Project Report
Adaptive Cruise Control

Compliance Design of Automotive Systems
a.y. 2020/2021

Group Members:
Barro Alessandro, Bruscoli Nicolas, Ceccarelli Viviana,
Chiacchiararelli Leonardo, Cintura Manuel, Mariani Lucia

May 11, 2021

Contents

List of Figures

Chapter 1

Introduction

The reason for the choice of the Adaptive Cruise Control is because it is one of the most commonly used features of the ADAS. ADAS (Advance Driver Assistance Systems) are the electronic systems that assist drivers in driving and parking functions. ADAS increase car and road safety, using automated technology, such as sensors and cameras.
Nowdays, more and more vehicles are equipped with them, often as standard features. Adaptive cruise control is one of the most advanced technology, being able to facilitate the driving.
The Adaptive Cruise Control (ACC) is an active safety system that automatically controls the acceleration and braking of a vehicle. It is activated through a button on the steering wheel and disengage by the driver's braking and/or another button. This report offers an example which illustrates the benefits of using Model-Based Design in the engineering workflow. The goal is to show how this approach can be used in the entire development of the system design.
This report is divided in to the following sections:

- *Chapter* 2: The project organization is shown and the Model-Based Design is esplained;
- *Chapter* 3: The Model explanation and the design of the Adaptive Cruise Control are illustrated;
- *Chapter* 4: The different steps for the project development (Unity test,MIL, Automated Generated Code,SIL and PIL simultion are described;
- *Chapter* 5: A conclusion with the future developments.

Chapter 2

Project organization

2.1 Hardware and software components

Hardware and software components used to realize the project are listed below:

2.1.1 Hardware

The hardware used for this project is a PC running Windows and a Raspberry Pi 4 Model B.
The Raspberry has been used as the core component of the Processor-in-the-loop simulation.

2.1.2 Software

The software used for this project is Simulink: Embedded Coder package, Simscape package, Powertrain Blockset are required. The software employed to organize the work and the communication between team members:

- GitHub: VCS;
- LaTeX: for the realization of the documentation;
- Microsoft Teams: for the meetings;
- Draw-io: for the graphs realization.

2.2 Model-based design

The realization of the project is based on the V-model, which is one of the most used approaches for the model-based design. The V-Model highlights the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time or project completeness (left-to-right) and level of abstraction, respectively.

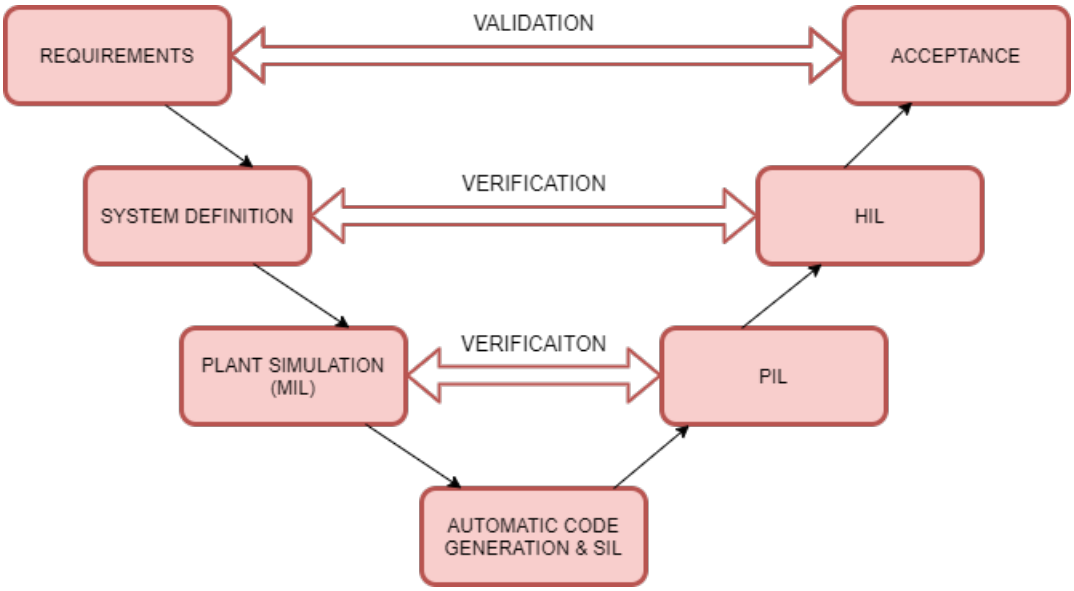

Figure 2.1: The V-model

The project definition phases are:

- **Requirements**: evaluation of the model requirements, planning how to proceed with the following steps and fixing the final goals;
- **System definition**: defining the high-level design for the model;
- **Plant simulation**: realization the simulation of the plant using Simulink;
- **Automatic code generation**: using Embedded Coder, the low-level code for the controller is generated;
- **PIL**: deploying he generated code to the hardware, performing the processor-in-the-loop simulation;
- **HIL**: the plant should be simulated in the HIL module. This step could not be implemented;
- **Acceptance**: the real plant is realized, integration tests are performed. This step could not be implemented.

Chapter 3

Model explanation & design

As anticipated before, the this project aims to design an Adaptive Cruise Control: to design it following the MBD, we of course need a plant that models the vehicle longitudinal dynamics. In Fig 3.1 it is possible to see the whole plant model.

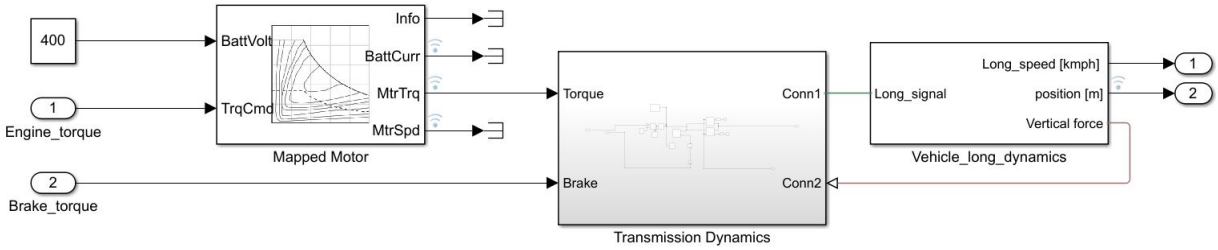

Figure 3.1: Plant

Going deeper, it is possible to see that it consists of three main blocks:

- Vehicle longitudinal dynamics block;
- Transmission dynamics block;
- Mapped motor block.

The description starts from this last block.

3.1 Mapped motor

This first block is used to implement a mapped motor and drive electronics operating in torque control mode: the output torque tracks a reference demand one, directly coming from the output of the PID controller. This torque will be one of the possible input of the next block, the Transmission Dynamics one. Particularly, from the output of the PID controller we could have a positive torque if the vehicle needs to accelerate, or a negative torque if the car needs to brake: the first one is the above mentioned input of the Mapped Motor block, the latter one will be directly applied to the brake system in the Transmission Dynamics block.

3.2 Transmission Dynamics

Thanks to this block it is possible to model the entire driveline, from the ECU torque request to the motion of the wheels.

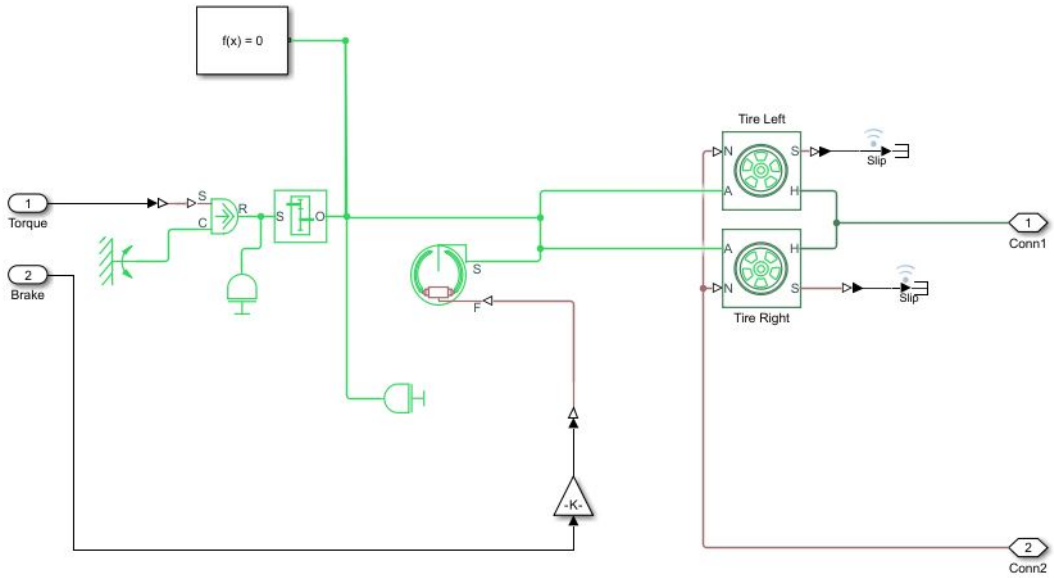

Figure 3.2: Transmission Dynamics block

As anticipated before, this block has two inputs:

- The positive torque input, output of the Mapped motor block;
- The negative torque input, output of the PID controller.

In the first case, the torque is modified by a gear ratio block which models the one-speed transmission of the electric vehicle. Conversely, when a negative torque is requested, the brake block provides a negative torque directly at the wheel input. Then, the Simscape wheels blocks model the behaviour of the wheels in terms of inertia, rolling resistance and slip, providing an angular speed of the wheel proportional to the torque given as input.

3.3 Vehicle longitudinal dynamics

Finally, to model the longitudinal dynamics of the vehicle, from the Simscape Driveline library the "vehicle longitudinal dynamics" block has been picked.

This block represents a two-axle vehicle body in longitudinal motion: it accounts for body mass, aerodynamic drag, road incline, headwind speed (in these simulations they are not considered influencing quantities)and weight distribution between axles due to acceleration and road profile. This block accepts as input the resulting traction motion developed by tires (that is to say, the output of the Simscape wheels blocks), giving as output the vehicle velocity and also the front normal wheel forces.
Finally, from this block is also emulated the radar function, that is to say the monitoring of the distance between this vehicle and the leading one: in fact, thanks to the presence of an integrator block, from the actual speed of the vehicle its position is computed. Then this position is fed back and subtracted to the position of the leading vehicle, resulting in radar distance.

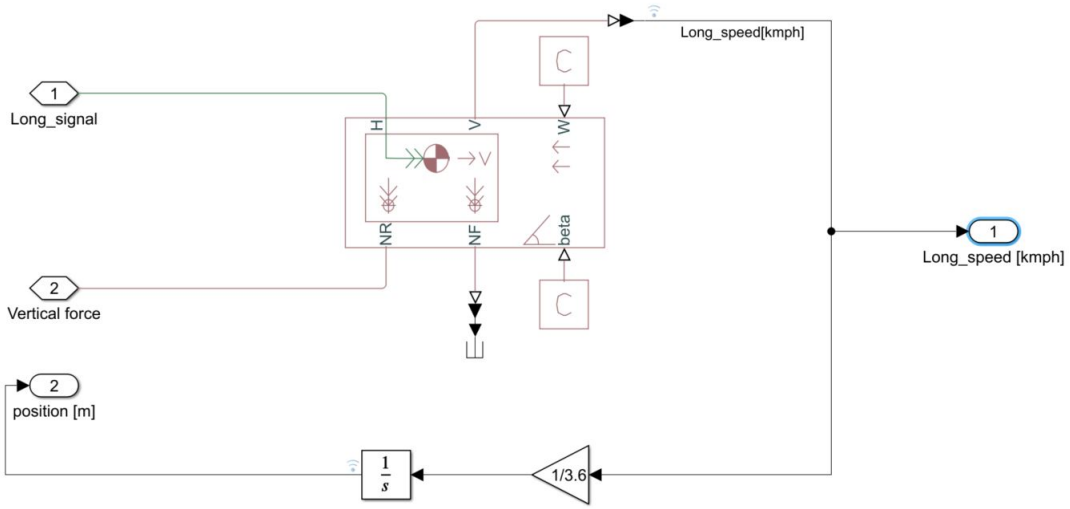

Figure 3.3: Vehicle longitudinal dynamics block

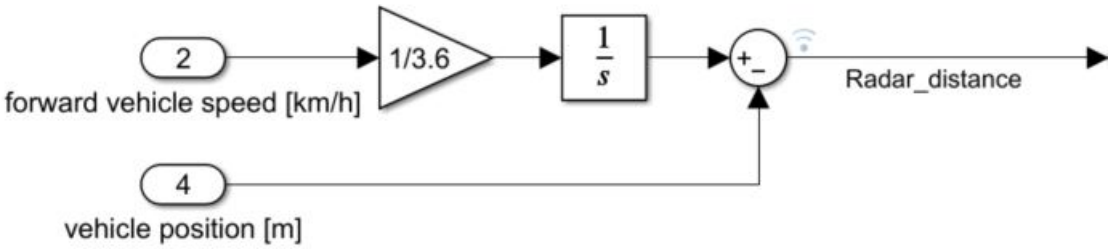

Figure 3.4: Emulation of the radar function

Chapter 4

Project development

4.1 Requirements

4.1.1 Software requirements

In order to be able to interact with the developed model, the following software tools are needed:

- Matlab R2019b;
- Simulink;
- Simscape;
- Simscape Driveline.

4.1.2 High-level requirements

The goal of the overall control system is to extend the mission of the simpler Cruise Control: regulate the speed to the desired one and keep it. Additionally, the Adaptive Cruise Control adapts the speed of the vehicle with respect to the velocity of the following one, being able to correctly maintain the safety distance dictated by the Highway Code when needed (i.e. when the current distance between the vehicles is lower than the required safety distance):

$$d_s = d_{min} + \frac{1}{k}v^2$$

4.1.3 Low-level requirements

The designed system is intended to work on an electric vehicle, considering also the state of charge of the battery: particularly, the overall behaviour of the control action will be more or less aggressive depending on the state of charge level.

Thanks to the presence of a sensor (typically a radar installed in the front of the car), the distance with respect to the following vehicle is continuously monitored: when this distance falls below the threshold dictated by the Highway Code minus a safety margin quantity (one meter as dictated by requirement, to be sure to be able to brake in the correct timing), the control must act on the brakes. Then, during the acceleration phase,

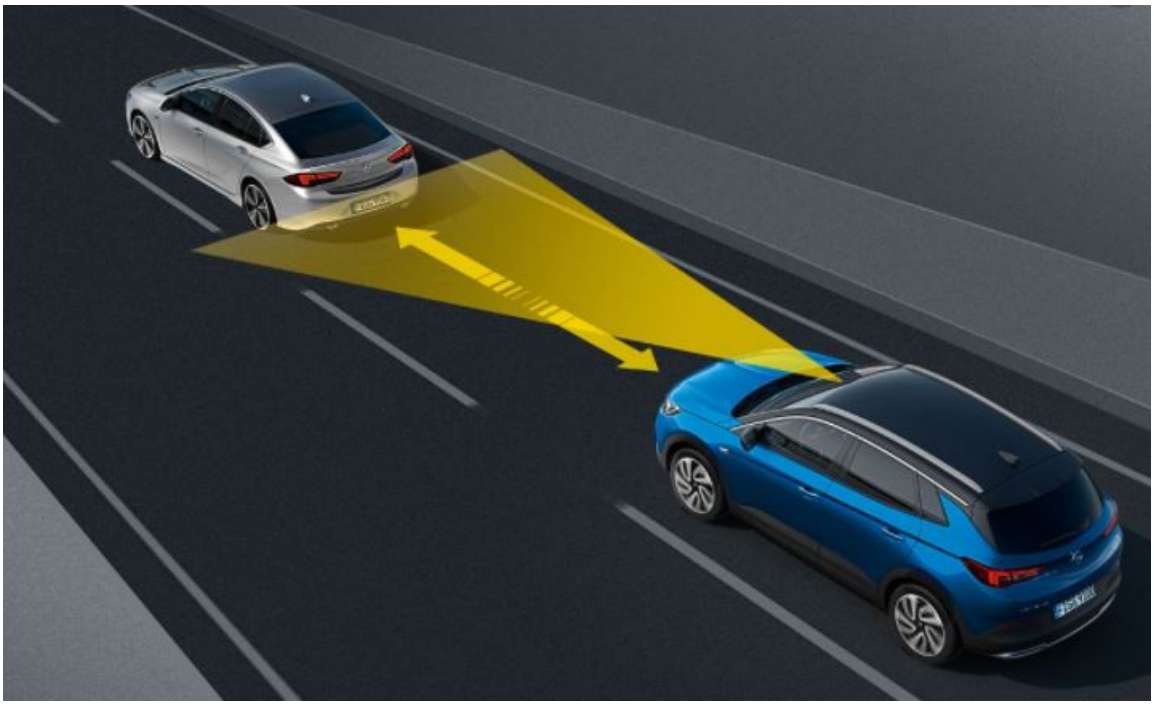

Figure 4.1: ACC

Figure 4.2: Front radar

the system needs to be designed as following: when the first vehicle accelerates, the control must operate so that the vehicle speed becomes the one desired if it is possible, that is to say if the two vehicles are far enough. Otherwise, the system must adequate the vehicle speed to the one needed to keep the safety distance.

4.2 Unit & Integration testing

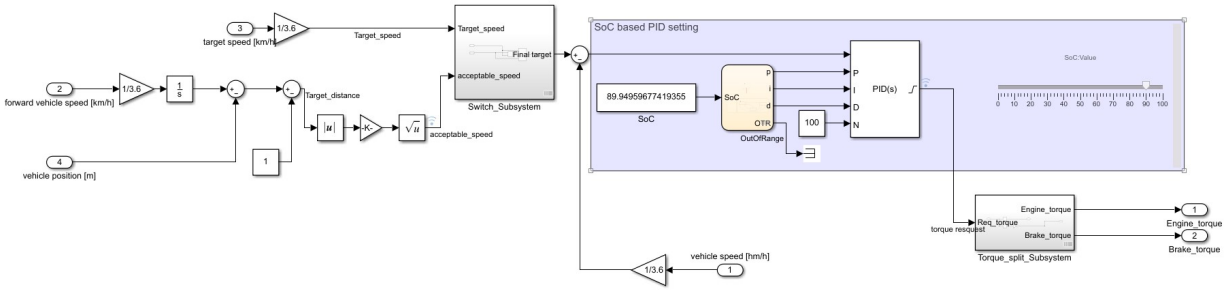

Figure 4.3: An overview of the controller block

The controller is composed of 4 main blocks: Switch_Subsystem, SoC handler, Torque_split_subsystem and the PID regulator. While it is impossible to test the PID behaviour without a model to be controlled, it is necessary to test separately the other blocks. The units have been tested using the Simulink Test tool. Initially a harness has been built for each unit to isolate it from the rest of the model, then the input signals have been designed in such a way to stimulate the units under all possible conditions. Various baseline signals have been designed in order to tell the tester which output is expected when feeding the unit with a given input.

4.2.1 Switch subsystem

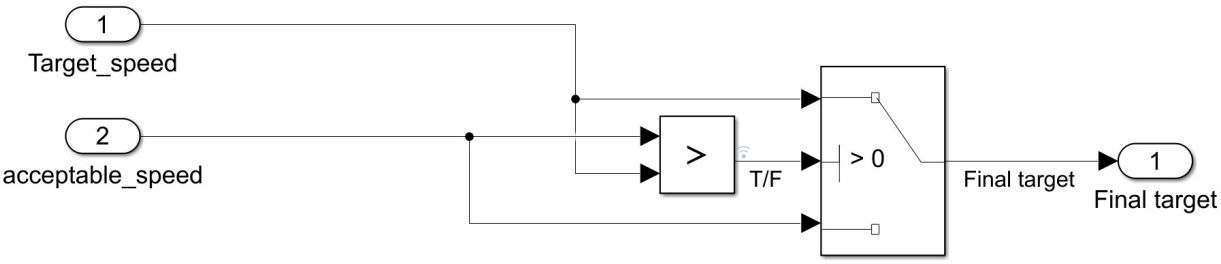

Figure 4.4: An overview of the switch subsystem

This system is composed of a switch commanded by a majority operator fed with the same signals that have to be switched. In Fig 4.5 the inputs used to test the unit are shown that. The two inputs were differentiated as much as possible in order to better understand which of the signals has been chosen As shown in Fig. 4.6 (right), the Simulink test tool

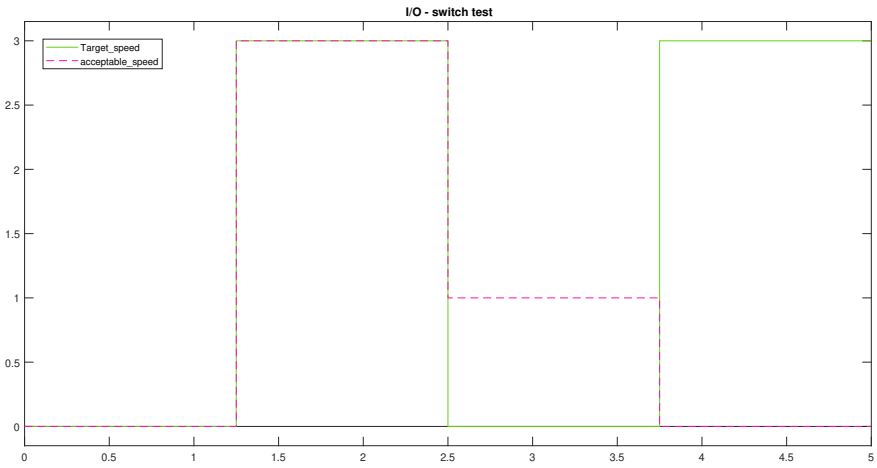

Figure 4.5: Inputs of the test: acceptable speed is the one set by the user, the target speed is the one of the car ahead

is able to generate a document reporting the configuration of the test, to guarantee to the customer that the test was performed accordingly to his needs. This feature also gives the opportunity to an external part to perform the same test and verify that the results are coherent. In the same way the tool can generate a report regarding the output of the test, in Fig. 4.6 (left) is shown that the test produces successful results, in fact the output of the test coincides perfectly with the baseline signal.

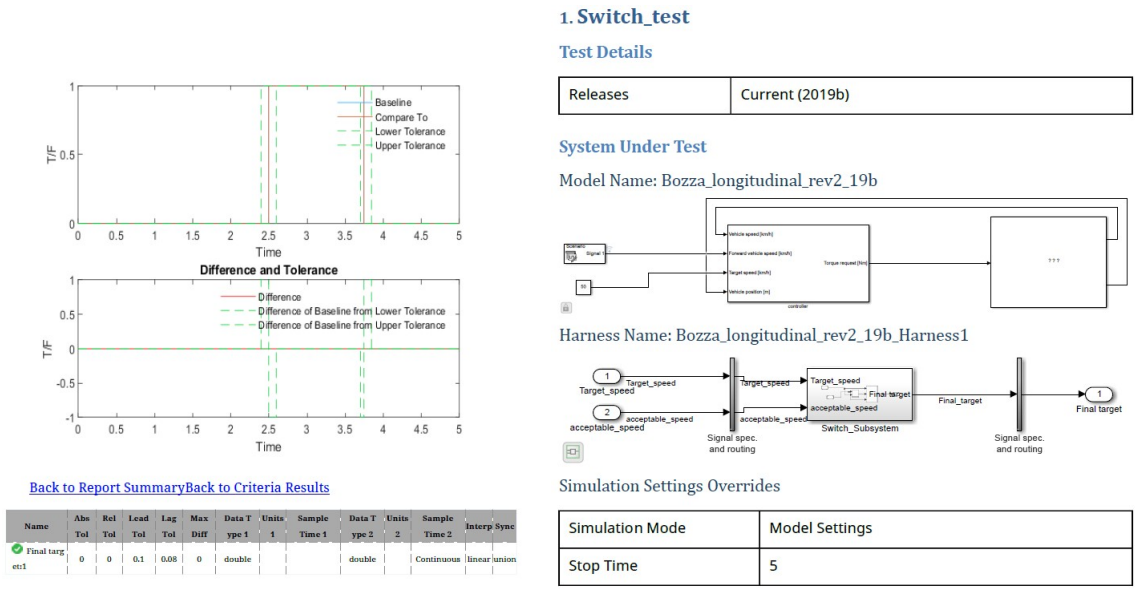

Figure 4.6: Automatically generated test reports

4.2.2 Torque split

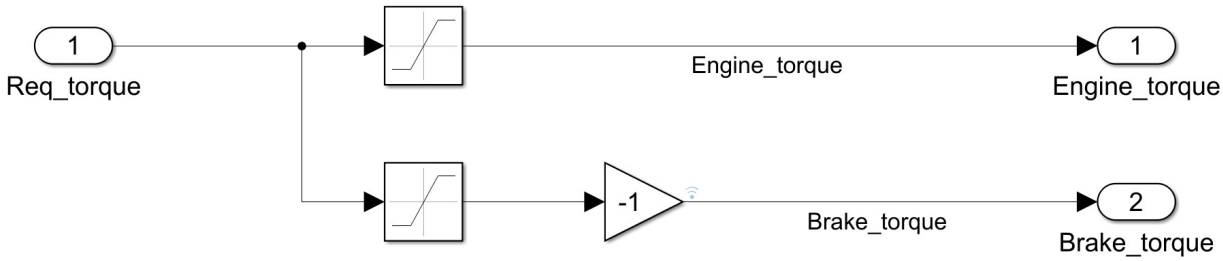

Figure 4.7: An overview of the torque split subsystem

This unit's goal is to split the signal coming from the PID, asking for torque. The PID asks for a negative or positive torque, the system has to be able to translate the negative torque in a signal to be sent to the brake module. To test this unit the following steps have to be performed: when a positive input is given this is transferred to the engine while when a negative input is given it is transferred to the brake output. At the same time the output assigned to the opposite sign has to be zero. In Fig. 4.8 is possible to see how the test input was designed and also that the output corresponds exactly to the baseline signal, meaning a successful test.

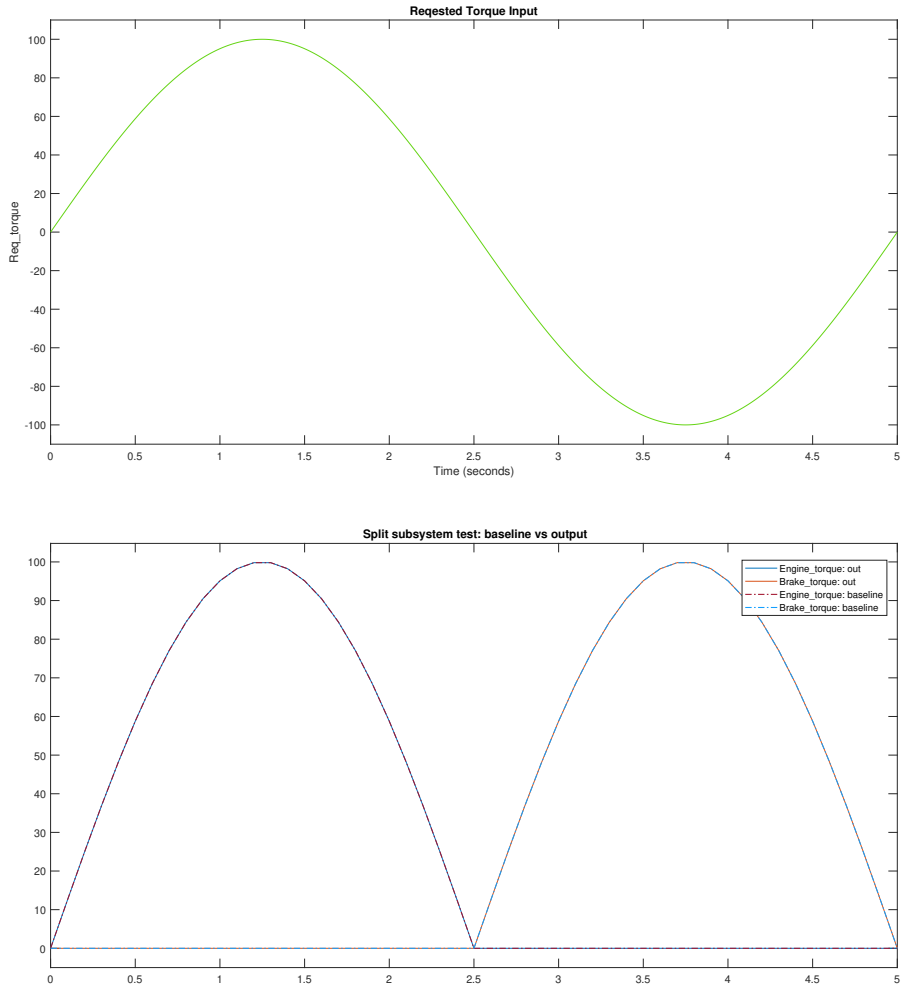

Figure 4.8: Test inputs (up) Outputs vs baseline (down)

4.2.3 SoC handler

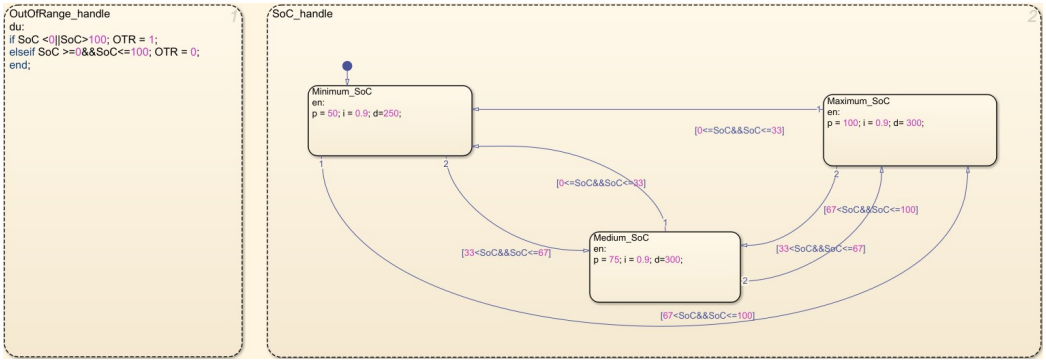

Figure 4.9: An overview of the SoC handler chart

This unit is a FSM, so a slightly more complicated test has to be performed. The goal of this unit is to manage the aggressiveness of the control depending on the SoC of the vehicle's battery pack: a less aggressive control when the car has less autonomy left. The interval of possible SoC values has been devided in three segments: minimum, medium

and maximum. Each segment corresponds to a different state. There are three states, since from each state you can go to the other two there are in total 6 transitions. There is only one input: SoC signal. The input signal, shown in Fig.4.10, is designed in order to stimulate all the possible transitions. To be more robust against unexpected behaviour an additional output has been added for the FSM. The SoC signal is expected to never trespass the 0% and 100% boundaries, so a signal "OutOfRange(OTR)" that rises when this condition is verified was created. The FSM is designed to check constantly the OTR signal while performing the ordinary tasks, so a parallel state is created in the FSM. The

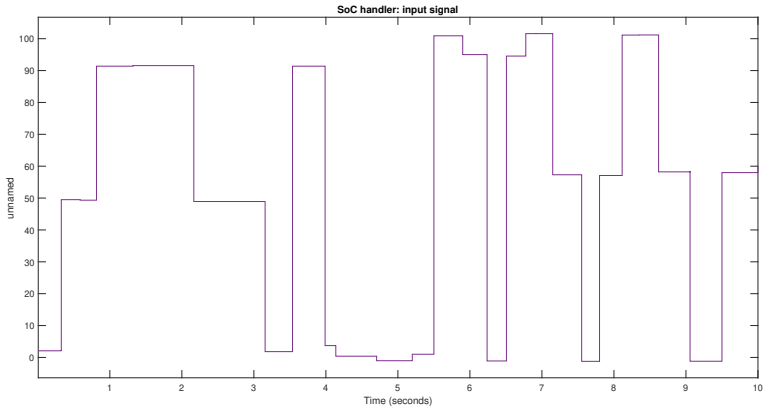

Figure 4.10: Input fed to the SoC handler

behaviour of this signal has been tested too. In Fig.4.10can be deduced from the input signal (second half of the signal) that regardless the current state and the previous value of the SoC signal, the OTR signal raises when expected to. In Fig.4.11 can be seen that the output signal matches the baseline, meaning a successful test.

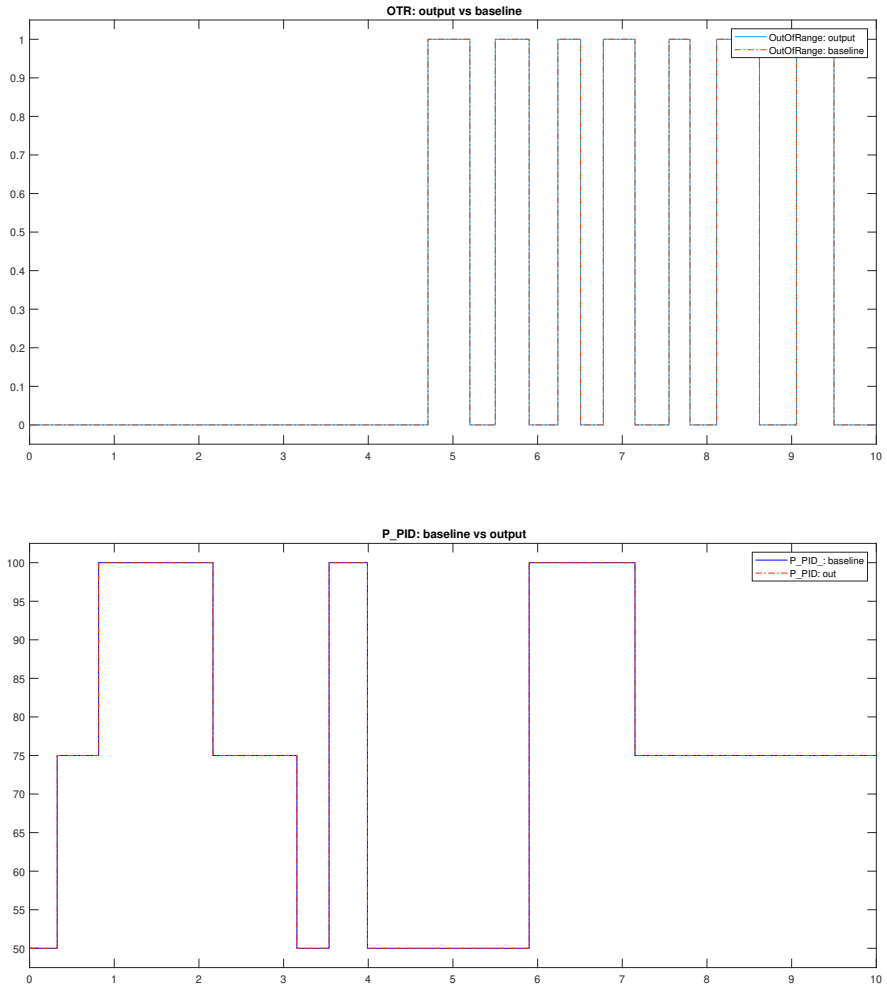

Figure 4.11: OTR output vs baseline (up) PID::P output vs baseline (down)

4.2.4 Integration test

All these units cooperate in the same controller module, so an integration test would be necessary. In this specific case these three units do not have in common any input or output, meaning that there is no possible integration test to be performed. All these units are meant to condition the PID behaviour, so in order to perform an integration test the PID would have to be involved. Since the PID block itself is assumed to be tested by MATLAB there is no point in testing the PID behaviour when not regulating a plant model. According to the previous affirmations we decided to let the integration test coincide with the MIL test, in order to test the behaviour of the whole controller.

4.3 Model-in-the-loop (MIL)

MIL is a type of testing that can be done at the very first stage of the development process.
It is performed before any software generation code and it does not require any dedicated hardware. For all of these reasons, it is very useful because it allows the developers to test their model at the very beginning of the whole process. So, if something fails, time

and cost are significantly reduced.
When validating the model, the first step to be performed is to create the plant of the system. Then, after the design of the controller, the next step is to verify that "the controller is able to control the plant" fulfilling all the requirements that were previously discussed. If this happened, the model is verified and it is possible to continue with the next step (SIL).
In this case, MIL simulation has the aim to verify the Adaptive Cruise Control.
The implemented control law depends on the speed of the leading vehicle and the distance that must be respected between two consecutive vehicles according to the Highway code:

$$d_s = d_{min} + \frac{1}{k}v^2$$

The behaviour of the whole system has been tested by plotting the speed of the first vehicle and the following one.
Since, as mentioned before, the controller takes into account also the SoC value, three different situations depending on the State of Charge of the battery have been plotted. As a result, three different responses are shown:

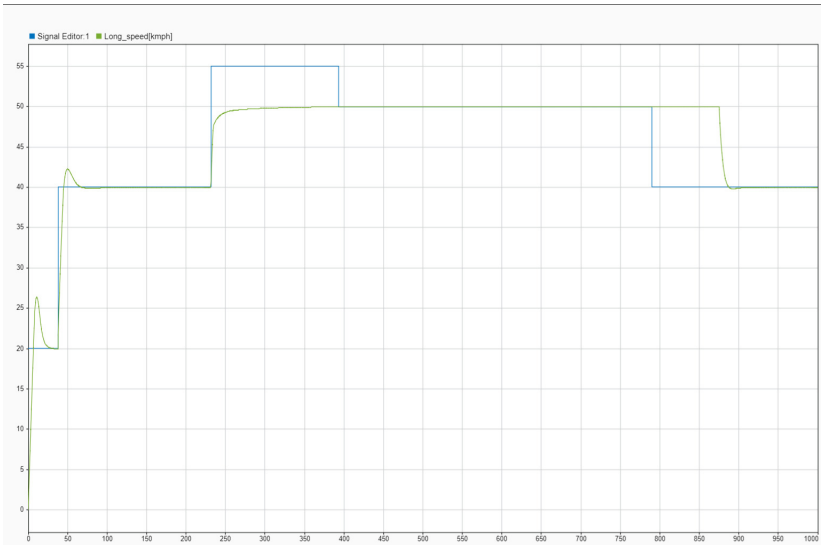

Figure 4.12: Speed of the vehicle for 10 SoC

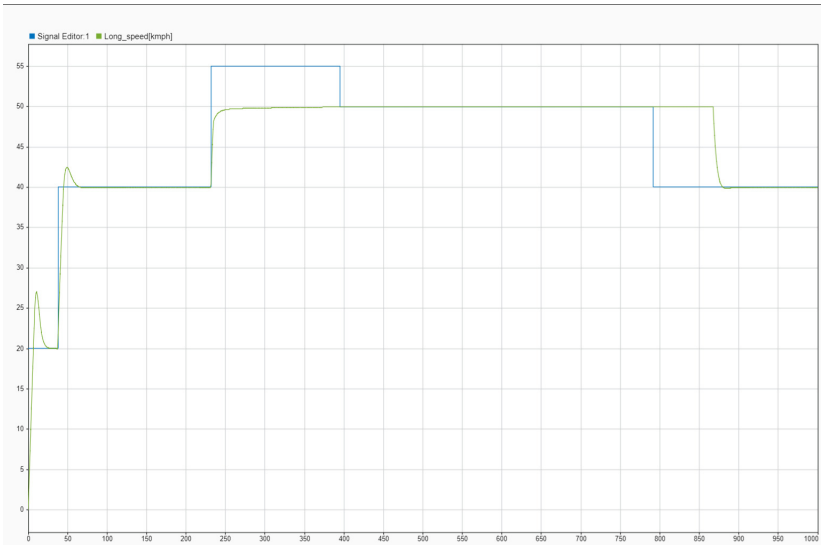

Figure 4.13: Speed of the vehicle for 50 SoC

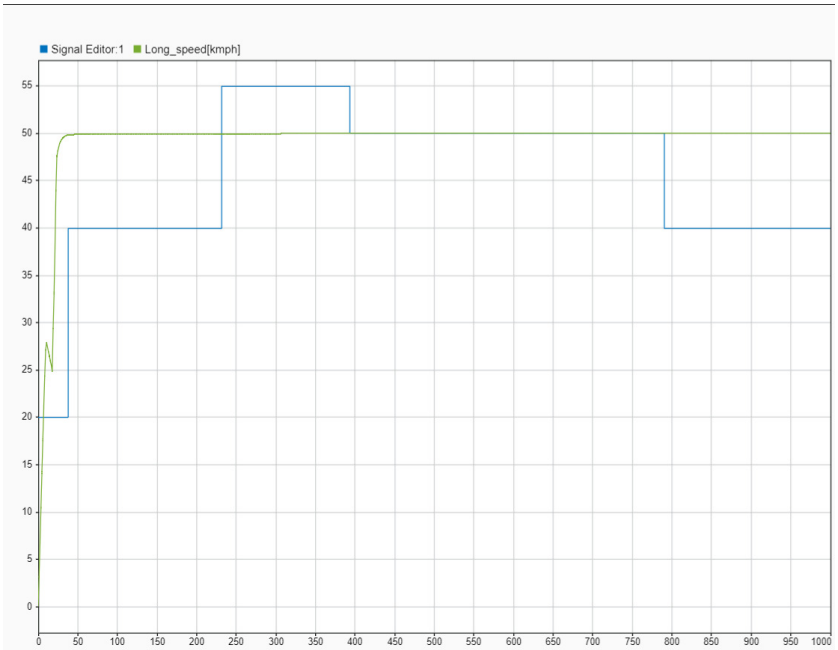

Figure 4.14: Speed of the vehicle for 100 SoC

As can be seen from the graphs, the model in order to fulfils all the requirements previously defined: the speed limit imposed by the driver (50 Km/h in this case) is not overcame.
In addition, a particular time interval has been considered to show how the SoC value influences the controller response.

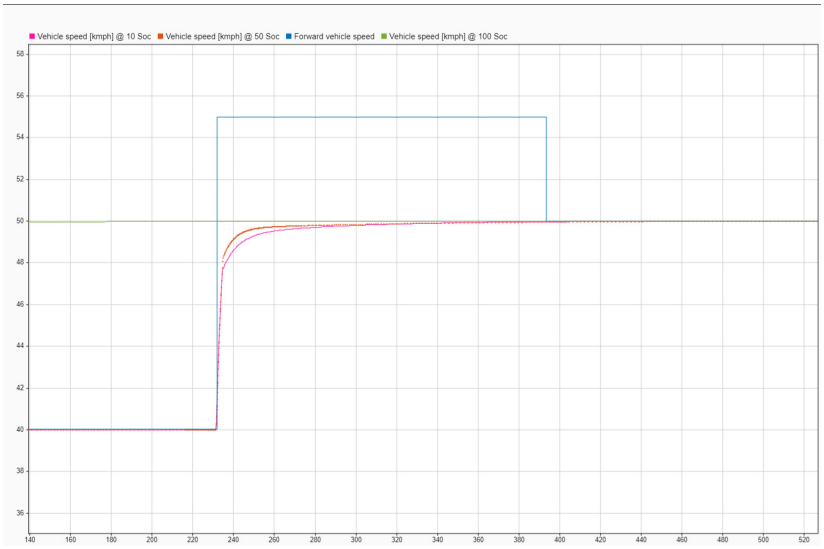

Figure 4.15: Comparison of the speed with different SoC

As can be seen from the graphs, where the SoC is higher, the controller can be more aggressive since it can drain more battery. On the other hand, when the SoC is at its minimum, the controller has a smoother and slower response to preserve the battery charge extending its life.

4.4 Automatic code generation

Embedded Coder is one of the tools used for the automatic generation of code. It is a feature widely used by developers and engineers that automatically generate their code: improves the quality and avoids the errors introduced manually. Embedded Coder generates readable, compact and fast C and C++ code for embedded processors.
The generated code is ISO C compliant, so it can run on virtually any fixed- or floating-point device and is well suited for applications that need to minimize memory usage or maximize speed.
Embedded coder allows to review and navigate from model to generated code and back using code reports with traceability links. It is also possible to verify that the code execution matches model simulation results using software-in-the-loop and processor-in-the-loop testing. SIL and PIL tests can also include code coverage analysis and execution profiling.
Embedded Coder offers built-in support for AUTOSAR, MISRA C®, and ASAP2 software standards. In addition, it offers support packages with advanced optimization and device drivers for specific hardware. Matlab and Simulink offer the **Embedded Coder** to generate C/C++ code from Matlab and Simulink model or subsystem.
To generate the code it is possible to establish the objectives using the Code Generation

Advisor in order to check that the controller is MISRA C compliant.

4.4.1 Preparation of a Model for Code Generarion

At first it is needed to prepare a Model for Code Generation:

1. Open the **C Code** tab, click **Settings** to open the Configuration Parameter dialogue box (or use the shortcut **Ctrl+E**).
   Then select **Simulation** → **Model Configuration parameters**
2. Open the **Solver** pane and select:
   - **Solver type**: Fixed-Step.
   - **Solver**: discrete.
3. Open the **Optimization** pane, and set Default parameter behaviour to Inlined.
4. Open the **Code Generation** pane, and specify ert.tlc as the System Target File.
5. Clear **Generate makefile**.
6. Select **Generate code only**.
7. Enable the HTML report generation by opening the **Code Generation** → **Report** pane and selecting Create code generation report and Open report automatically. Click the horizontal ellipsis and, under Advanced parameters, select **Code-to-model**. Enabling the HTML report generation is optional.
8. Click Apply and then OK to exit.

4.4.2 Generate Code using Embedded Coder

The steps that must be followed are:

1. Open the model that contains the block.
2. In the Code Generation Advisor, click Set Objectives. This is an important step in which it is possible to check the coding standard compliance. There are different objectives that can be selected and it is possible to select and prioritize the combination of objective before the code generation.
3. On the C Code tab, click Quick Start.
4. In the Generate Code step, apply the proposed changes and generate code by clicking Next.
5. Click Finish, then return to the C Code tab. From this it is possible to configure code generation customizations, and then check the results in the Code view next to the model.
6. With the right clock o the component it is possible to build the process and going to C/C++ Code Generation, then click on **Build**.
7. View the code generation report.

4.5 Software-in-the-loop Simulation (SIL)

A software-in-the-loop (SIL) simulation compiles generated source code and executes the code as a separate process on a target host computer. By comparing normal and SIL simulation results, it is possible to test the numerical equivalence of the model and the generated code. Thanks to the **Embedded Coder Toolbox** there are different ways to perform a SIL simulation.
Through a communication channel, Simulink sends stimulus signals to the code on the computer or target process for each sample interval of the simulation:

- Top-model (SIL simulation generates the stand-alone code interface) and Model block. For both, the Test behaviour of generated source code on the development computer. Execution is host/host and non-real time.
- SIL or PIL block (block uses stand-alone code interface). In this case the Simulation runs compiled object code through S-function. S-function communicates with object code executing as stand-alone application on the development computer. Execution is host/host and non-real time.

The method used in this report is the creation of a SIL block.
In the model setting it is needed to use the Embedded Coder in the Code Generation section (ert.tlc).

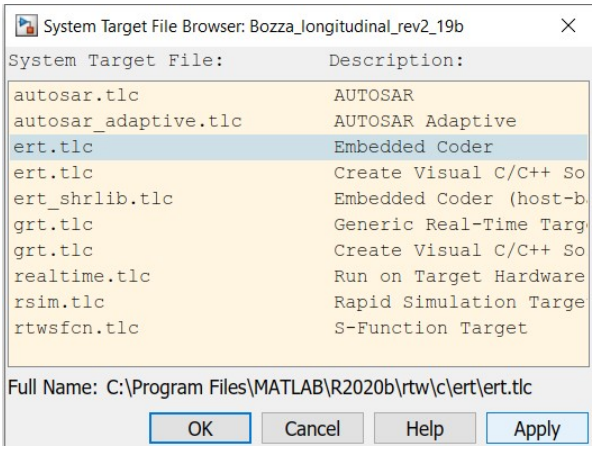

Figure 4.16: Target selection

Then it is possible to check the hardware implementation and check the host architecture on which automatically generated code for the controller will run.
The following step is to check a specific flag in the Code Generation section, in particular in the Verification in which it is possible to select the created block. After that, clicking on the unit, in the C/C++ Code, clicking *Build* creates the SIL version of the Controller subsystem block.
After the creation of the SIL block, a comparison simulation has been carried out. The entire plant has been simulated and then it is possible to observe the error between the two different simulations.

The following steps summarize the creation of a SIL block:

1. From the Configuration Parameters → Code Generation → Verification → Advanced Paramters → Create block drop-down list, select the SIL block.

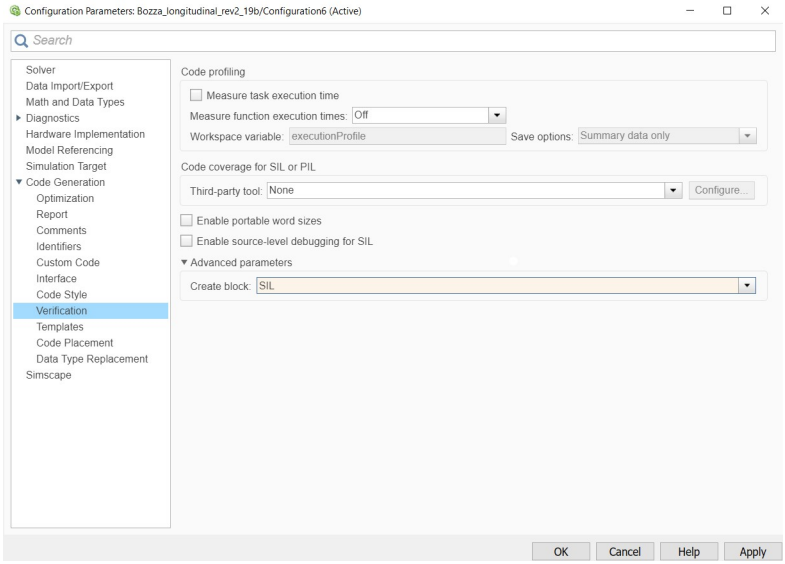

Figure 4.17: Creation of the SIL block

2. Click OK.
3. In the model window, right-click the subsystem that it is need to simulate.
4. Select C/C++ Code → Build This Subsystem.
5. Click Build, which starts the subsystem build process that creates the SIL block for the generated subsystem code.
6. Add the generated block to the environment.
7. Run the simulation with the environment.

The simulation results are represented in the following figure:

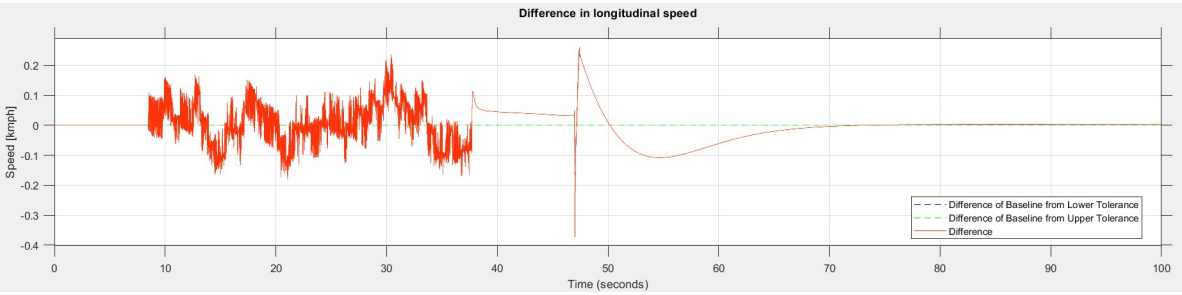

Figure 4.18: Difference in longitudinal speed of the original Simulink model and the SIL model

It is possible to see that the difference between the two simulations is negligible, so the whole result is acceptable.

4.6 Processor-in-the-loop Simulation (PIL)

The chosen hardware for the simulation is the Raspberry Pi 4 model B, equipped with quad-core processor (ARM-cortex), wirelessly connected to the PC. To provide a stable connection between hardware and Simulink it is necessary to set properly the device address (IP address) and the credential of the OS system for privileged read/write operation.

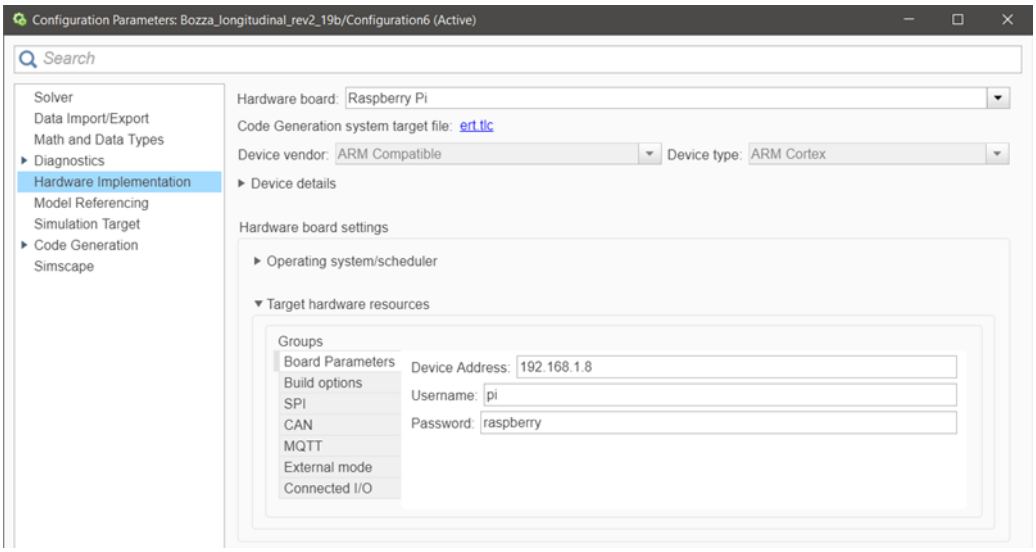

Figure 4.19: Hardware settings

The generation of the code is widely described above on the SIL section and the procedure for generation of the PIL block is the same as for SIL.

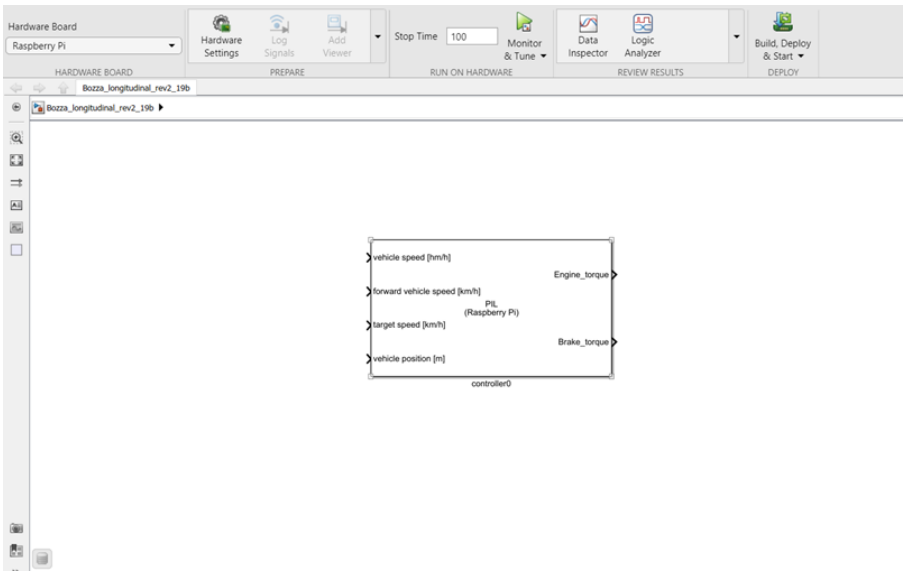

Figure 4.20: PIL block

Once the controller has been replaced by PIL block, the entire code is deployed on Raspberry board and the simulation starts.

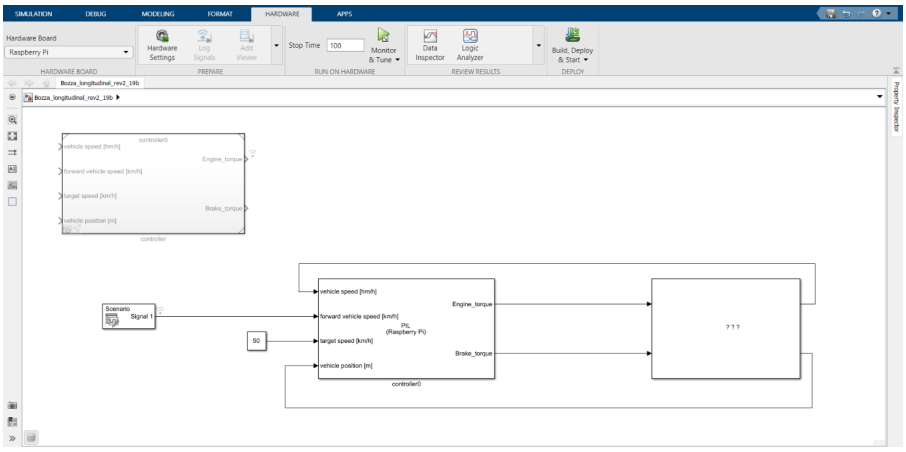

Figure 4.21: Simulation PIL block

From the comparison simulation, it can be noticed the difference between the longitudinal speed of original Simulink model and the one of PIL model.

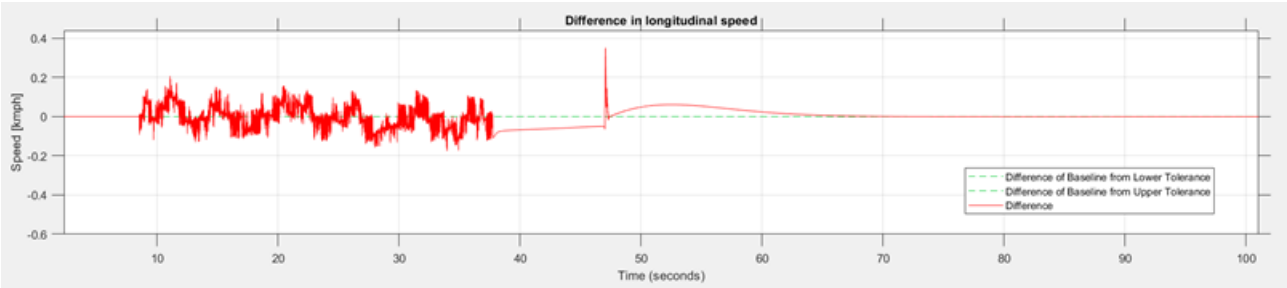

Figure 4.22: Difference in the longitudinal speed of the original Simulink model and the PIL model

The overall result is acceptable. The difference between the two simulations is negligible and bounded in a very short range.

Chapter 5

Conclusion

The designed controller is quite simple and cheap in terms of control design (a PID controller is the core of the control law). It can be easily integrated into a more complex structure thanks to the used modular approach. Future improvements would consist of using the OTR signal for more complex control actions, like some checks or the disengagement of the whole system. The system can be integrated with other ADAS systems like an emergency brake system or a collision-avoidance system (alternatively a simpler overtake assist). The system can not be directly deployed into a passenger vehicle as it is. Its interface by the way is structured in order to be integrated in a more complex ECU software: in modern ECUs there are various systems able to request torque, these signals are handled by a high level controller that dispatches the most adequate torque signal coming from one of the various subsystems, accordingly to predefined policies. The presented document illustrates how powerful the model-based design approach can be. Following the V-model tool, the designer is able to guarantee a rigorous step by step workflow. Thanks to the followed standardized workflow the output product could be sold to a final user that would have all the needed documentation to assess the correct behaviour of the system.