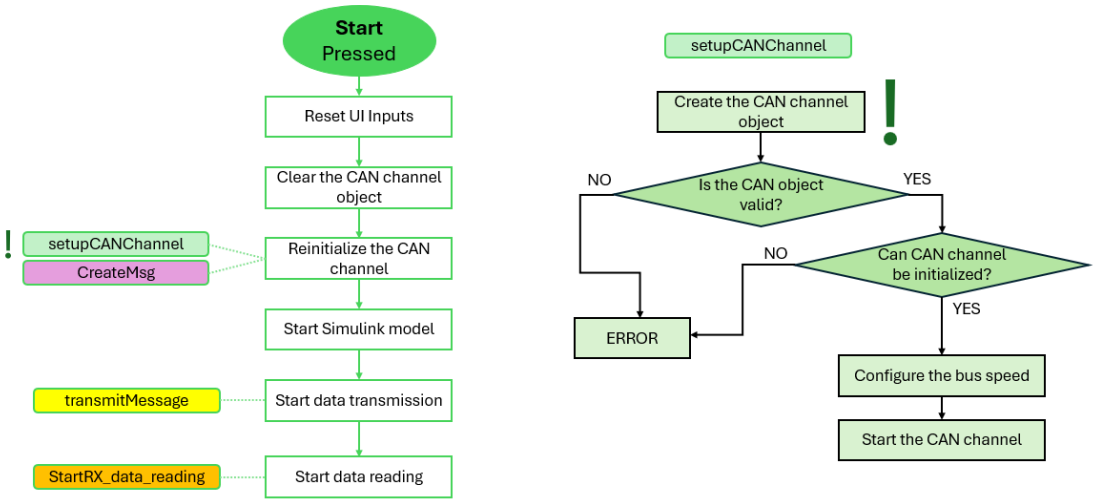

Figure 4.3. Initialization phase of the system with Start button and setupCANChannel function.

communication. Then, the CAN channel is initialized by calling the setupCAN-Channel function to configure the CAN channel, it will be explained in details later on.

Once the channel is ready, the Simulink model is started. To manage the message flow, the transmitMessage and startRX_data_reading functions are called, enabling a continuous data exchange to accurately simulate the real behavior of a CAN network. The first function, transmitMessage, ensures the transmission of simulated CAN messages, while the second one, startRX_data_reading, manages the reception of incoming data, updating the user interface with real-time information.

Looking at Figure 4.3, on the right side is illustrated how the setupCANChannel function works. This function is responsible for creating and configuring the virtual CAN channel. The step-by-step explanation of its logic is the following: the function initializes the CAN channel object, specifying the type of channel and configuration parameters. If the CAN object is invalid (e.g., due to incorrect parameters), an error is shown, such as if the CAN channel cannot be initialized, an appropriate error is displayed. Once validated, the bus speed is configured to match the desired specifications (e.g., 500 kbps for a virtual setup) and the CAN

channel is started, enabling communication.

The green exclamation points in the diagrams are used to indicate that those parts of the algorithm need modification when passing from a virtual CAN setup to a physical CAN bus. In the next paragraph, the configuration differences between the two simulations will be shown.

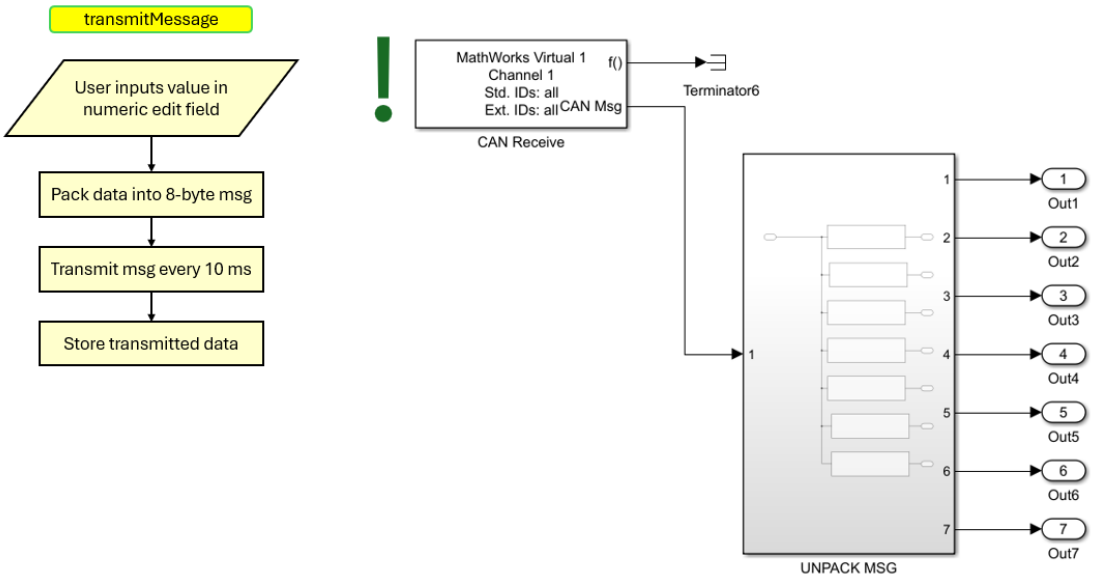

Figure 4.4. Flowchart for tranmitMessage function and Simulink model integration.

The Figure 4.4 shows the algorithm involved for the transmitMessage function, that is responsible for sending data over the virtual CAN channel. This function enables the application to simulate the transmission of CAN messages based on user input and a predefined schedule. When the user inputs a numeric value into the edit field within the GUI, the entered value is packed into an 8-byte CAN message. This is the standard format for CAN messages, ensuring compatibility with the CAN protocol.

The packed message is transmitted periodically (every 10 ms) using the transmitPeriodic function from VNT toolbox. This simulates the periodic transmission typical in real-world CAN systems. Each transmitted message is stored in a struct for future analysis or debugging. This feature is useful for monitoring the system's behavior and ensuring that the transmitted data is correct.

The diagram on the right of Figure 4.4, evidence how the messages transmitted by the transmitMessage function are received and processed within the Simulink model. The CAN Receive block is configured to receive all CAN messages transmitted on the virtual channel (MathWorks Virtual 1). It captures both Standard IDs and Extended IDs, ensuring that no message is missing. Also in this case, a green exclamation point is needed, this block requires modification when transitioning to a physical CAN setup.

The message Unpacking sub-system separates the data based on message IDs. Each output corresponds to a specific ID and the data payload associated with it. This step ensures that the received messages are correctly interpreted and can be used as inputs to the test bench model for further processing.

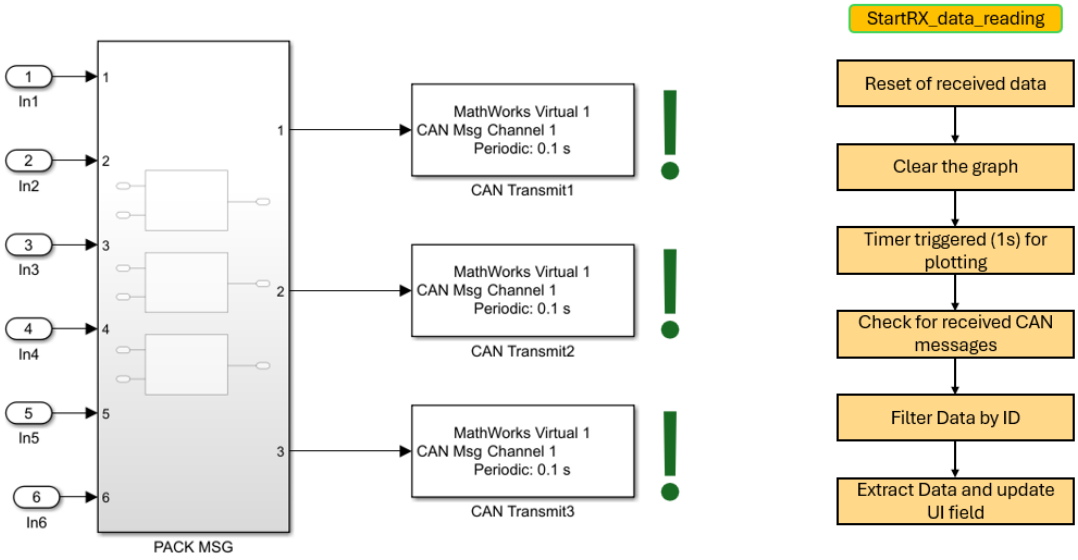

Figure 4.5. Description of Simulink message transmission and the StartRX_data_reading function.

Once the messages have been processed by the test bench model in Simulink, the next step is their re-packaging and transmission to the user interface for visualization. Looking at Figure 4.5 after the data is processed, it is prepared for transmission using CAN Pack blocks. These blocks take individual signals or parameters and pack them into CAN messages. Each CAN message is then assigned with a unique identifier (ID) and properly structured for communication over the

CAN bus. The prepared messages are then transmitted using CAN Transmit blocks.

It's important to note that the blocks marked with an exclamation point (!) in the diagram are currently configured for virtual CAN channels. When the system transitions to using a physical CAN cable, these blocks will be updated accordingly. Once the messages have been transmitted from Simulink, the application takes over to process and displays the received data.

This is where the StartRX_data_reading function becomes critical. This function is triggered to handle the reception and processing of CAN messages. Its workflow is described by the flowchart and can be broken down into the following steps: before initiating new data processing, the function resets all previously received data. This step ensures that old or irrelevant data does not interfere with the new session, maintaining data consistency.

Then, any graphs or visualizations in the user interface are cleared to provide a clean state for displaying the incoming data.

A timer is activated to update the graphical representation of data at regular intervals (e.g., every 1 second). This ensures real-time visualization of CAN message data without burdening the timing performance of the software. The function continuously monitors the CAN channel for incoming messages to ensure that no data is missing during the reading process. After that, to enhance usability, the received messages are filtered based on their CAN IDs. This step allows the application to isolate and process specific messages relevant to the user or system. The last step extracts the filtered data that is used to update the corresponding fields in the user interface. This could include displaying signal values, plotting graphs, or updating numerical indicators.

In the Figure 4.6 is represented the last point of the algorithm. When the Stop button is pressed in the application, a specific sequence of actions ensures that all operations related to data reading, CAN communication, and the Simulink model are safely terminated. The application immediately stops the ongoing process of reading data from the CAN Bus, this step ensures that no further messages are processed from the CAN channel while shutting down the system. Then, the CAN channel object is stopped, this action stops the transmission and reception of any new messages, preventing interference or conflicts during the shutdown

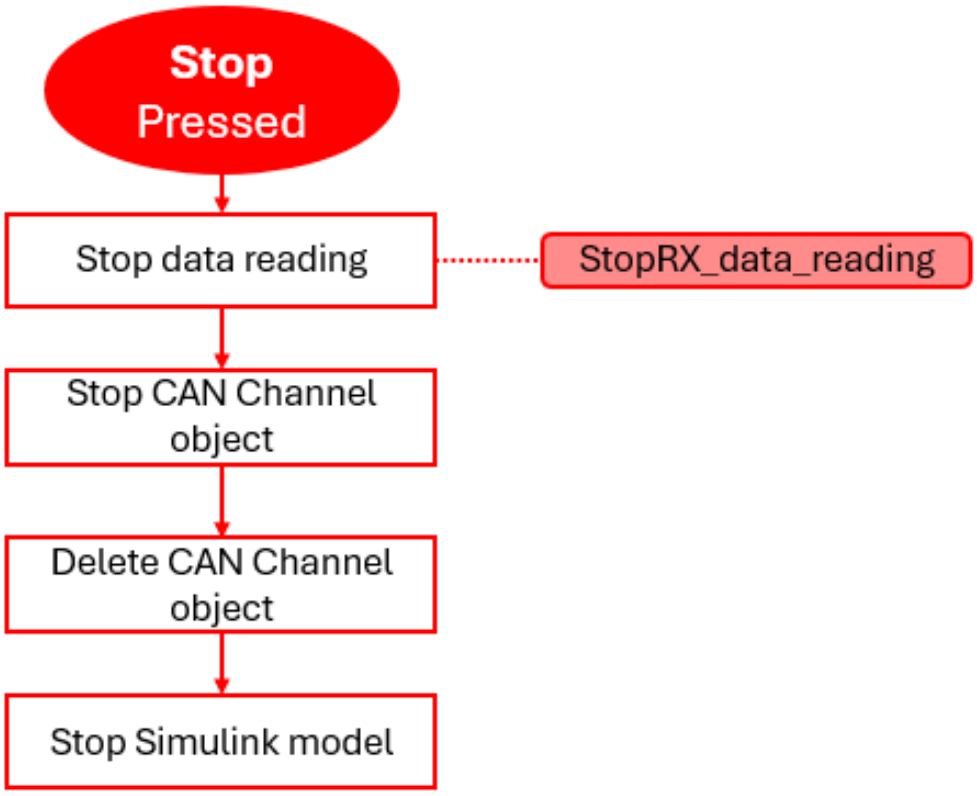

Figure 4.6. Flowchart of the stop procedure.

process. Once stopped, the CAN channel is deleted, to ensure that no residual configurations remain, which could affect future runs. Finally, the Simulink model is stopped.

The stop procedure is designed to ensure a reliable and controlled shutdown of the entire system, addressing critical aspects such as data integrity, resource management, and system safety. By stopping data reading as the first step, the system ensures that incomplete or corrupted data is not processed during the shutdown process. This is particularly important when real-time data transmission is involved, where even a little inconsistency could lead to inaccurate analysis or interruptions in subsequent operations.

By stopping and deleting the CAN channel object, the system prevents resource leaks that could otherwise impair the functionality of the CAN interface in future sessions. This step guarantees that the communication channels are reset and prepared for subsequent use, maintaining the reliability and efficiency of the system over time.

Finally, the procedure prioritizes system safety by ensuring that the Simulink model is stopped as the last step. This controlled sequence ensures that all system components are properly shut down, preventing abrupt interruptions or potential damage to hardware or software.

Both the start and stop procedures were critical parts during the thesis work, as they were the basis for ensuring that each simulation strictly adhered to timing and data consistency requirements. The entire process was designed with the aim of ensuring that CAN messages were processed, transmitted and received reliably, avoiding any form of corruption or data loss. Starting from these procedures was essential to build a robust system, capable of meeting the required time specifications and ensuring that every element of the test bench operated in perfect synchrony, without compromising the integrity of the data during the simulations.

4.2 Functional overview: real-time data transmission, reception and monitoring through CAN-Bus

The integration of real-time data transmission, reception, and monitoring through the CAN-Bus is the main point of this project, enabling a deep understanding of data flow within the system. The primary goal was to ensure that the messages transmitted and received respect timing requirements and data consistency, fundamental aspects for any real-time application, especially in the context of Hardware-in-the-Loop (HIL) and Software-in-the-Loop (SIL) simulations.

To achieve this, has been used the tool provided by the MATLAB VNT toolbox, CAN Explorer, a powerful platform for monitoring, verifying, and analyzing the correctness and timing of data exchanged over the CAN-Bus. The CAN Explorer provided real-time insights into message integrity, allowing for a detailed evaluation

of message transmission, reception, and traffic under various conditions. This was critical in identifying potential bottlenecks or issues that could compromise system performance.

Additionally, the simulations included a high-density CAN-Bus traffic environment by injecting into the system numerous additional messages. This approach was intentional and aimed at replicating real-world scenarios where the CAN-Bus is often saturated with data from multiple sources. By doing so, it was possible to test the robustness and reliability of the system under stress, as well as to assess its ability to handle extensive message exchanges without data loss or timing violations.

The virtual simulation setup was the preliminary testing stage, providing a safe and controlled environment to refine the system. This phase ensured that the timing of transmitted and received messages matched the requirements and validated that the implemented algorithms could handle high traffic volumes effectively. Moreover, the transition from virtual simulations to physical setups was designed to minimize errors when connecting to real-world systems, as explained in the next section.

Once multiple messages were added to the system to simulate CAN-Bus traffic, a critical challenge emerged: ensuring that all messages adhered strictly to their required transmission timings while maintaining data consistency. This necessitated a significant revision of the initial algorithm to optimize its structure and reduce the software's impact on the transmission process.

To address this, structured data containers (structs) were introduced for both transmitted and received messages in the software. Structs enable more efficient organization of the data, reducing the processing time required for handling and transmitting messages. Additionally, the timing for updating the data was set to a regular interval of 1 second. This decision not only ensured synchronization across system components but also minimized computational overhead, allowing the system to meet transmission timing requirements even under high traffic conditions.

Another critical aspect of this optimization was the development and integration of the start and stop functions, as detailed in the previous section. These functions played a crucial role in ensuring controlled initiation and termination of data transmission, guaranteeing that each operational cycle was synchronized

correctly. This structured approach ensured data consistency during all simulations and that timing requirements were reliably met, even in scenarios involving complex and dense CAN-Bus traffic.

To show results of the test with a high data flow condition, the following table 4.1 provides the timing characteristics of messages transmitted and received through the CAN-Bus. These results are derived from the post-processing of data collected during tests conducted with a high number of messages on the CAN-Bus. The data has been taken from 3 simulation cycles, with around 40000 messages for each transmitted message, and 4000 for each received message. The table includes the message ID, as well as the maximum (DeltaTimeMax), minimum (DeltaTimeMin), and mean (DeltaTimeMean) intervals observed between consecutive transmissions of each message.

ID	DeltaTimeMax [ms]	DeltaTimeMin [ms]	DeltaTimeMean [ms]
1	13.1	7	10.114
2	12.9	7	10.111
3	13.3	5	10.102
4	10.3	5	10.101
5	10.8	5	10.095
6	11.3	7	10.097
7	13.1	5	10.095
10	104	97	100
20	105	97	100
30	106	97	100
40	104	97	100
50	105	97	100
60	106	97	100
70	105	97	100

Table 4.1. Data timing results for transmitted and received messages for the virtual simulation.

IDs 1 to 7 correspond to messages transmitted by the GUI, typically system commands sent from the graphical user interface to the system. IDs 10 to 70 represent messages received by the GUI, which contain feedback to provide real-time monitoring. A key observation from the table is that the mean value of the transmitted messages, which constitutes the critical part of the simulation, consistently

remains close to 10 milliseconds. This shows excellent timing precision, as the maximum DeltaTime value for transmitted messages is around 13 milliseconds. These results highlight the robustness of the implemented algorithm in maintaining consistent timing even under heavy message traffic.

It is important to note that the timing of received messages is determined by the CAN Receive blocks in Simulink, which operate based on the configurations set within the Simulink model, in this case 100 ms. In contrast, the timing of transmitted messages is dictated by the transmitPeriodic function implemented in the MATLAB App Designer. The latter is a key element of the GUI's functionality, designed to ensure that messages are sent at regular intervals as specified by the user. However, confirming that the transmitted messages are coherent with desired timing is not a straightforward task; it requires additional monitoring tools.

For this purpose, CANexplorer was used as a monitoring and validation tool. Observing live traffic on the CAN-Bus using CANexplorer, it was possible to verify that the messages transmitted adhered to the desired timing intervals. The Figure 4.7 shows how the CANExplorer appears during live monitoring of message flow. This verification process was critical to ensuring the correctness and consistency of the transmitted data. Additionally, CANexplorer provided insights into the behavior of received messages, further validating the overall system's timing performance.

The results clearly demonstrate that the first goal of achieving accurate and consistent message timing in the fully virtual simulation was achieved successfully. This achievement was made possible by the structured design of the algorithm and the good configuration of both the transmit and receive functionalities. These elements proved essential in managing the challenges posed by high data flow conditions, ensuring that all messages consistently adhered to their specified timing constraints, a critical aspect of the thesis work.

The integration of Simulink, responsible for receiving messages, and MATLAB App Designer, handling the periodic transmission of data, highlights how the interaction between software tools can effectively replicate real-world scenarios and validate system behavior. By achieving this level of synchronization, the virtual simulation model successfully emulated realistic CAN-Bus traffic, establishing a reliable platform for further testing and future development.

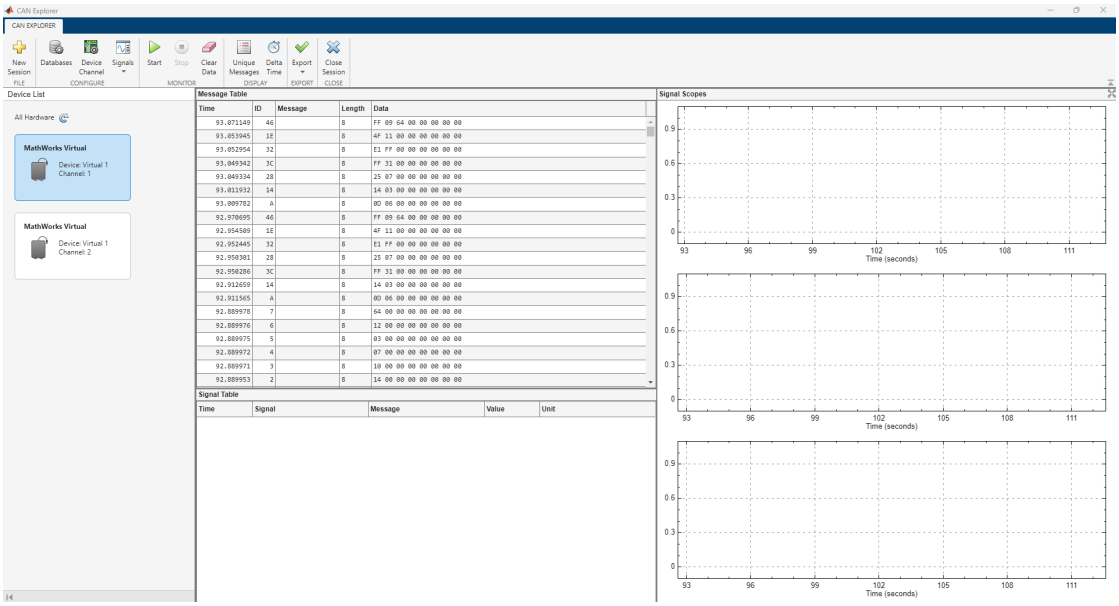

Figure 4.7. CAN Explorer for monitoring messages flow on Virtual CAN Channel.

4.3 Validation of the GUI: Numerical Simulations and Experimental Testing

The transition from a virtual simulation environment to real hardware testing represented an important step in the validation process of the developed system. While virtual simulations, Software in the Loop, provided a controlled and flexible environment for early-stage development and functional testing, the Hardware in the Loop provides a test including a real hardware. Testing with real CAN hardware allowed for a more accurate assessment regarding the system ability to manage actual CAN-Bus communication, ensuring it could handle realistic traffic loads, timing constraints, and effective message exchange.

Integrating the Peak System CAN-USB interfaces, which are the same devices installed on the real test bench in the company Ecothea, allowed the GUI to operate within a realistic hardware environment, bridging the gap between simulation and practical application. The configuration of the HIL is showed in the Figure 4.8.

The GUI, developed in MATLAB App Designer, is the interface for transmitting commands (e.g., setpoint speed) and receiving feedback (e.g., motor angular

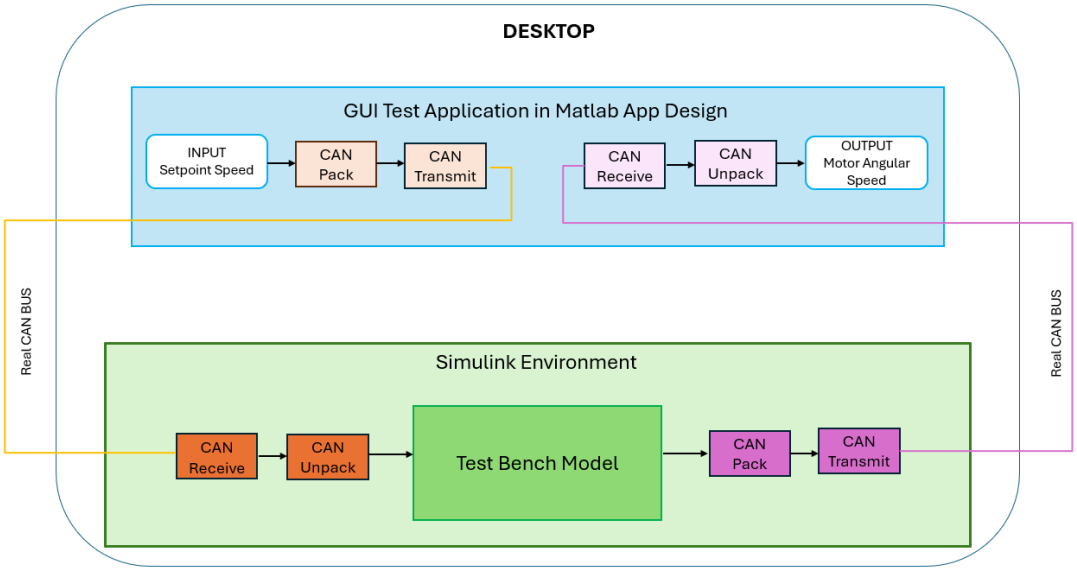

Figure 4.8. Hardware-in-the-Loop configuration of the system .

speed), with the key addition of physical CAN cables connecting the GUI to the hardware. Using CAN Pack and CAN Transmit blocks, messages are sent to the real CAN bus, while CAN Receive and CAN Unpack blocks decode incoming messages, providing real-time feedback to the GUI. The Simulink model, representing the test bench, results as the computational engine in the HIL setup, with the CAN Receive block processing messages from the real bus and unpacking the data for the model. Feedback data, such as motor angular speed, is then packed and transmitted back via the real CAN bus to the GUI. Physical connections are established through PEAK-System CAN-USB. Real CAN cables introduce the latency, noise, and constraints typical of physical networks.

One of the key objectives of transitioning to an HIL setup was to validate the GUI's dual role: as a transmitter of commands and as a receiver of feedback messages. On the transmitting side, the GUI needed to maintain precise timing and structured message formats, as previously demonstrated in the virtual simulation. On the receiving side, it had to accurately process and display feedback data from the system, ensuring real-time monitoring capabilities. By incorporating physical hardware into the loop, it became possible to evaluate the GUI's performance in

a realistic context, where timing and data consistency were directly influenced by external factors.

The HIL setup required the use of multiple CAN-USB interfaces and cables to replicate the system's real-world operation. Specifically, one cable was dedicated to the transmission of messages from the GUI to the system, while a second cable was used to receive feedback messages from the system to the GUI. Additionally, a third cable was employed to monitor the data flow in real-time using the CAN Explorer tool, ensuring that all messages adhered to the expected timing and structure.

The first step in transitioning from virtual CAN simulation to real CAN-Bus testing was adapting the system configuration to accommodate the physical hardware.

First of all, the differences between the virtual and real setups were highlighted by the green exclamation point in Figures 4.3, 4.4, 4.5.

The first difference is that two CAN objects (canChannelObj1 and canChannelObj2) were created, one for each of the physical USB ports of the PEAK-System device, but the logic of the function setupCANChannel is the same. The creation and configuration of these objects is essential to ensure communication between the GUI and the CAN network with physical hardware.

During the experimental validation, corresponding Simulink blocks were used to model the CAN communication network with the two channels. The CAN Configuration blocks were set up for each channel, as shown in Figure 4.9. The first block configures PEAK-System PCAN_USBBUS1 for transmission, while the second block configures PEAK-System PCAN_USBBUS2 for reception.

Then, looking at the Figure 4.4 the CAN Receive block in Simulink will be set in a different way for message reception on PCAN_USBBUS2. The parameters of this block were adjusted to allow all message IDs (standard and extended), with a sample time of 0.01 seconds to ensure real-time performance.

The configuration is shown in Figure 4.10.

This block reads messages transmitted over the CAN network and sends them to connected subsystems for further processing.

Looking at Figure 4.11, the CAN Transmit block is configured to send CAN messages to the physical CAN bus through the selected hardware interface. In

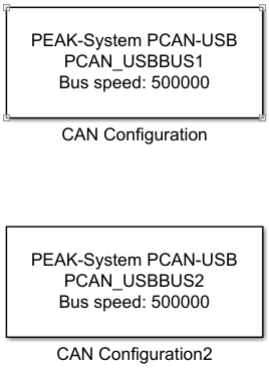

Figure 4.9. CAN Configuration in Simulink with PCAN USB peaks.

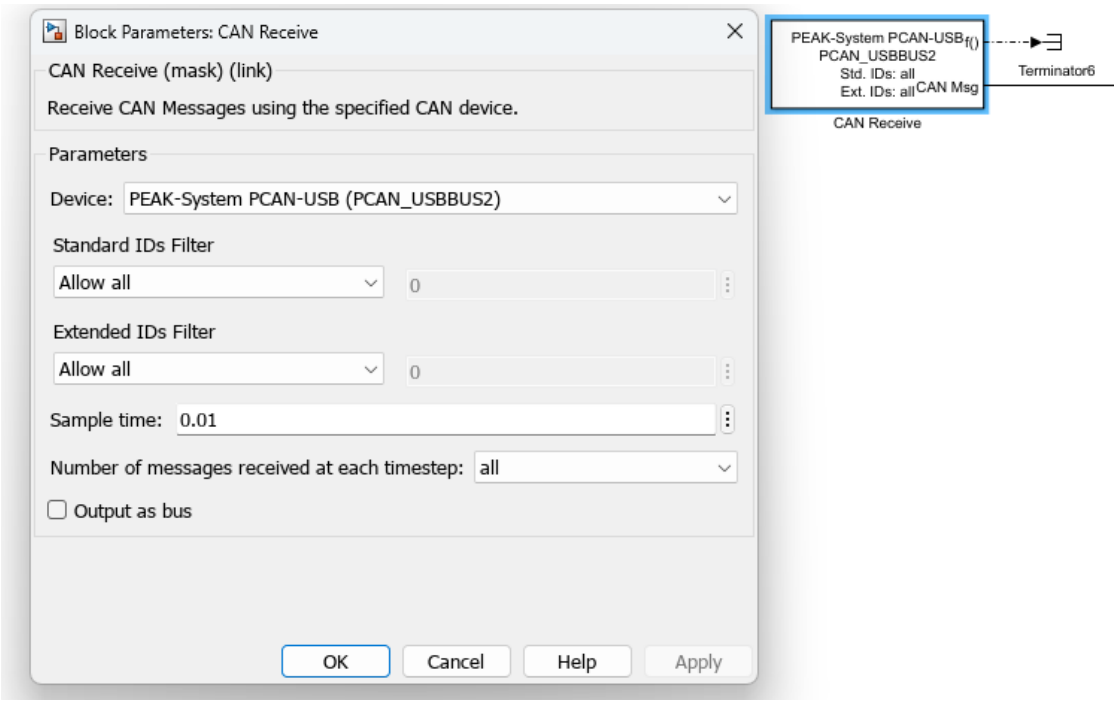

Figure 4.10. CAN Receive block configuration in Simulink with PCAN USB peaks

this case, the PEAK-System PCAN-USB 1 device is chosen as the communication channel. The transmission is set to periodic, with a defined Message Period of 0.1

seconds (100 ms). This ensures that the block continuously transmits messages at regular intervals, regardless of data changes.

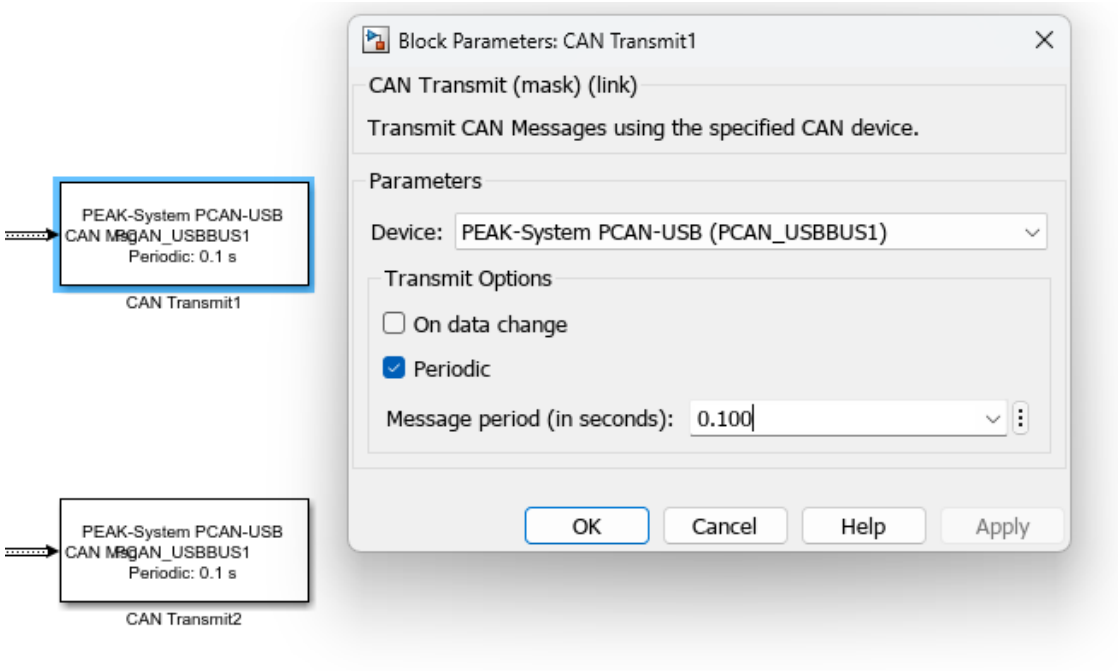

Figure 4.11. CAN Transmit block configuration in Simulink with PCAN USB peaks

As previously mentioned, it was necessary to add a third PEAK-System device to monitor the data flow using the CAN Explorer tool. Figure 4.12 illustrates the setup with all the devices connected seen by the CAN Explorer interface, which provides real-time insight into the communication on the CAN bus.

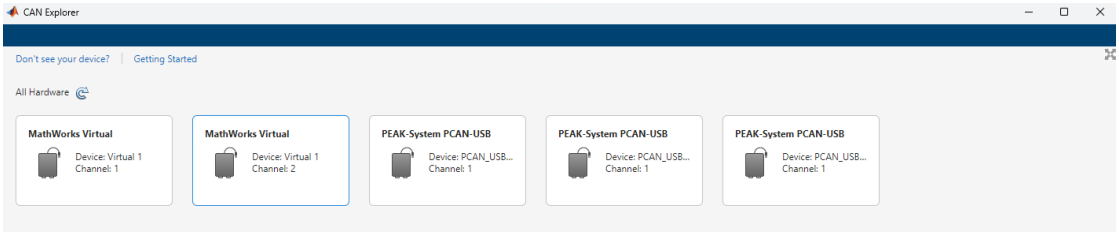

Figure 4.12. CAN Explorer for monitoring messages flow with all Hardware devices connected.

To thoroughly evaluate the performance of the updated system, simulations

were conducted to analyze the transmission times of CAN messages. These simulations replicate the scenarios tested in the previous section with a full virtual environment. The measurements were taken to investigate the potential impact of introducing physical hardware into the communication loop. Factors such as network latency, potential interference, and the overhead caused by the monitoring device were carefully considered. The table below 4.2 presents the transmission times of CAN messages under this configuration, offering a comparison to the earlier results.

ID	DeltaTimeMax [ms]	DeltaTimeMin [ms]	DeltaTimeMean [ms]
1	10.3	2	10.0003
2	10.5	2	10.0001
3	11.2	2	10.0001
4	11.1	2	10.0001
5	10.9	5	10.0002
6	11.7	2	10.0003
7	10.9	5	10.095
10	100.005	90	100
20	105.003	91	100
30	106.003	90	100
40	104.003	90	100
50	105.005	93	100
60	106.003	90	100
70	105.005	90	100

Table 4.2. Data timing results for transmitted and received messages for the simulation with PEAK-System PCAN-USB devices.

The transmitted messages must respect a fixed time cycle of 10 ms, but the numerical data from tests show differences between using real CAN PEAKS and virtual CAN channels. Using the PEAK-System PCAN-USB the mean time for transimtted messages (IDs from 1 to 7) is extremely close to 10 ms. PEAK hardware synchronization ensures that the rated transmission cycle is adhered to. PEAK's time management algorithms are optimized to reduce jitter and minimize any delays. It is interesting to highlight that the minimut delta time is below 5 ms with real hardware, this means that messages can be sent in advance of the full cycle.

This can happen if the bus is free, and the hardware transmission is programmed to ensure maximum efficiency. The maximum value for the delta time increase over the expected time, it can be attributed to traffic spikes on the CAN network or the priority of other messages on the bus. However, the increase remains within acceptable values for a robust drive system.

In virtual channels, the nominal cycle is less strictly respected due to the limitations introduced by the software implementation. The operating system (OS) manages processes and introduces latency due to multitasking and scheduling. The minimum delta time during tests with virtual channels is higher than in tests with real peaks. This increase compared to 2 ms in real PEAKS is due to the fact that virtual channels simulate transmission based on software timers that are not strictly linked to CAN hardware. Also with regard to the maximum delta time value, simulations with virtual peaks recorded slightly higher times than simulations with real peaks. The increase compared to real PEAKS is due to the lower efficiency of the software in responding to high load situations. Software jitter amplifies in scenarios where there are multiple simultaneous messages or load on the system.

As for the messages transmitted, there are no great differences between the virtual simulation and the one with real peaks. This is due to the fact that the messages received were with a longer warp time than those transmitted (100 ms vs 10 ms).

Real PEAK devices offer a number of advantages that make them better with respect to virtual channels, especially in applications that require time accuracy and reliability. One of the key factor is hardware time synchronization. PEAK devices use a dedicated, high-precision internal clock, which allows nominal transmit and receive cycles to be strictly respected. This clock is independent of the operating system, eliminating the uncertainties introduced by multitasking and software timers, typical of virtual channels. The stability of these clocks ensures that messages are transmitted or received exactly on schedule, with negligible jitter.

Another crucial aspect is the presence of dedicated hardware buffers. Real PEAKS use optimized internal memories to handle incoming and outgoing CAN messages. This allows you to maintain high performance even in heavy traffic on

the CAN bus. In virtual systems, however, buffering is handled at the software level, which introduces higher latency and increases the risk of data loss or overhead in message-dense scenarios.

Real PEAKS are also better at handling the priority of CAN messages. The CAN protocol prioritizes messages based on their ID: messages with lower IDs have greater urgency and can access the bus before the others. In real PEAKs, this feature is implemented directly in hardware, ensuring an immediate and correct response. In virtual channels, arbitrage is simulated via software, making it less accurate and potentially slower in high-load situations.

Finally, the optimized hardware drivers of real PEAKS allow to make the most of the capabilities of the CAN bus. Direct communication between the hardware and the bus minimizes processing time, ensuring that messages are transmitted and received with insignificant delays. In virtual channels, on the other hand, communication passes through the operating system, introducing overhead due to the management of processes and system resources.

Using real PEAK devices has significant implications for accuracy and reliability in hardware-in-the-loop (HIL) and software-in-the-loop (SIL) testing. In HIL systems, the ability of PEAKS to replicate the real-world operating conditions of the CAN bus is essential to ensure that test results are representative of actual operating conditions.

The time accuracy and the absence of significant jitter allow to reliably evaluate the behavior of the system, preventing any errors introduced by the simulation from affecting the outcome of the tests.

From a scalability perspective, true PEAKS are designed to handle message-dense CAN networks without any performance degradation. This makes them particularly suitable for complex systems, such as those used in electric vehicles, where multiple control units communicate simultaneously. In virtual channels, on the other hand, scalability is limited by the capacity of the operating system and the computational resources available, which can become a bottleneck in high-traffic applications.

Another important practical aspect is the responsiveness of the system. With optimized hardware drivers and dedicated buffers, true PEAKS provide faster and more stable transmission and reception than virtual channels. This responsiveness

is crucial in applications that require real-time control, such as electric motor management in a real test bench.In virtual channels, the latency introduced by the operating system can cause delays that impair the system's ability to react in a timely manner.

The developed GUI represents a significant achievement both from a design and practical point of view. During initial tests with the virtual CAN channel, the interface demonstrated excellent reliability and accuracy in handling the transmission and reception of messages, allowing detailed analysis of the system's behavior in a simulated environment. This robustness made it easy to transition to the use of real PEAK devices, thanks to a simple and intuitive configuration of data transmission. The GUI architecture, designed to be flexible and modular, made it possible to quickly adapt the parameters required to operate with the physical network, maintaining consistent and precise results even in a real context. This highlights not only the quality of the design, but also the ability of the application to respond effectively to different operating conditions, confirming itself as a robust and reliable tool for both HIL and SIL testing.

Chapter 5

Conclusions and Future Work

The final chapter of this thesis summarizes the main results obtained and offers an overview of possible future evolutions. This work focused on the creation of a Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) system for an electrical test bench, highlighting the integration and validation of CAN communication within a simulation environment. An overview of the key results is then provided, followed by a discussion on the potential future development of the model towards more advanced configurations and new applications.

5.1 Summary of key findings and insights

The work described in this thesis represented a significant contribution in the development and validation of CAN-based communication systems for a test bench. With a focus on Chapters 3 and 4, which deal with CAN communication and Graphical User Interface (GUI) design, respectively, the main results can be summarized as follows:

- Integration and management of CAN communication within Simulink and GUI. A key result of this thesis was the development of robust CAN communication, which supported both SIL simulation and tests with real hardware

devices, integrating typical aspects of HIL configuration. Using the Vehicle Network Toolbox, a virtual CAN configuration was designed to transmit and receive messages between the test bench model and the GUI. The use of Peak System CAN-USB devices allowed the system to be extended to a HIL configuration. This made it possible to validate the transmission and reception of CAN messages in real time using physical hardware, faithfully replicating the behavior of the real test bench. Complex scenarios have been implemented and tested, including high traffic on the network, message loss, and bus offs. These tests demonstrated the system's ability to maintain stability and consistency in data, even under critical conditions.

- Transition from SIL to HIL. A crucial aspect of this thesis was the transition from the SIL model to a partial HIL configuration, which allowed to integrate real hardware elements. The connection between the GUI and the test bench was made using physical CAN-USB devices, demonstrating the possibility of testing the software in semi-real conditions and verifying the consistency of the CAN messages with respect to the physical bench. The developed system provides a solid foundation for future expansions to a complete HIL, including physical components such as motors and sensors, enabling testing under realistic operating conditions.

- Performance and robustness of the system. The tests carried out have highlighted the robustness of the system developed. CAN messages were handled correctly both in simulation environments and with real hardware, ensuring accuracy in the transmission of critical parameters such as speed and torque. The system has shown the ability to handle fault injection scenarios, such as message loss, overloaded buses, and silent nodes, due to error handling strategies implemented in the software.

In summary, the results obtained in this thesis demonstrate how the integration of SIL and HIL systems, supported by reliable CAN communication and an advanced GUI, represents an effective approach for the validation of electrical test benches. This work provides a solid basis for future developments, both in the improvement of the actual system and in the application to more complex areas.

5.2 Future developments: expanding functionalities and new applications

The work developed in this thesis offers numerous opportunities to expand the functionality of the test bench and adapt it to new applications, leveraging both Software-in-the-Loop (SIL) configuration and Hardware-in-the-Loop (HIL) integration with real devices. Future developments can be divided into three main areas: model improvement, integration of advanced architectures, and application expansion.

- Improved model and simulation capabilities. To further improve the functionality of the developed model, the following interventions can be considered. Regarding the model dynamics, it could be useful to integrate advanced models of electric motors that include phenomena such as electrical losses, magnetic saturation, the effect of harmonics, and thermal dynamics, to more accurately simulate the real-world behavior of components. Implement a current and power measurement system to monitor consumption in real time and analyze energy recovery strategies during regenerative braking. Replace PI controllers with predictive control (MPC) or adaptive control techniques, which can better handle complex and dynamic scenarios, such as sudden load changes or failures. Simulate the thermal behavior of batteries and motors to assess the impact of overheating on the system and optimize cooling strategies.

- Integration with full HIL configurations. Integration of a complete HIL configuration is the next natural step for the test bench. Connect physical components such as motors, inverters, sensors, and electronic control units (ECUs) to the simulation model to validate behavior under real-world operating conditions. Expand fault injection testing to include physical failures, such as sensor failures, overheating, or communication problems on the CAN network, and evaluate the response of the system.

- Expansion to complex applications. The architecture of the test bench can

be expanded to include more advanced configurations and more complex applications. Adapt the model to simulate systems with multiple electric motors and advanced transmissions, such as those used in heavy-duty vehicles and agricultural machinery. Integrate vehicle-to-everything (V2X) communication protocols to simulate cooperative driving, traffic management, and vehicle-to-infrastructure communication scenarios.

- Automation and optimization of the testing processes. An important aspect for improving the test bench is the automation of test processes. Integrate machine learning-based analytics tools to spot anomalies in the data, such as changes in performance or signs of imminent failure.
- Standardize CAN configurations. Develop a CAN database (DBC file) for more effective management of large-scale communications.

Bibliography

[1] F. Mocera, "A model-based design approach for a parallel hybrid electric tractor energy management strategy using hardware in the loop technique," *Vehicles*, vol. 3, no. 1, pp. 1–19, 2020.

[2] N. Sharma, B. Jiang, A. Rodionov, and Y. Liu, "A mechanical-hardware-in-the-loop test bench for verification of multimotor drivetrain systems," *IEEE Transactions on Transportation Electrification*, vol. 9, no. 1, pp. 1698–1707, 2022.

[3] J. Huang, S. S. Naini, R. Miller, D. Rizzo, K. Sebeck, S. Shurin, and J. Wagner, "A hybrid electric vehicle motor cooling system—design, model, and control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4467–4478, 2019.

[4] R. Ahmadi, P. Fajri, and M. Ferdowsi, "Dynamic modeling and stability analysis of an experimental test bench for electric-drive vehicle emulation," in *2013 IEEE Power and Energy Conference at Illinois (PECI)*, pp. 88–94, IEEE, 2013.

[5] F. Mocera and A. Somà, "A review of hybrid electric architectures in construction, handling and agriculture machines," *New Perspectives on Electric Vehicles*, p. 49, 2022.

[6] A. Rassõlkin and V. Vodovozov, "A test bench to study propulsion drives of electric vehicles," in *2013 International Conference-Workshop Compatibility And Power Electronics*, pp. 275–279, IEEE, 2013.

[7] A. Mondal, J. Khan, S. Prins, and S. Kumaravel, "Control of dual motor test bench for performance testing of pmsm for traction application," in *2023 IEEE Silchar Subsection Conference (SILCON)*, pp. 1–6, IEEE, 2023.

[8] J. Brousek, L. Krcmar, and P. Rydlo, "Efficiency measuring of electric drive with traction synchronous motor with permanent magnets," in *2021 IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM)*, pp. 1–5, IEEE, 2021.

[9] W. Xiaoxu, W. Sibo, C. Mingjian, and Z. Huichao, "Efficiency testing technology and evaluation of the electric vehicle motor drive system," in *2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific)*, pp. 1–5, IEEE, 2014.

[10] H. Zha and Z. Zong, "Emulating electric vehicle's mechanical inertia using an electric dynamometer," in *2010 International Conference on Measuring Technology and Mechatronics Automation*, vol. 2, pp. 100–103, IEEE, 2010.

[11] F. Mocera and A. Somà, "Analysis of a parallel hybrid electric tractor for agricultural applications," *Energies*, vol. 13, no. 12, p. 3055, 2020.

[12] F. Mocera, V. Martini, and A. Somà, "Comparative analysis of hybrid electric architectures for specialized agricultural tractors," *Energies*, vol. 15, no. 5, p. 1944, 2022.

[13] P. Mindl, P. Mňuk, Z. Čřfovskỳ, and T. Haubert, "Ev drives testing and measurement system," in *2015 International Conference on Electrical Drives and Power Electronics (EDPE)*, pp. 328–332, IEEE, 2015.

[14] V. Rjabtšikov, M. Ibrahim, A. Rassõlkin, T. Vaimann, and A. Kallaste, "Ev-powertrain test bench for digital twin development," in *2022 IEEE 20th International Power Electronics and Motion Control Conference (PEMC)*, pp. 559–563, IEEE, 2022.

[15] V. Burenin, J. Zarembo, A. Žiravecka, and L. Ribickis, "Model of laboratory test bench setup for testing electrical machines," in *2020 IEEE 61th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON)*, pp. 1–5, IEEE, 2020.

[16] P. M. Fonte, P. Almeida, R. Luís, R. Pereira, and M. Chaves, "Powertrain test bench system," in *2019 Electric Vehicles International Conference (EV)*, pp. 1–5, IEEE, 2019.

[17] A. Somà, F. Mocera, and S. Venuti, "Thermo magnetic fem simulation of a pm synchronous motor with input data from telemetry driving cycles," in *IOP Conference Series: Materials Science and Engineering*, vol. 1214, p. 012050, IOP Publishing, 2022.

[18] T. M. Inc., "Matlab version: 9.13.0 (r2023)," 2022.