POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering (Ingegneria Meccatronica)

Master's Degree Thesis

Novel Hardware Verification Techniques for Electric Vehicles

Advisor:
Prof. Massimo Violante

Candidate:
Marco D'Auria

Co-advisors:
Prof. Jacopo Sini

Academic Year 2019/2020

CONTENTS

1. INTRODUCTION

Embedded systems are heavily used in any aspect of automotive applications. Nowadays, vehicles integrate more complex functionalities, that's why the demand for Electronic Control Units (ECU), able to perform safety-relevant tasks, is increased. Some of these functions are defined as safety-critical because, in case of faults, could lead to anomalous vehicle reactions with risk for passengers and pedestrians safety. This pushes the automotive industry to invest in new verification, and validation methodologies act to improve the quality of the assessment and, at the same time, to decrease the time-to-market.

The development process of electronics and electrical (E/E) components is related to the automotive functional safety standard ISO 26262:2018. This standard gives information to estimate, through the Hazard Analysis and Risk Assessment (HARA), the critical system level. Moreover, it provides a guideline for all the phases of the item life, above all for the hardware/software integration validation, since, usually, the software is in charge to mitigate the effects of some possible hardware failures.

Today the industrial companies must design their E/E components to be integrated into the vehicle, according to this standard, and they cannot overlook it. It describes the way that they shall approach all the phases of the design of E/E units for automotive usage.

This thesis proposes a simulation-based technique to support the designers to perform the Failure Mode, Effect, and Diagnostic Analysis (FMEDA), in order to assess the capability of the hardware design, and so. to achieve the ISO 26262 reliability requirements.

It proposes to simulate the hardware reaction, injecting the faults; then, these have been extended to the vehicle-level through a vehicle dynamics simulator. In this way, it is possible to make a failure effect classification, considering the effects on the dynamics and driveability of the vehicle. It can be particularly useful in cases where the behaviour of the item is strongly coupled with the one of the whole vehicle.

In particular, the case study is a power electronics module embedded in an electric vehicle powertrain. It considers a rear traction car with two independent electric motors, one per each wheel. The presence of this automotive architecture on the rear allows steering the vehicle using a virtual differential gear that has a strong safety impact on the driveability of the car.

The approach results able to get useful output for the functional safety engineers' job.

2. AUTOMOTIVE STANDARD ISO 26262

Since 2011 the ISO 26262 [1] standard provides how to shall design, validate and test the item in such a way that it is functional safety. It was updated in 2018 to be applied to all road vehicles except for mopeds and aftermarket parts for vehicles designed to be operated by drivers with disabilities.

The standard is structured into different parts. Each one of them regulates a specific phase of the lifecycle of items that have to be deployed inside a road vehicle.

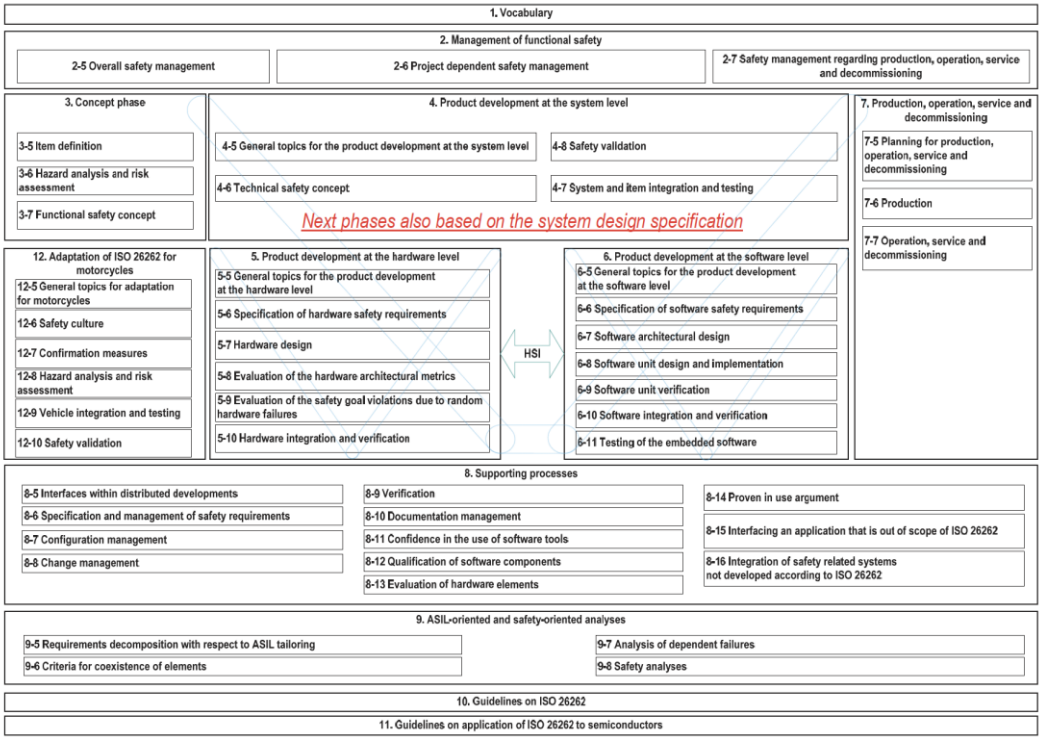

Figure 1: Overview of ISO 26262 standard.

In general, the standard is divided into eleven parts (Fig. 1), as follows:

1. Vocabulary;
2. Management of Functional Safety;

3. Concept phase;
4. Product Development: System Level;
5. Product Development: Hardware Level;
6. Product Development: Software Level;
7. Production and Operation;
8. Supporting Processes;
9. ASIL-oriented and Safety-oriented Analyses;
10. Guidelines on ISO 26262;
11. Guidelines on application of ISO 26262 to semiconductors.

Starting from the "Vocabulary" that defines all terms will be used in the standard, there is the first phase: "Management of functional safety". It describes how to choose conventions, tools, and workflow to be adopted at the company level for all design, production, and decommissioning activities. After that, several phases take care of understanding the specific purposes of the item that is going to design. They are arranged, according to the V-cycle, like a "Concept Phase". At this stage, it is possible to understand which is the ASIL levels of the various safety goals related to the vehicle function provided by the item. Once the level of criticality of the system is known, it proceeds to "Product development level" for the hardware and the software. In these phases, the procedures define guidelines about:

- how to address the definition of the specification of components both HW and SW;
- which are the interface between them;
- how it has to proceed with the design of HW and SW components;

- how to validate each part;
- how to integrate them together;
- how to test them together and so on.

When the design is completed and validated there is “Production and operation”, here the validation activity is finished so the item can be manufactured and shipped to the customer.

2.1. Vocabulary

In order to better understand the following paragraphs, it is convenient to define some keywords:

- *Item*: system or array of systems or a function to which ISO 26262 is applied;
- *Harm*: physical injury or damage to the health of people inside and around the vehicle [2];
- *Severity*: a measure of the extent of harm to an individual; how critical the system is. It is classified into four categories (Figure 2):
    1. S0: No injuries;
    2. S1: Light and moderate injuries;
    3. S2: Severe and life-threatening injuries (survival probable);
    4. S3: Life-threatening injuries (survival uncertain), fatal injuries.

Class	S0	S1	S2	S3
Description	No injuries	light and moderate injuries	Severe injuries, possibly life-threatening, survival probable	Life-threatening injuries (survival uncertain) or fatal injuries
Reference for single injuries (from AIS scale)	AIS 0 Damage that cannot be classified safety-related, e.g. bumps with roadside infrastructure	more than 10% probability of AIS 1-6 (and not S2 or S3)	more than 10% probability of AIS 3-6 (and not S3)	more than 10% probability of AIS 5-6
Informative examples	-Pushing over roadside infrastructure, e.g. post or fence -Light collision -Light grazing damage -Damage while entering or leaving a parking space -Leaving the road without collision or rollover			
-Side collision, e.g. crashing into a tree (impact to passenger cell) 15 < Δv < 25 km/h		Δv <15km/h	15 < Δv <25 km/h	Δv >25 km/h
Side collision with a passenger car (impact to passenger cell)		Δv <15km/h	15< Δv <35 km/h	Δv >35 km/h
Rear/front collision between two passenger cars		Δv < 20km/h	20< Δv <40 km/h	Δv>40 km/h,
Other collisions		-Scrape collision with little vehicle to vehicle overlap (< 10%)		-Roof or side collision with considerable deformation
Under riding a truck		Without deformation of the passenger cell		With deformation of the passenger cell
Pedestrian/bicycle accident			E.g. during a turning manoeuvre inside built-up area	Outside built-up area

Figure 2: Examples of severity classification [4].

- *Failure*: termination of the ability of an item or an element to perform a function as required;
- *Fault*: physical damage for the hardware part or programming errors (bugs) appearing during the software execution;
- *Risk*: a combination of the probability of occurrence of harm and the severity of that harm [3];

- *Controllability*: it is the capability of the driver to avoid the injury or the damage through a timely reaction in a hazard situation. It is arranged into four classes:
    1. C0: Controllable in general;
    2. C1: Simply controllable;
    3. C2: Normally manageable;
    4. C3: Difficult to control or uncontrollable.

    In Table 1, the four classes are better classified in terms of driving factors and scenarios:

Table 1: Examples of controllability classification.

Class	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable
Driving Factors & Scenarios	Controllable in general	99% or more of all drivers or other traffic participants are usually able to avoid harm	90% or more of all drivers or other traffic participants are usually able to avoid harm	Less than 90% of all drivers or other traffic participants are usually able, or barely able to avoid harm
Situations that are considered distracting	Maintain the intended driving path			
Unexpected radio volume increase	Maintain the intended driving path			
Warning message - gas low	Maintain the intended driving path			
Unavailability of a driver assisting system	Maintain the intended driving path			

- *Exposure*: it is a measure of the probability of being in an operational situation that can be hazardous if coincident with the failure mode under analysis. It is divided into five groups:

    1. E0: Incredible;
    2. E1: Very low probability;
    3. E2: Low probability;
    4. E3: Medium chance;
    5. E4: High probability.

    Classification in terms of duration (percentage of average operating time), road layout and the road surface is shown in figure 3:

Class	E1	E2	E3	E4
Description	Very low probability	Low probability	Medium probability	High probability
Definition of duration/ probability of exposure	Not specified	< 1% of average operating time	1% - 10% of average operating time	> 10% of average operating time
Informative examples	Highway – lost cargo/obstacle on road Mountain pass – driving down hill with the engine off Jump start Garage – vehicle on roller rig	Pulling a trailer Driving with roof rack Driving on a mountain pass with an unsecured steep slope Snow and ice Driving backwards Fuelling Overtaking Car wash City driving – driving backwards City driving – parking situation Country road – crossing Country road – snow and ice Country road – slippery/leaves Highway – entering Highway – exit Highway – approaching end of congestion Parking – sleeping person in the vehicle Parking – parking with trailer Garage – diagnosis Garage – vehicle on auto lift	Tunnels Hill hold Night driving on roads without streetlights Wet roads Congestion City driving – one way street Highway – heavy traffic/stop and go	Accelerating Braking Steering Parking Driving on highways Driving on secondary roads City driving – changing lane City driving – stopping at traffic lights Country road – free driving Highway – free driving Highway – changing lane Parking – parking lot

Figure 3: Examples of exposure classification [5].

- *Automotive Safety Integrity Level (ASIL)*: it is a label attached to the design in order to specify the necessary requirements to apply to the item, for avoiding an unreasonable residual risk. All the ASIL levels are reported in the following paragraphs;
- *Hazard*: a potential source of harm;
- *Hazard Analysis and Risk Assessment (HARA)*: method to identify and categorize hazardous events of the item and to specify safety goals and ASILs related to the prevention or mitigation of these hazards in order to avoid unreasonable risk;
- *Safety*: the absence of unreasonable risk;
- *Safety Goal*: essential safety requirement as a result of the hazard analysis and risk assessment.

2.2. Concept phase

The concept phase process (Fig. 4) starts from the definition of the item that is going to design, through a series of steps leads to the functional safety concept. This part specifies which are additional functionalities to add to the item in order to guarantee the safety of the vehicle function implemented by the item.

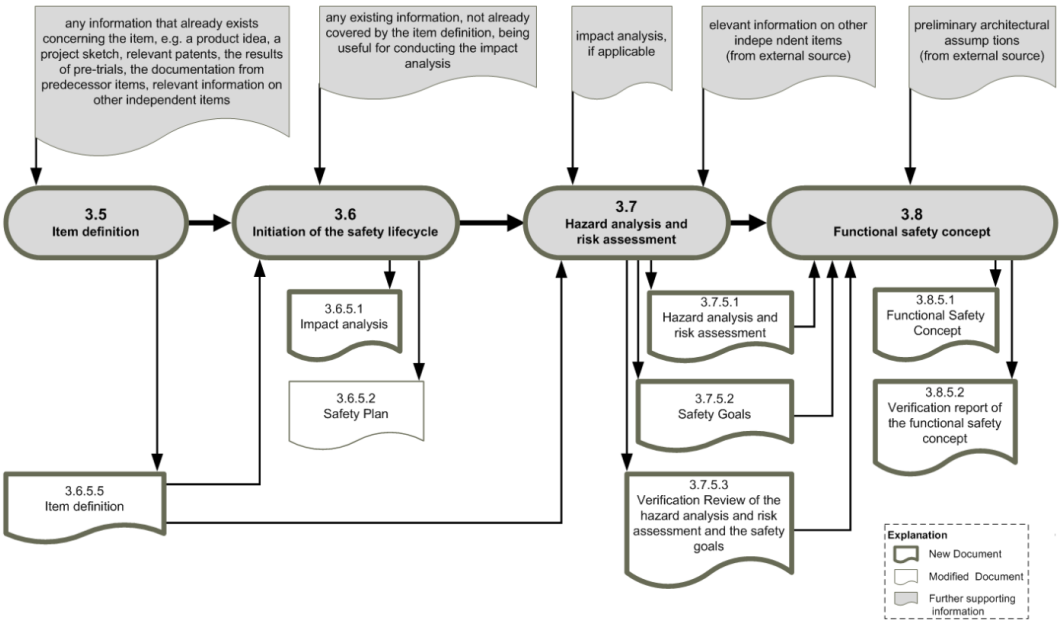

Figure 4: Process steps in the ISO 26262 - Part 3 [6].

In the "Item definition", it has to describe what vehicle functions the item is in charge of. It has to be specified, which is the functionality that the item exposes, understanding which are the electrical/electronic elements that concur in delivery of this functionality, such as sensor, network and so on. It doesn't care about mechanical parts and E/E components that are outside the item. Moreover, it has to understand which are the functionalities provided/required to/from other items, elements or the environment.

In the "Initiation of the safety lifecycle" block, it has to define the safety lifecycle activities making the distinction between the development of new item and a modification to an existing one.

In the "Hazard analysis and risk assessment" stage, it has to identify and categorize hazards that could result from malfunctions of the item. In this part, it can understand what happens if an item component got broken, so that the specific

functionality that is depending on that component cannot be delivered and, depending on when the failure happens, it decides if there is a risk for the user or not. This part is not based on the mathematical model so there is not a coded or standard procedure to address it, there are few steps that are suggested, but they require many experts and brainstorming in order to produce usable results. They are:

1. "Situational analysis" describes operational situations and operating modes in which the item could be used and where a malfunction can show up as a hazardous event. For example, driving at low speed, driving at high speed and so on;
2. "Hazard identification" means that it has to identify which could be misbehaviour that the item is exposing as a result of a fault. There are appropriate techniques to do that; one kind entirely qualitative is the "Failure Mode Effect Analysis" (FMEA). Systematically, for each component of the item, it has to try to guess which are the failure modes that could affect that component and, based on that, which are the effects of that failure mode on the item and the functionality that the item provides;
3. "Hazard classification" for each operational situation, it has to consider all the possible hazards, and it has to classify each combination of them in terms of severity, exposure and controllability. The hazard analysis is quite tricky because the outcome and the weights that are assigned to the different parameters depend on what is happening on the specific functionality the item has to provide. So, there is not a golden rule when it has to approach on this analysis, and it always has to apply the methods analysing carefully

the functionalities, the operational situations, and hazard that are taken in that particular item;

4. “ASIL determination” is assigned based on the combination of severity, exposure, and controllability. It is obtained through figure 5:

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Figure 5: ASIL Determination based on Severity/Exposure/Controllability levels [7].

Where “QM” is an acronym of quality management, it is not an ASIL level where to spend time and money to mitigate the fault, and it can be neglected because nothing wrong happens out of that. Then, the least important level is “A” instead the most important one is “D”. In this last level, the risk is very high, so it must intervene in implementing, inside the item, some functional safety functionalities that mitigate this risk.

Depending on the functionality, it has to take into consideration very carefully, the different aspects of severity, exposure, and controllability. When it did the work with one item, that work is not valid for any possible applications of that item, because it can have different functionalities that

are added to the item from time to time. Although it did the work once, it doesn't mean that it will stay the same at any time, so it has to evaluate carefully where the item is used.

5. "Safety goal determination" is the objective that it wants to reach in order to mitigate the risk associated with the obtained ASIL.

   If the hazard in a specific operational situation is safety-relevant from A to D, then it has to put at work measures to mitigate the risk. This is quite tricky because the objective, the goal to be reached, is not necessarily an additional functionality that it is put on the top of the functionalities needed for delivering the mission for which the item is intended. The objective could be something that relates to the development flow. In this way, the mitigation works applying specific techniques when doing the design and validation to be sure that the risk is mitigated through the design process and not through techniques that are inserted inside the item.

6. "Functional safety concept" is additional functionality that the item has to expose to guarantee the safety of the people involved. It has to specify, if appropriate, which are the functionalities to put at work for reaching the defined safety goal. This is the case when a higher ASIL level is found so that, besides the methodology to apply during the development flow, it must put also thinks at work inside the item. There are specific architectures or design techniques to adopt for the item, in particular, the redundancy.

2.3. Product Development: Hardware Level

ISO 26262 design process starts with hazard analysis and risk assessment (HARA) activity at the item level. As the output of this phase, designers obtain the item Automotive Safety Integrity Level (ASIL). According to the obtained ASIL, the standard prescribes to assess the sources of possible failures, which are the effects of these failures on the behaviour and safety of the item. In particular, Failure Mode and Effect Analysis (FMEA) [8] is a technique to be used for accomplishing such a task, and it is strongly recommended for ASIL C and D items.

Another technique for the hardware integration verification is the Failure Mode, Effect, and Diagnostic Analysis (FMEDA). FMEA is performed during the concept phase and has a higher level of abstraction. Its purpose is to find how the item can fail. It is possible to perform this analysis since the Concept phase. The FMEDA, instead, is performed at a lower level of abstraction (SPICE level and model of the embedded software), in order to take into determining the random hardware failure metrics. The item has to achieve metrics suitable for the strictest ASIL level of its safety goals.

Moreover, the implemented Diagnostics and mitigation strategies have to be assessed. This phase, like the FMECA, has to be performed during the hardware design phase (part 5). Of course, if the embedded software is in charge to detect or mitigate an HW failure, it has to be developed in compliance with the ISO 26262 part 6 for the item ASIL level.

The common methodology analyses the failure modes of each component that is part of an item to rank all the possible failures of all components. The FMEA results and target system reliability are included in the FMEDA assessment results [8], as the following figure (Fig. 6).

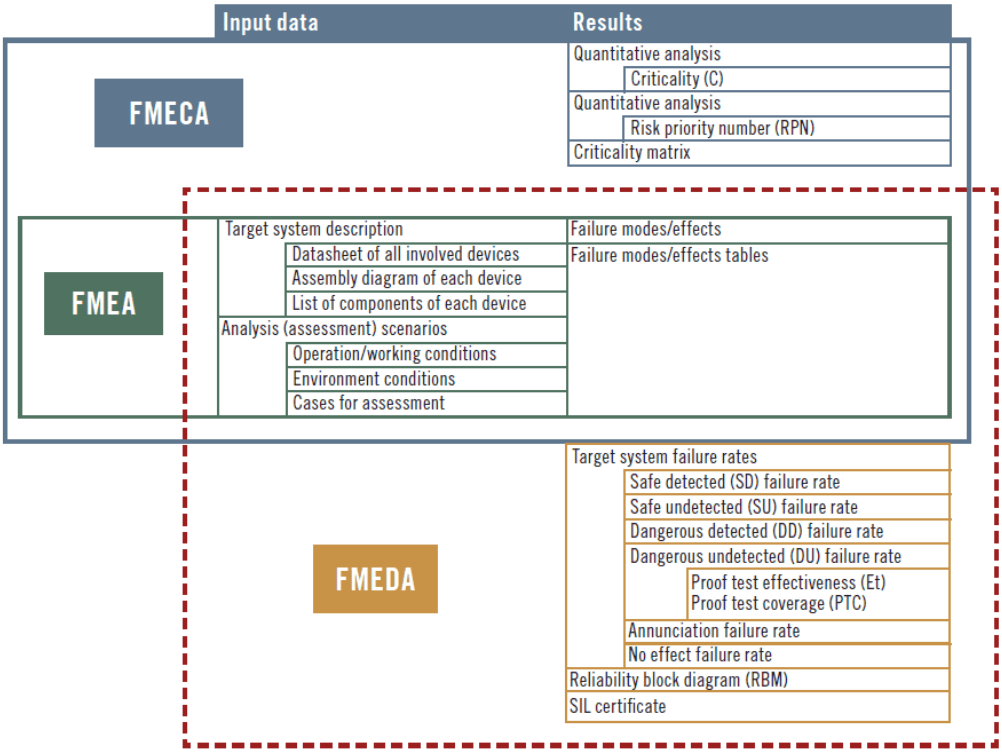

Figure 6: Failure mode analysis Venn diagram [9].

At the end of the design verification phase, engineers have to provide the item robustness. It has to be summarized by three metrics: "Random hardware fault" rhf, "Single point fault metric" $spfm$, and "Latent fault metric" lfm.

It has to define the following rates for a given fault f [10] to compute these three metrics:

- Failure rate, λ^f: is the occurrence rate of the fault f expressed as Failure-In-Time (FIT), that is the number of predicted failures in a billion hours;
- Safe Detected (SD) rate, λ^f_{SD}: the rate of faults that are detected through the functional safety mechanism embedded in the item;
- Safe Undetected (SU) rate, λ^f_{SU}: the rate of faults that are not detected through any of the functional safety mechanism embedded in the item, and they do not provoke any harm to the item user;
- Dangerous Detected (DD) rate, λ^f_{DD}: the rate of faults that are detected through the functional safety mechanisms embedded in the item;
- Dangerous Undetected (DU) rate, λ^f_{DU}: the rate of faults that are not detected through any of the functional safety mechanisms embedded in the item, and they provoke harms to the item user.

They are classified in descending order of severity as dangerous undetected (DU), safe undetected (SU), dangerous detected (DD) and safe detected (SD). The SU cases are more stringent than DD ones because the system cannot predict possible misbehaviour if another safe undetected fault happens. Instead, for the dangerous detected cases, after a fault detection, the driver or another electronic control unit (ECU) can mitigate the failure limiting potential risks.

One technique for evaluating the sensitivity of systems to failures is the fault injection. It consists in inoculate misbehaviours that emulate the failures and observing how the system react to them when it is stimulated by various workloads [11][12]. Historically, this technique was applied to hardware for the dependability validation [11], failure prediction [13], and validation of functional verification

methodologies. With the increase in the complexity of the software, it is increasingly difficult to test and validate it, which is why nowadays the fault injection is also applied to the software.

Once all possible faults are injected for every component, there is the failure classification in terms of safe/unsafe.

The calculation method used to prove the robustness of the hardware design shall provide for each failure mode affecting each component of the design, some estimation for λ^f, λ^f_{SD}, λ^f_{SU}, λ^f_{DD}, λ^f_{DU}; from these rates, it is possible to define the following quantities.

The item failure rate can be expressed according to the equation below, assuming all faults independent:

$$\lambda = \sum_f \lambda^f$$

The single point fault rate is defined as:

$$spf = \sum_f \lambda^f_{DU}$$

As it can see, the single point faults cause a violation of the safety-goal, and there is no safety mechanism implemented in the item to detect these faults.

The residual fault rate is defined as:

$$rf = \sum_f \lambda^f_{DD}$$

They are faults that are not covered by an implemented safety mechanism.

The latent fault rate is defined as:

$$lf = \sum_{f} \lambda_{SU}^{f}$$

They are faults that do not violate the safety goal but do it if another fault occurs.

Given the above rates, it is possible to estimate the three metrics; in particular, the random hardware fault metric is defined as:

$$rhf = spf + rf$$

The single point fault metric is defined as:

$$spfm = 1 - \frac{spf}{\lambda}$$

It represents the effectiveness of the safety architecture to protect from individual faults.

The latent fault metric is defined as:

$$lfm = 1 - \frac{lf}{\lambda}$$

Once the metrics are computed, the hardware design verification is completed when they are fulfilled with the requirements imposed by ISO 26262 for the given ASIL. Below there is a small figure (Fig. 7) that reports an overview of requirements, in term of metrics, for different ASIL levels.

Metric	ASIL B	ASIL C	ASIL D
Probabilistic Metric for Random Hardware Faults (PMHF)	10^{-7} h^{-1} (100 FIT)	10^{-7} h^{-1} (100 FIT)	10^{-8} h^{-1} (10 FIT)
Single Point Fault Metric (SPFM)	≥ 90%	≥ 97%	≥ 99%
Latent Fault Metric (LFM)	≥ 60%	≥ 80%	≥ 90%

Figure 7: Overview of requirements for different ASIL-Levels [14].

2.4. Product development: Software Level

Software faults are programming errors that appear during the execution of the software. The main problem is that possible bugs within the code can be sources of damage for both the driver and the people around the vehicle. For example, unintended acceleration of the vehicle can be a source of casualties or not brake the car when it is required to do so.

For these reasons, the ISO 26262 standard also provides a guideline for software development. Even this phase is quite tricky because the software is responsible for mitigating the risk. In particular, a block diagram is provided with the main steps to follow; everything is regulated by the V-cycle (Fig. 8)

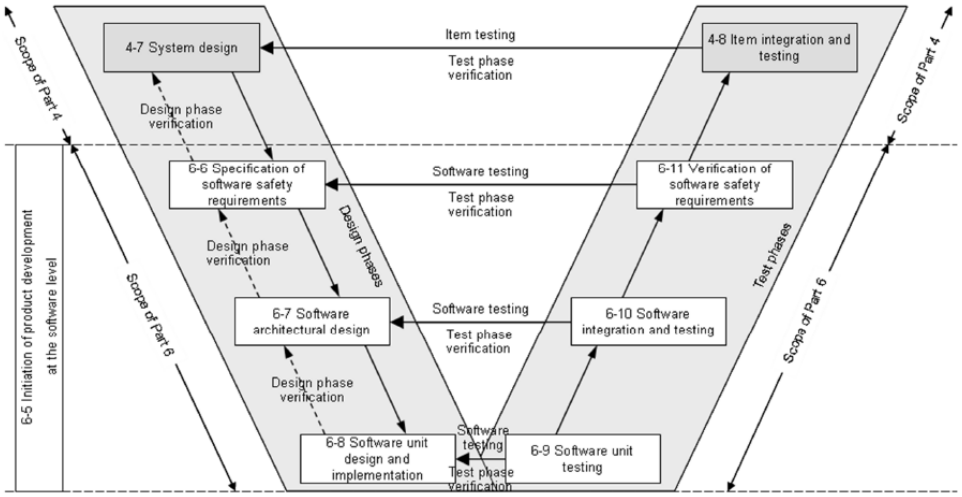

Figure 8: Software Lifecycle V-model [15].

It always will have a top-down approach where it starts from the design phase branch and then move on to the test and verification branch. Initially, the system design is developed, as seen previously, the entire system is divided into sub-components, and each of them will be designed and implemented.

Once the design is completed, it moves on the ascending branch that consists of doing verification and testing. The individual sub-components are first tested alone. If the result is positive, the different pieces of code are put together and check that each sub-component communicates appropriately with the other. Then, the whole system is tested. This is in according to the V-shape development flow where the descending branch consists of doing the design, and the ascending one consists of doing verification/testing.

2.5. Safety life cycle

The safety life cycle (Fig 9) is a flow chart that describes the flow operations summarizing what has been said previously.

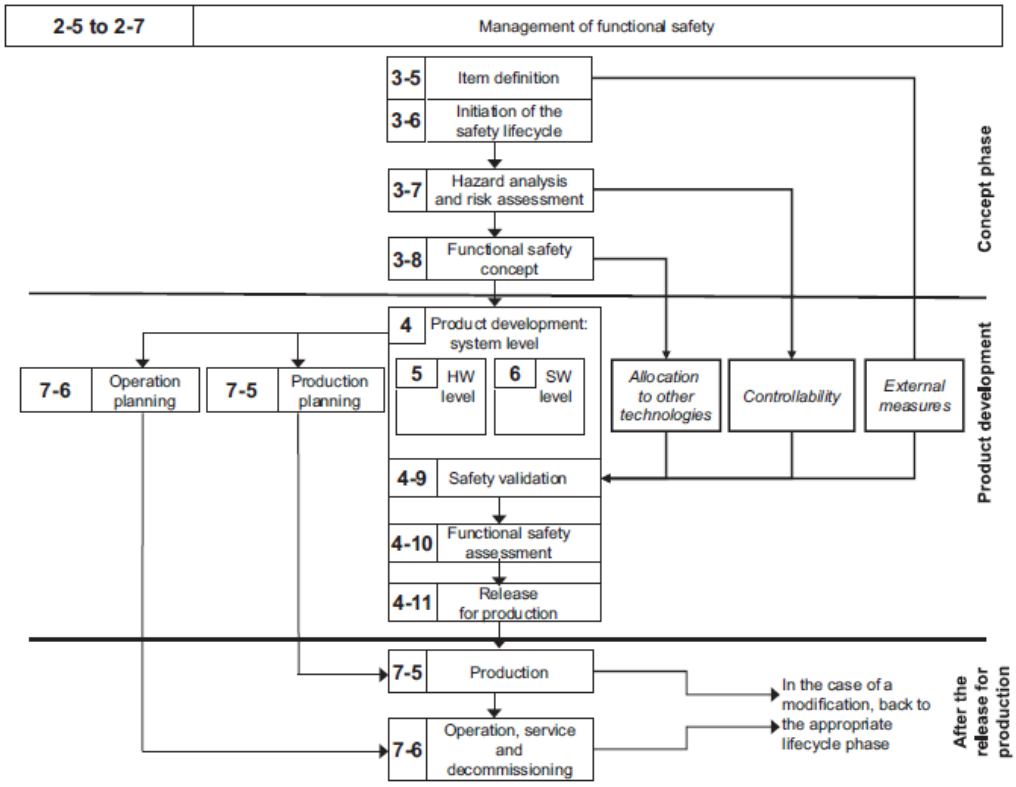

Figure 9: Safety life cycle [16].

At the beginning, there is a "concept phase" that starts from the definition of the item that is going to design, through a series of steps leads to the functional safety concept. Here, it specifies which are the additional functionalities to add to the item to guarantee that is functionality safe.

Once defined this concept, it passes on the "product development" phases. These are the phases when the hardware and software are designed that will implement

the functionalities for the item shall provide, including functional safety. The ISO 26262 standard establishes that the development of the intended functionalities for the item and the development of the functional safety-related functionality is done concurrently. First of all, it defines what the item shall do, and it partitions the functionality in hardware and software, then for each component HW and SW it performs a fared refining of the design so that using a *"divide et impera"* approach, in this way functionality is partitioned in sub-functionalities.

They are designed and implemented individually, according to the "V model", and then they are merged in order to implement the final hardware and software. After that, everything is ready to integrate them to obtain the final artefact that is the output of the "product development" phase. Once the design has been completed, there are the "production" and "operation, service and decommissioning" steps.

One aspect that it is relevant throughout all the standard is the fact that it is a clear model of operations. It is based on a few steps that it has always applied:

1. Declare which are the safety goals and the relative ASILs for the vehicle function that you want to implement by the item, this can be done through documents that have different scope;
2. Implement what it has been declared; basically, it has to put at work all the techniques that are suggested within the standard to obtaining the object declared;
3. After the implementation of a particular phase of the standard, it has to check what obtained is consistent with the one declared.

If repeat systematically this structured approach, it is able to find problems hazard as possible. For this reason, it is essential that the item implementation must be done after all the validation phases.

3. PROPOSED METHODOLOGY

Failure Mode, Effect, and Diagnostic Analysis (FMEDA) for hardware designs verification and validation methodology is strongly recommended by ISO 26262 for ASIL C and ASIL D items.

In the industrial practice, the FMEDA analysis starts from the bill of material (BOM) and the circuit schematic of the hardware to be verified. After analyzing the circuit, it is required to compute the component failure rate, using data from hardware part failure rate catalogues, like IEC 61709 [17], and reliability calculation models, like FIDES [18]. The ISO 26262 classifies the possible effects of each possible failure mode of each one of the components installed on the board into four different classes. These are safe detected (SD), safe undetected (SU), dangerous detected (DD) and dangerous undetected (DU).

Usually, designers inspect the item hardware schematics manually, and through an inductive analysis based mainly on their knowledge and expertise, they perform, for each fault, the classification.

This approach suffers from two shortcomings. With the growing complexity of embedded systems employed in safety-critical applications, manual inspection of the item schematic can become ineffective in identifying all the possible misbehaviours. This method is efficient to analyze those systems which have interactions with the physical environment, where the cause-effect relationship is well defined. While, in those systems where this interaction becomes complicated, a perfect knowledge of the item cannot be sufficient to classify the effects of the

failure mode. Moreover, the manual approach cannot take in detailed consideration the contribution of embedded software. Often, the hardware to be verified includes microprocessors or microcontrollers, and the manual fault effect analysis does not take into detailed consideration on how the software contributes to the fault propagation.

3.1. Simulation-based approach

To overcome the lack of objectivity and repeatability of manual hardware design inspection, it would like to propose another approach to perform an FMEDA analysis automatically [10]. It is based on a simulation-based criterion as a CAD tool used to validate and develop an item until it is able to reach all the required safety objectives.

The automatic tool needs:

- A model of the device under test (DUT), implemented, from the circuit netlist, as a Simulink model using the SimScape toolbox to simulate the electrical component. This model should follow precisely the topology for the integrated circuit pins and the actual printed circuit board;
- A fault catalogue for the components present in the DUT bill of materials (BOM), with the relative FIT values computed during the Reliability, Availability, Maintainability, and Safety (RAMS) analysis;

- A software tool able to manage, in an automatic manner, the simulations, inject the failure and classify the failure effects on the safety goals of the item (as dangerous or safe).

The tool is fully implemented resorting to the MATLAB/Simulink environment. The design of item hardware has to be modelled through the MathWorks Simulink SimScape toolbox, while the software is implemented as a MATLAB function. Even the fault list generator, the saboteur, and the classifier modules have to implement for the same environment as MATLAB script. In this way, it is possible to obtain a single executable model that capture the relevant characteristics of both hardware and software.

Some aiding tools, to improve the validation and verification of the hardware, have been proposed in the past [19][10][20]. However, they are intended only for the classification of the failure effects at item(actuator)-level. For many automotive functions, finding item-level classification criteria can be thorny, due to the strong coupling between the item local failure effects and vehicle behaviour.

This thesis would like to investigate the effect of a fault-affected item on the vehicle behaviour. In this way, it is possible to assess also the expected effects on the whole car propagating the behaviour of these faults to a vehicle-level simulator. Of course, to identify and classify them, this method needs to define classification rules at the vehicle-level.

In this work, it is adopted an actuator-based perspective where it is only considered the failures that propagate to the actuators. Under this hypothesis, all the failure

effects can be propagated from the inside of the item to the actuator without losing generality [21].

A specific platform is required to perform the FMEDA by the simulation-based approach. It is composed of [22]:

- the embedded software of the item;
- the physical models of the item (at printed board circuit level) considering the fault-free and the fault-affected conditions, needed to perform the SPICE-level simulations;
- the physical model of the car and the surrounding environment, provided by the vehicle-level simulator;
- the physical model of the sensors and the controlled actuators;
- scenarios in which test the failure effects;
- vehicle behaviour classification rules.

The following accessory components are also required:

- fault list generator module;
- saboteur;
- circuit (item-level) simulator;
- vehicle-level simulator;
- failure effect classifier.

The whole system architecture is represented in the following figure (Fig. 10):

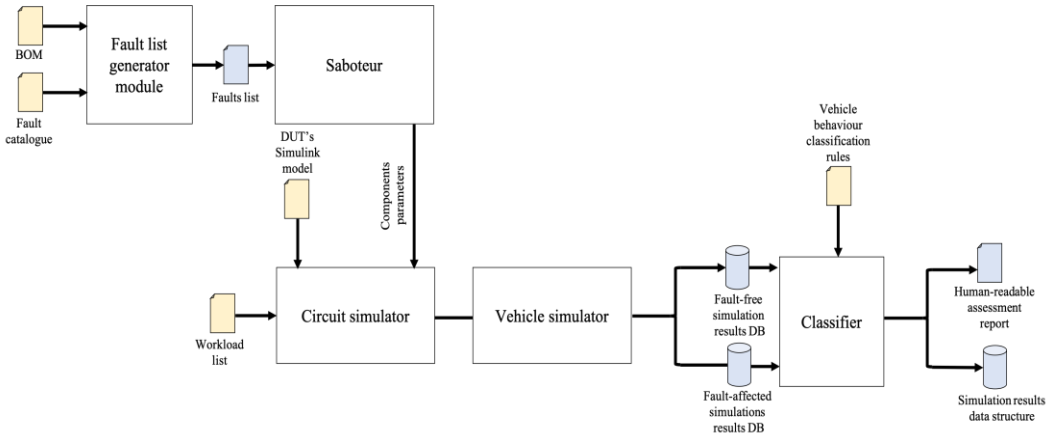

Figure 10: Tool software architecture [20]

The environment works as follows. It starts from the bill of materials (BOM) looking the hardware schematics, and a fault catalogue (for example [18], with the probabilities, computes as described in [17]). By combining these two documents, it is possible to generate a faults list for each component. The saboteur receives this failure modes list as input which will use it to inject the faults during the simulation. The circuit simulator takes a Simulink model of the device under test. It can be instrumented with dummy elements to allow to inject particular failure modes (for example, block parameters with a constant value changing during the simulation or switch to simulate open or short circuits).

Moreover, the circuit simulator needs also a workload list representing the conditions where to assess the failure mode effects. These conditions can be shared with the vehicle-level simulator to create the same scenario. During the simulation, the circuit simulator interacts with the vehicle simulator, taking the input signals from the latter and generating the outputs for the actuators, closing the control loop. At this point, the simulation results can be stored and passed to a classifier. It

compares the system outputs in fault-free (golden condition) with the ones obtained after the failure inoculation (fault-affected condition) according to the vehicle behaviour classification rules provided. Instead, the detection/undetected classification is obtained directly from the simulation implemented, inside the item, as the failure detection system.

As a single failure mode only is considered during the FMEDA, the saboteur injects one by one the faults into affected component, since the successful detection is part of the analysis.

In this work, the silicon-level faults that could affect the microcontrollers (MCUs) are not considered, since the modern automotive-grade MCUs integrate fault detection and mitigation mechanisms [23][24]. So, they have to be regarded as Safety Element out of Context (SEooC) [1].

3.2. The case study

To better describe the simulation-based approach, it has been considered an electrically powered vehicle (EV). In particular, a rear-wheel-drive electric vehicle with two independent motors, one for each wheel. This automotive architecture allows better control on the car since the torque on the wheels can be varied to take into account the radius of curvature that the driver intends to travel, in a differential-drive like fashion. Therefore, the implementation of a virtual differential should be considered. It also allows recovering weight, since the presence of the mechanical

differential gear and the two-axle shafts that connect the latter to the wheels are not required. Moreover, each motor has to supply half of the required power.

As a counterpart of all these impressive characteristics, such a system must guarantee a high level of reliability, because a failure on one motor can cause a disparity torque on the rear axle, making it very difficult for the driver to keep control of the car on the travelled trajectory. Fortunately, the embedded software is in charge to detect motor failures, and it must be able to take action to minimize the torque disparity.

3.3. Simulation system set-up

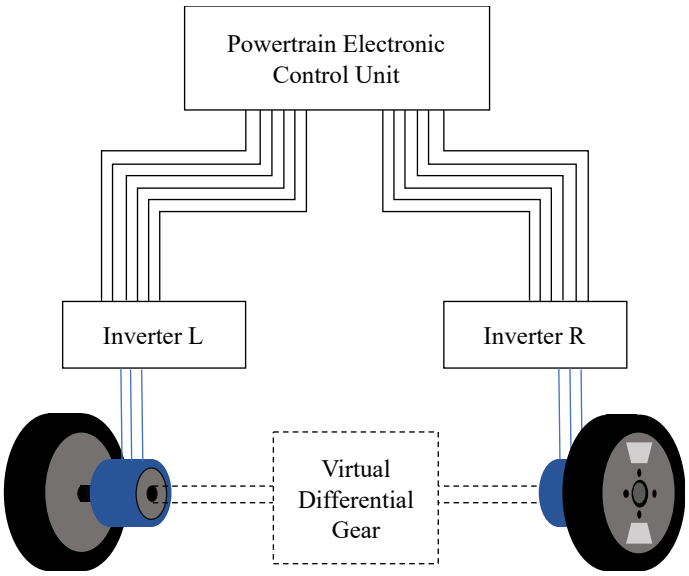

Figure 11: Structure of the rear dual-motor axle of the car [22].

The benchmark application (see Fig. 11) is composed of:

- the embedded software of the dual-inverter system;

- the physical model of the inverters, with their fault model to be injected;
- the physical model of the motors;
- the physical model of the car and the surrounding environment, provided by the vehicle-level simulator;
- scenarios in which test the failure effects;
- vehicle behaviour classification rules.

As said in the 3.1 section, the following components are also required:

- fault list generator module implemented as a MathWorks™ MATLAB™ script;
- saboteur implemented as a MATLAB™ script;
- circuit simulator, resorting to MathWorks™ Simulink™ with SimScape™ toolbox, to perform the SPICE-level simulation of the design;
- an off-the-shelf vehicle-level simulator (CarSim);
- failure effect classifier implemented as a MATLAB™ script.

3.4. Failure mode analysis

For the sake of this work, it has been considered only the faults that could affect the analogue components installed on the PCBs of the Powertrain Electronic Control Unit and of the two inverters. Therefore, it is identified 68 failure modes for the considered item:

- 30 regarding the gas pedal position acquisition chain circuitry;

- 2 regarding the power supply;
- 36 are about the two motors actuation chains.

For each motor, there are 18 possible failure modes:

- 6 regarding the triple redundancy encoders installed to monitor the wheel angular speed;
- 12 are about power electronics (inverter).

An inverter is composed of 6 Insulated Gate Bipolar Transistors (IGBTs), and each one of them can remain stuck at closed (short-circuit) or open condition. In case of a short circuit of an IGBT, there is no possible software mitigation solution because the fuses will melt down disconnecting the phase from the battery. For this reason, it has always been considered only open circuit failure. Due to the symmetries of that system, the faults are injected only on an IGBT of the left motor.

3.5. Detection and mitigation algorithms

The detection and mitigation algorithms are part of the benchmark application to develop.

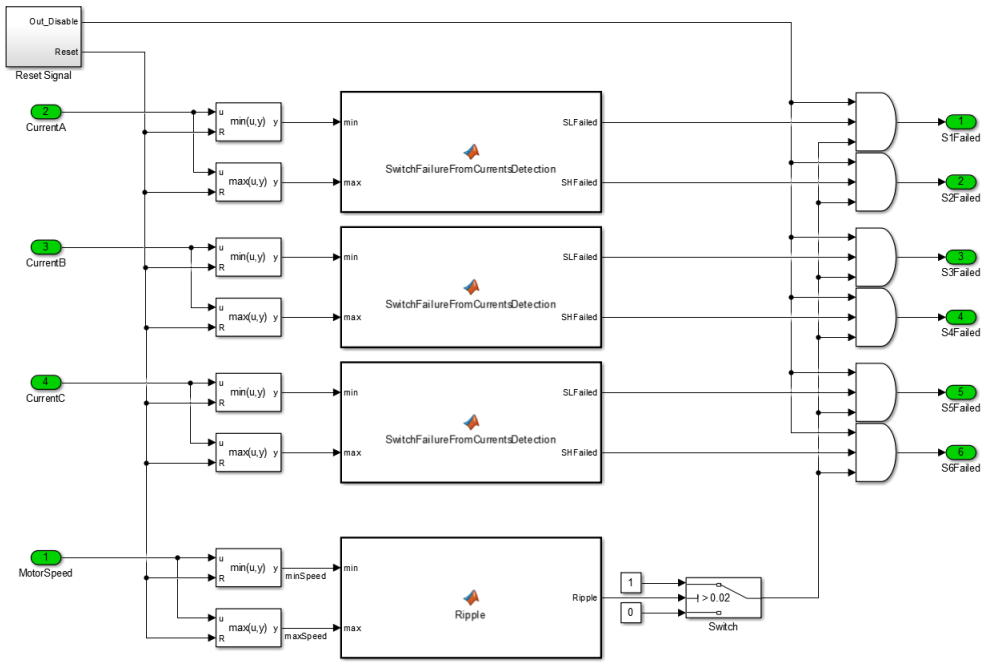

Figure 12: Failure detection algorithm.

The detection algorithm (Fig. 12) is based on a comparison between the current on one of the three phases with the minimum, maximum values of the other two. If the disparity is higher than 80%, a fault into the considered leg of the inverter is detected. If the difference is on the minimum values, the failed IGBT is the one connected to the negative pole of the battery; otherwise, it is the one connected to the positive pole. The MATLAB code is reported below:

```
{function [SLFailed, SHFailed] =
SwitchFailureFromCurrentsDetection(min,max)
    SLFailed=0;
    SHFailed=0;
    if(-min<max &&-min< 0.8*max)
       SHFailed=1;
    end
    if(max<-min && max< 0.8*(-min))
       SLFailed=1;
    end
end
```

In this way, it is possible to understand if a fault has occurred inside the inverter. Downstream of the block diagram shown in figure 12, there is another stage (Fig. 13) such that it recognizes which is the damaged switch and sets the *Detected* signal to *true*.

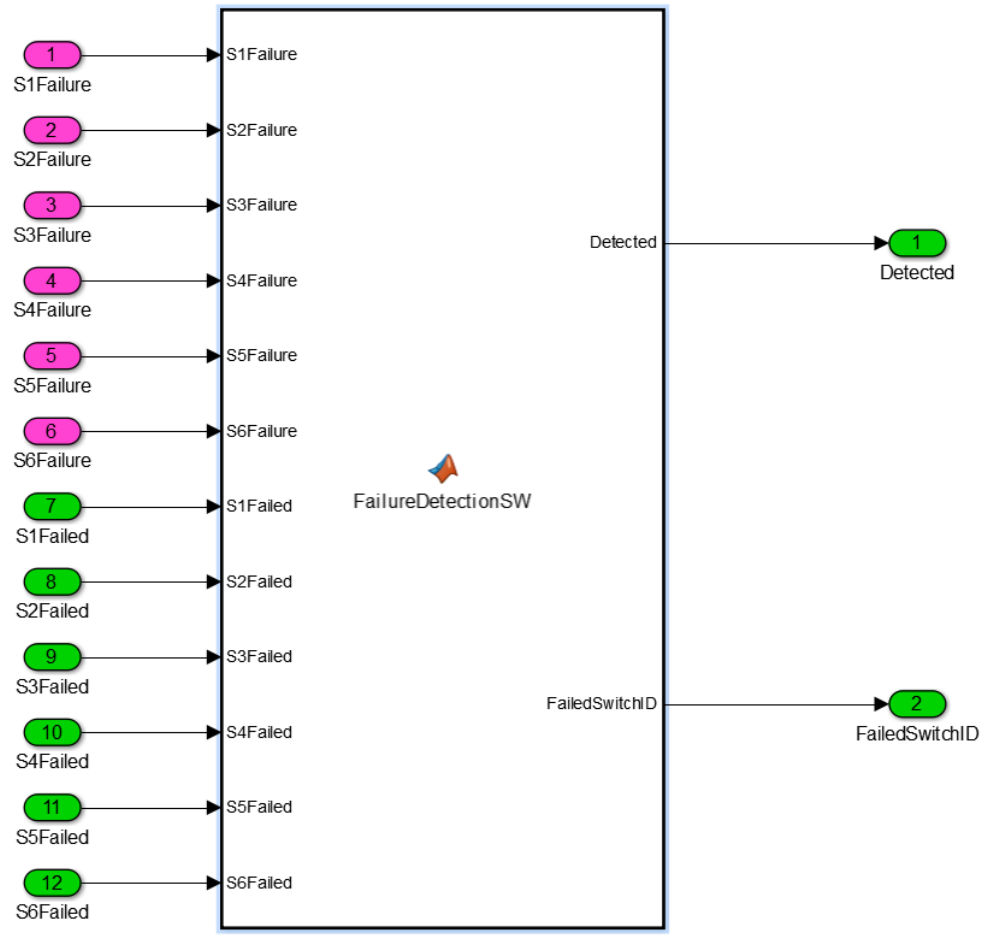

Figure 13: Failure detection algorithm part that recognizes which is the damaged switch.

The mitigation strategy is based on the assumption that the motor driven by the inverter with the broken IGBT is not more able to provide the full torque expected to produce. Therefore, since only one of the two motors is affected by this failure, and the asymmetrical torque gives the most critical situation from drivability point of view, so it could be possible to intervene on the fault-free motor.

A semi-formal representation of the algorithm is shown in figure 14:

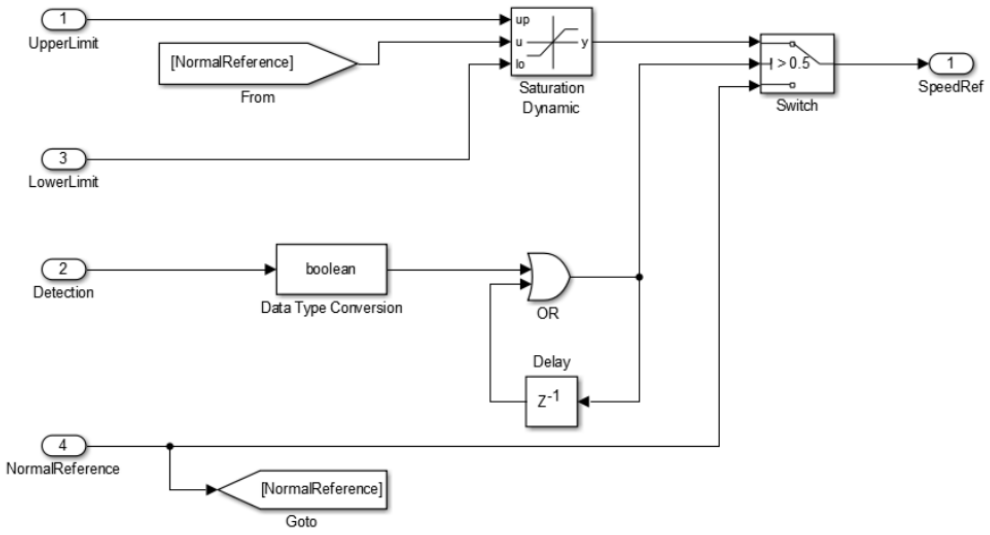

Figure 14: Semi-formal (MathWorks Simulink™) model of the mitigation algorithm.

UpperLimit and *LowerLimit* signals are equal to the speed of the fault-affected motor, *Detection* comes from the detection algorithm shown in figure 14, and *NormalReference* is the speed request from the driver during the simulation. Once a failure occurs, the *Detection* input signal switch on "true" and the driver's speed request is saturated up to the *UpperLimit* in case of forwarding direction or LowerLimit in case of reverse direction. In order to remain in a safe state, the *OR* element and the delay are required, even if the detection algorithm stops to detect a failure.

4. EXPERIMENTAL RESULTS

Before starting to analyze the results of the simulations, it is better to provide some more information on the vehicle-level simulator and item model. CarSim 9.0, produced by the company Mechanical Simulation Corporation, has been chosen because it includes datasets containing many vehicles and test procedures that can be used in the simulations. It offers a graphical user interface (GUI) to be able to configure scenarios, vehicle characteristics and environmental conditions. In addition, it provides Application Program Interfaces (APIs) as well in order to connect itself with third-party software. Thus, it is possible to extract the signals of interest, compute the item behaviour, and to close the loop: at this point, it is possible to apply the actuators command to the simulator.

4.1. CarSim environment

CarSim delivers the most efficient, accurate, and detailed methods for simulating vehicles performance [25]. The core of this vehicle-level simulator is composed of a set of differential equations to simulate the vehicle dynamics and an optimized solver. A graphical user interface is shown in figure 15.

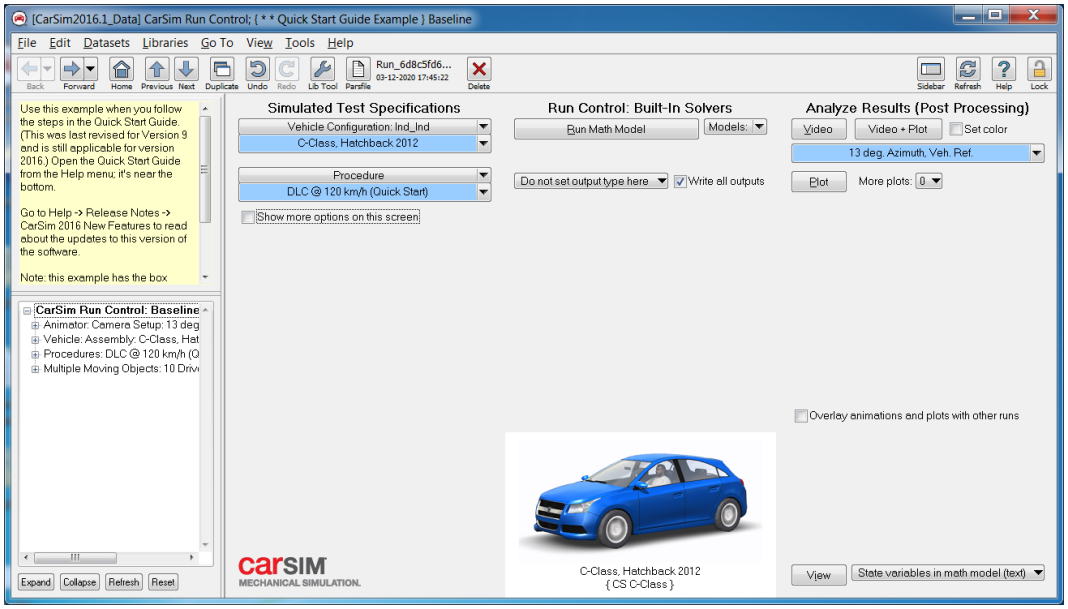

Figure 15: The CarSim Run Control screen.

The main part of the screen is divided into three sections:

1. the left column, titled *Simulated Test Specifications*, regards to the vehicle data and the test procedure;
2. the middle column, titled *Run Control*, takes into consideration controls for running the CarSim math models;
3. the right column, titled *Analyze Results (Post Processing)*, is used to visualize the simulation results and to provide access to the video.

A small picture of the vehicle chosen for the simulation is shown below (Fig. 15).

4.1.1. Simulated Test Specifications

The simulated test specifications are done by *Vehicle configuration* and *Procedure*. The former allows to create a new vehicle or change an already existing vehicle.

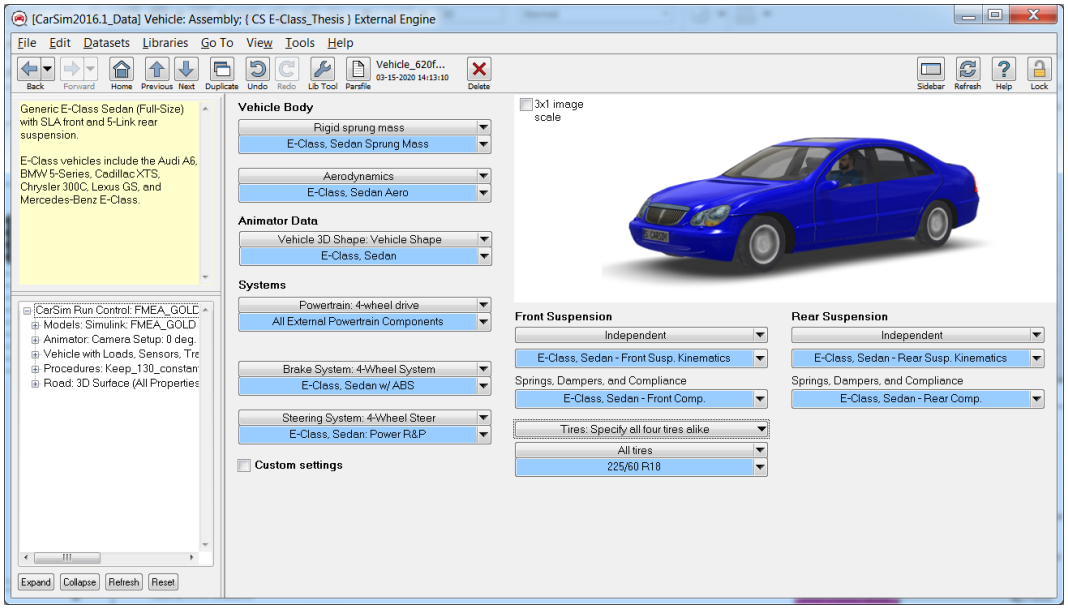

Figure 16: Vehicle Assembly screen

As shown in figure 16, it consents to modify:

- the vehicle 3D shape like length, height, width, sprung mass for the suspension and so on;
- the powertrain like engine model, gearbox, transmission and differential;
- the brake and steering system;
- front and rear suspensions like independent, solid-axle and twist-beam for the rear;
- tire specifications.

Moreover, in figure 16 is shown the type of vehicle chosen for the simulations; it is a class E sedan.

The *Procedure* (Fig. 17) is used to set the scenario and some driver controls for the simulation, for example:

- *brake control*, setting a braking pressure or a braking pedal force, like a constant, ramp and so on;
- *shifting control*;
- *steering mode*, setting the steering torque or a specific function, like following a particular path, keep a constant position. In the latter cases, the simulator implements the right steering angle to be applied;
- *3D road*, to change the geometry (like the elevation) and friction of the reference path;
- *start and stop conditions*, to stop running the simulation at a specific time or station or to set only the initial time;
- *plot definitions*, choosing the plot to show during and the end of the simulation.

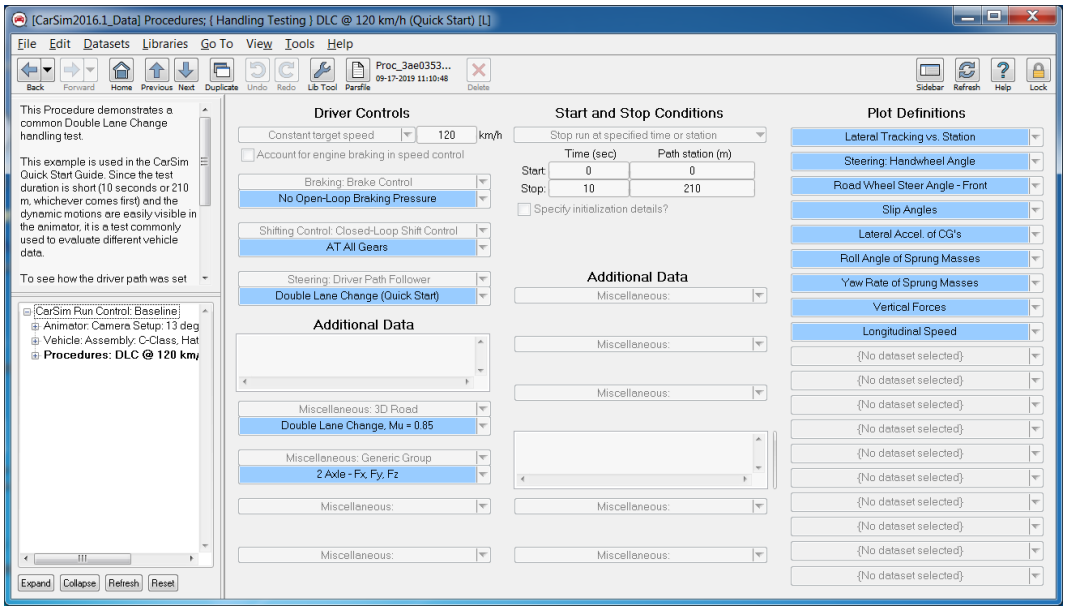

Figure 17: Procedure control screen.

4.1.2. Run control

This section is related to set the controls used for running CarSim math models. In particular, the simulator offers several environments in which implement the model (like Simulink, LabVIEW, and so on), as shown in Fig. 18.

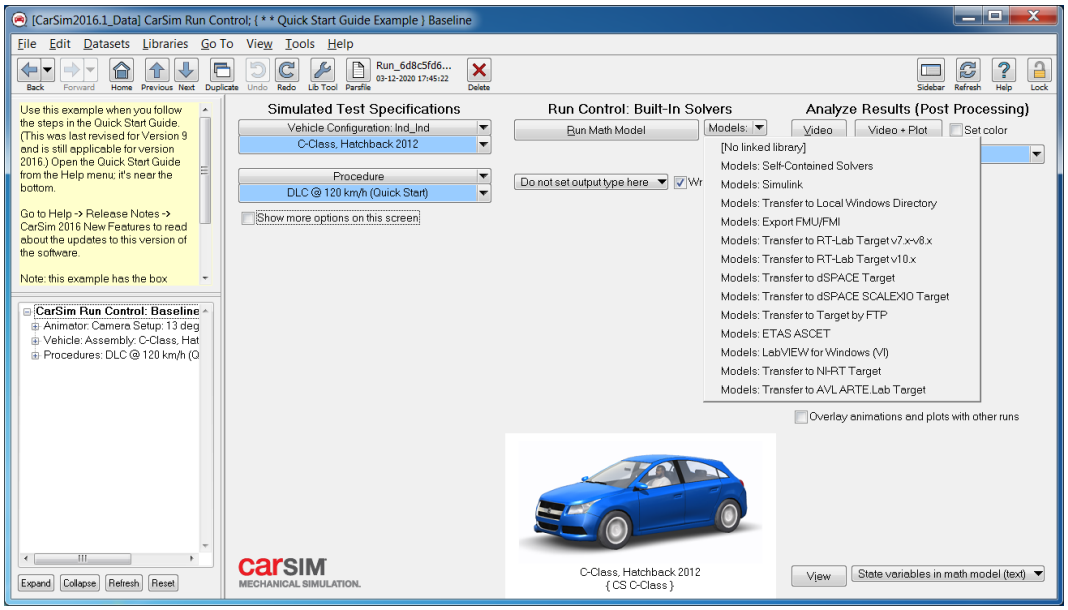

Figure 18: Solvers list screen.

To perform FMEDA analysis with the vehicle-level simulation proposed in paragraph 3.1, the physical models of the two inverters and motors are implemented in Simulink. Therefore, this is another excellent characteristic to use CarSim simulator.

4.1.3. Analyze Results (Post Processing)

The third column, of the *CarSim Run Control* screen, is dedicated to the visualization tool. When the simulation is finished, there are three ways to analyze the results:

1. *Plot*, setting the graphics to see in the *Plot Definitions* from *Procedure* menu, as said in paragraph 4.1.1;
2. *Video,* as a 3D reconstruction of the environment, the car and some arrows that represent some forces acting between ground and car;
3. *Video* + *Plot*, both of them.

The best way is the third one, because it allows, at the same time, to view the video of the simulation and the values of the graphs. In this way, it is easier to associate graph trends with the operating situation of the vehicle, as shown in Fig. 19.

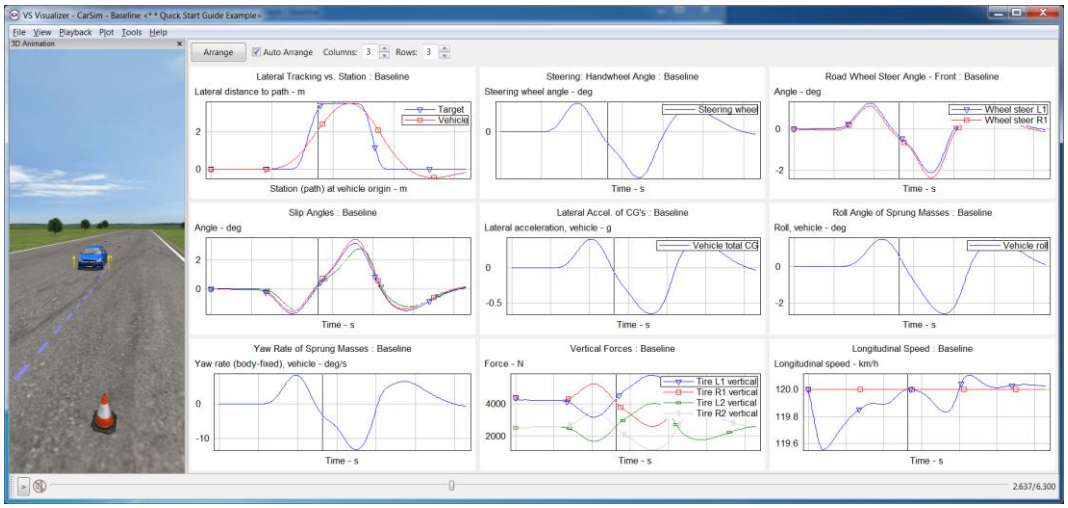

Figure 19: CarSim VS Visualizer screen.

4.1.4. Vehicle characteristics

As mentioned in paragraph 4.1.1, many predefined simulation sets (Datasets) are defined in CarSim. For each dataset, it is possible to change the type of vehicle, but for the majority of cases, they are all cars with a thermal engine. The vehicle examined for this thesis work is an electric vehicle. The powertrain of an electric vehicle is completely different from that of a thermal engine, as explained in paragraph 3.2. Therefore, an already existing dataset has to be adapted to our needs; in particular, a kinematic chain has chosen such that each mechanical component is supplied by an external source (Fig. 20).

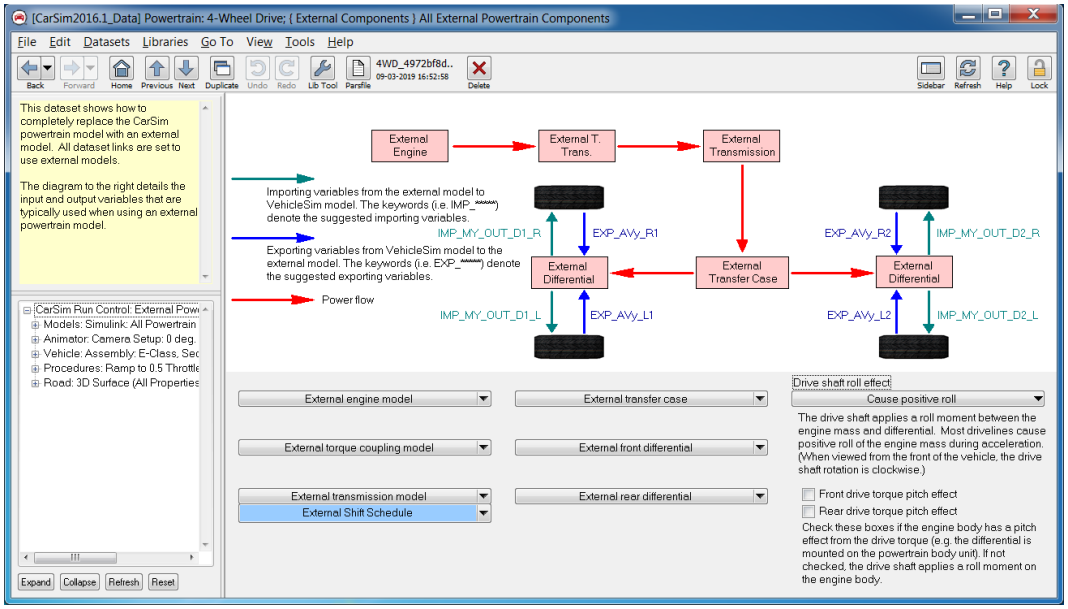

Figure 20: Powertrain Components screen.

As shown in figure 20, the only input is represented by the torque applied to the wheels. Vice versa, in terms of output, any signal can be selected.

In our specific case, the angular speed of the wheels has been taken into account, in order to provide them to the item model. Then, other signals used to implement the right control law will be listed.

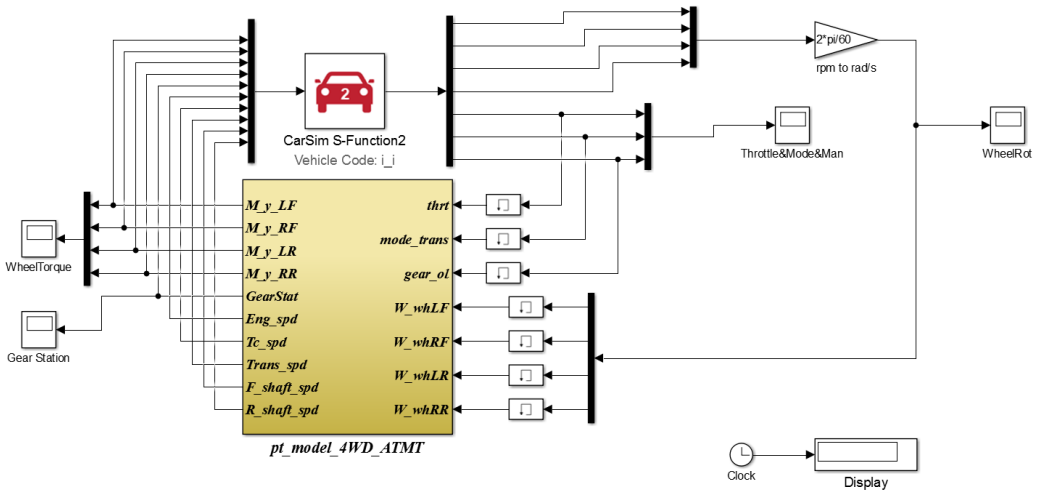

Figure 21: Simulink model of the original "All External Powertrain Components" dataset.

The Simulink model of the original dataset (Fig. 21) shows how CarSim works. All the characteristics related to the road, the vehicle, the simulation parameters and the solver used for the vehicle type that is simulated, are implemented in the *CarSim S-Function2* block.

4.2. Item model

Model is a simplified or partial representation of reality, defined to accomplish a task or to reach an agreement. Generally, we would like to have models that perfectly reflects the physical processes. But its complexity also depends on the limitations of the hardware systems used for the simulation. Therefore, a reasonable approach could be to set the level of complexity according to what is really required.

For this thesis, we decide to consider a trapezoidal brushless direct current (BLDC) motor type [26]. It is a kind of motor adopted in the most actual electrical vehicle powertrains. The alternative solutions (sinusoidal model) are more complicated from an electrical and control point of view influencing cost production [27]; even if, they present limited torque and speed ripple as the main benefits.

The block diagram of the item [28], as shown in Fig. 22, is composed of:

- microprocessor (μP), to generate PWM signals needed;
- power electronics stage required to drive the motor. It is composed of the six insulated-gate bipolar transistors (IGBTs);
- the motor itself;
- 3x hall-effect sensors to determine the rotor position.

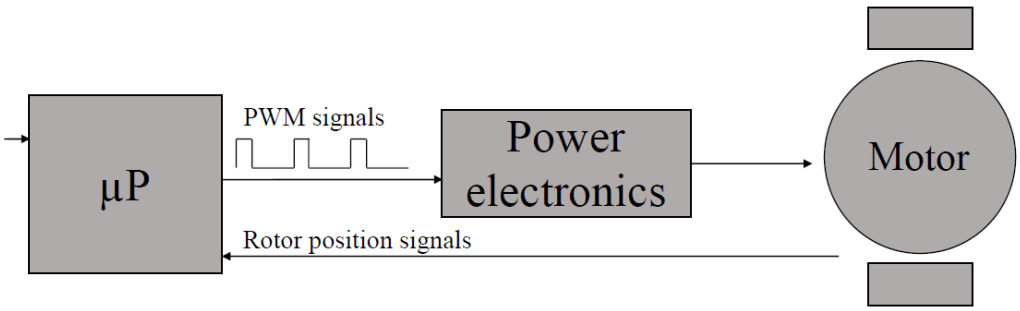

Figure 22: Block diagram of the item [28].

4.2.1. Embedded software model

The Simulink model of the microprocessor (Fig. 23) is mainly composed of a proportional-integral-derivative controller **(**PID) and a pulse-width modulation (PWM). The PID receives the speed error given by the difference between *SpeedRef* and *MeasuredSpeed*. Then, through an integral derivative proportional action, it calculates the command *D* to be passed to the PWM.

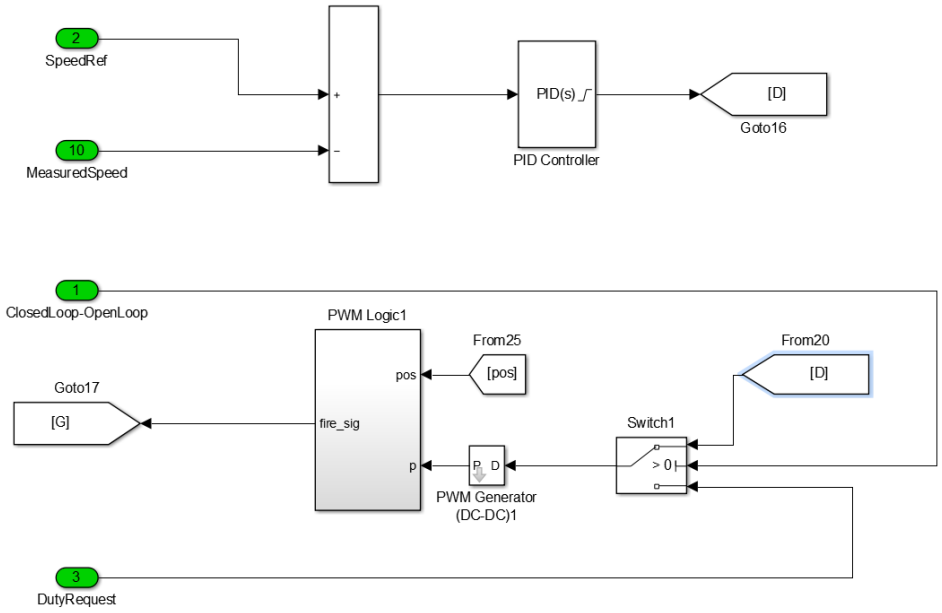

Figure 23: Embedded software Simulink model.

The compensator formula implemented by the PID is:

$$P + I \cdot \frac{1}{s} + D \cdot \frac{N}{1 + N \cdot \frac{1}{s}}$$

Where:

- *P* is the proportional gain;
- *I* is the integral gain;

- *D* is the derivative gain;
- *N* is the filter coefficient.

The *PWM Logic1* block receives the *D* command as an input and, based on the position of the rotor (*pos*), provides the command signal *G* to drive the inverter's MOSFETs. *Switch1* is used to set the PWM operating mode via the *ClosedLoop-OpenLoop* signal. When its logical value is low, the open-loop mode is chosen, the feedback loop opens, and the *G* command is calculated based on the *DutyRequest* coming from outside. Otherwise, the operative mode is described above, our case.

4.2.2. Motor model

In this case, as said before, we decide to consider a trapezoidal brushless direct current (BLDC) motor type. In addition to the block diagram, it is better to provide in details the schematics of the inverter (power electronics block) and the motor itself (Fig 24).

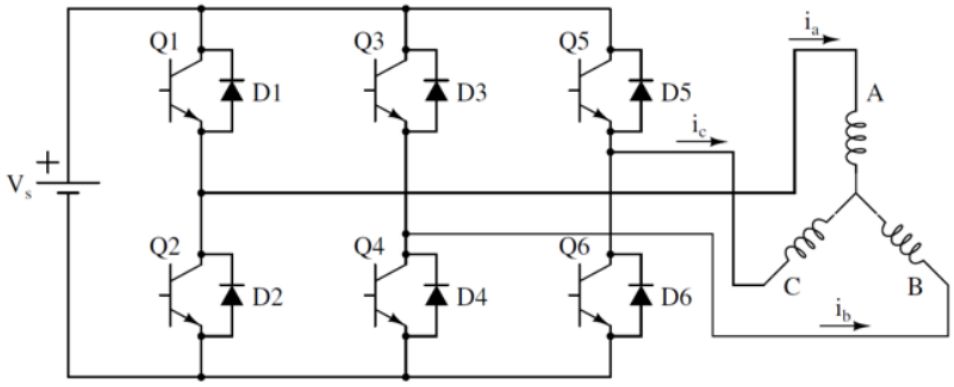

Figure 24: Power electronics and motor schematics [28].

An already motor on the market has chosen, and its parameters are listed in table 2.

Table 2: Motor data

Parameter	**Unit**	**Value**
Numbers of poles *n*	[pole pairs]	4
Nominal voltage *V*	V	500
Terminal resistance	Ω	12.5
Terminal inductance *L*	mH	0.091
Torque constant k_t	Nm/A (V·s)	1.5
Friction constant k_f	N · m s	$1.138 \cdot 10^{-8}$
Rotor inertia *J*	Kg cm^2	$5 \cdot 10^{-3}$

The not defined symbol reported in table 2 are:

- $i_{\{a,b,c\}}$: the current in phases a,b,c;
- $v_{\{ab,bc\}}$: the voltages between the phase a-b and b-c;
- $e_{\{ab,bc\}}$: the back electromotive force between the phases a-b and b-c;
- T_e: the electrical torque;
- T_L: the load torque;
- ω_m: the angular speed of the rotor;
- θ_m: the rotor angular position.

The space state equations needed to model the motor are:

$$\begin{cases} \dot{x} = A \cdot x + B \cdot u \\ y = C \cdot x + D \cdot u \end{cases} \rightarrow$$

$$\rightarrow \begin{cases} \begin{pmatrix} \dot{i_a} \\ \dot{i_b} \\ \dot{\omega_m} \\ \dot{\theta_m} \end{pmatrix} = \begin{pmatrix} -\frac{R}{L} & 0 & 0 & 0 \\ 0 & -\frac{R}{L} & 0 & 0 \\ 0 & 0 & -\frac{k_f}{J} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} i_a \\ i_b \\ \omega_m \\ \theta_m \end{pmatrix} + \begin{pmatrix} \frac{2}{3L} & \frac{1}{3L} & 0 \\ -\frac{1}{3L} & \frac{1}{3L} & 0 \\ 0 & 0 & \frac{1}{J} \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} v_{ab} - e_{ab} \\ v_{bc} - e_{bc} \\ T_e - T_L \end{pmatrix} \\ \begin{pmatrix} i_a \\ i_b \\ i_c \\ \omega_m \\ \theta_m \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} i_a \\ i_b \\ \omega_m \\ \theta_m \end{pmatrix} \end{cases}$$

By implementing this space state equations in Simulink, we obtain the figure 25:

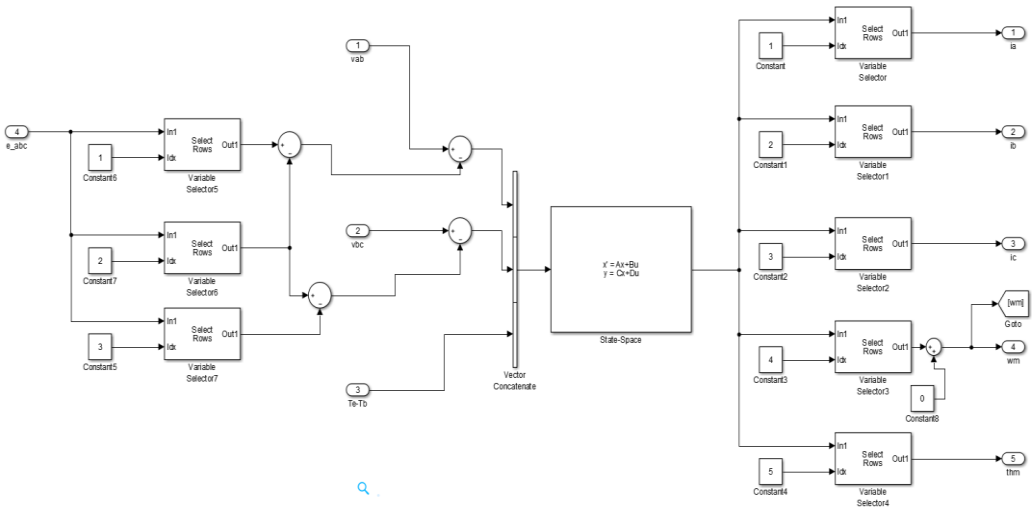

Figure 25: Motor state-space equations implemented in Simulink.

4.2.3. Power electronics model

Figure 24 also shows the inverter plot. It is composed of six insulated-gate bipolar transistors (IGBTs) powered by a direct voltage, while at the output they supply alternating electrical quantities.

In parallel with the design of the physical system, a proper error model has been chosen. A right approach is to reproduce the fault in a real scenario, trying to build an appropriate mathematical model. Therefore, apart from the effects, it is not always obvious to trace the causes that lead to a possible error condition. Furthermore, these conditions should be known from the beginning of the modelling, and this is not always true. Hence, a complete and exhaustive failure model is difficult to satisfy by covering all possible scenarios. However, for our specific application, it is not required; in fact, our goal is to reproduce the fault without investigating the boundary conditions that cause the break (such as overcurrent condition, failure of the junction). These aspects are well treated in other works [29]. We are much more interested in propagating these failures from the item-level to the vehicle-level and evaluating the effects. Therefore, the complexity of the fault model is significantly reduced and can be easily implemented by introducing a simple manual switching inside the inverter block. Which is why to perform fault injection, we used the MOSFET block provided by SimScape Toolbox, that is the built-in SPICE-level simulator of Mathworks Simulink^{TM}. In this way, the fault can be injected at any time during the simulation.

As described in [20], these switches are configured by the saboteur, reproducing a fault case where the single IGBT can no longer be driven, remaining permanently in an open or closed (short-circuit) configuration. Since in case of a short circuit, the fuses will melt down disconnecting the phase from the battery, as described in paragraph 3.4, so we inject only an open circuit failure.

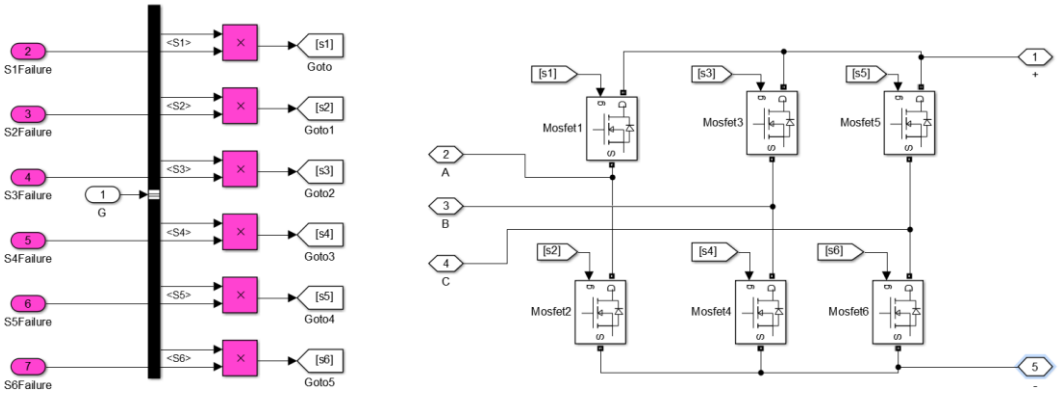

Figure 26: The complete model of the power electronics block implemented in Simulink.

Figure 26 shows that there are five "PMC_port" and seven "inport". The former are dedicated ports for the power connection of the subsystems, while the latter always provide a connection to a subsystem or model but with low power as signals.

The PMC_ports are:

- 1 and 5, respectively "+" and "-", are dedicated for the DC power supply voltage coming from the battery pack. In our case, we have chosen to omit them and to supply the inverter with a continuous and stable voltage;
- 2, 3, 4, respectively *A, B, C*, are the outputs from which to draw a three-phase voltage to drive the motor.

Instead, as regards the “inport”:

- 1, named *G*, is the output signal to the PWM, in charge of modulating the use of the MOSFETs based on the position error of the rotor at the input of the microprocessor;
- 6, in magenta, are the signals managed by the saboteur to inject the faults inside the inverter.

The *G* signal leaving the PWM passes through a *BusSelector*. This block accepts a bus as input which can be created from a Bus Creator, Bus Selector or a block that defines its output using a bus object. Six product blocks are present at the output, in order to implement the control logic for the individual MOSFETs even in the presence of the fault. Then the six signals are connected to the six gates.

4.3. Simulation results

The key point of the proposed approach concerns how to improve the assessment of the effects of failures on vehicle driveability. Because, as mentioned in paragraph 3.1, we inject the faults into the inverter and evaluate how failure modes propagate at the vehicle-level.

In the analysed system, the disparity torque on the rear axle can cause an unexpected turn of the vehicle and, in the worst case, even a rollover. For this reason, the built-in software must be able to modulate the torque of the two motors when cornering and limit the torque of the fault motor in case of failure.

The level of risk associated with the "vehicle function" (ASIL level) is determined, taking into account the ability of an average driver (defined as *controllability* by the ISO 26262) to mitigate the "failure effect". For this reason, we have represented the average driver as a PID controller, taking into account the reaction time of a human, with a target behaviour represented by a predetermined trajectory that has to be followed by the vehicle. This has been possible, thanks to the five driver sensor points provided by CarSim (Fig. 27).

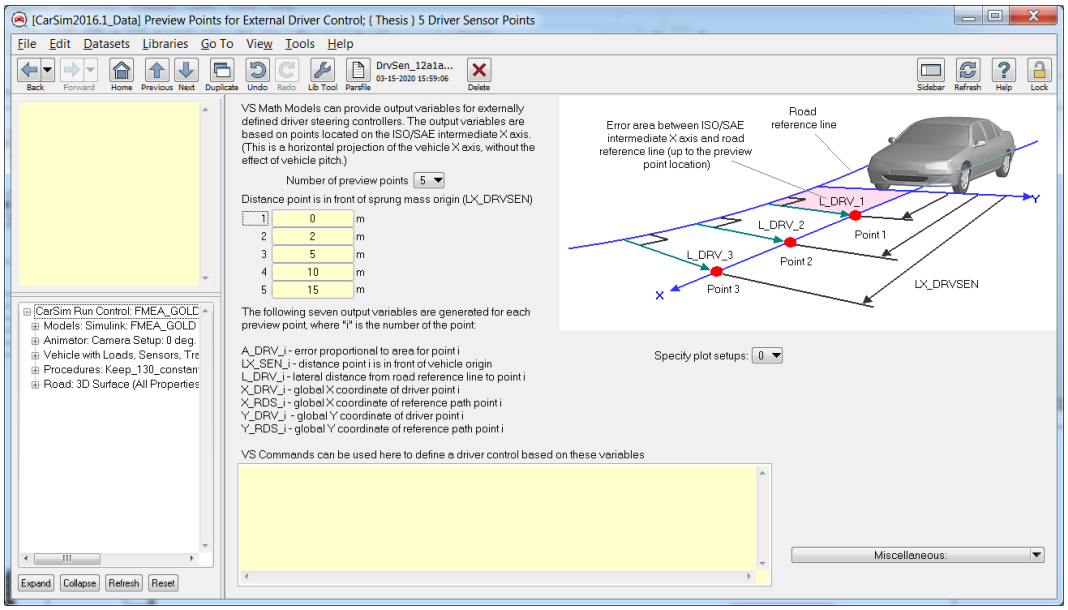

Figure 27: Preview points for external driver control screen.

As shown in fig. 27, the centre of the roadway is the *Road reference line* while the x-axis represents the trajectory that the car is travelling. We have set five preview points on which to calculate the steering angle. For each point, the error area (*A_DRV_i*), between the intermediate x-axis and road reference line up to the preview point location, and the lateral position error (*L_DRV_i*), lateral distance from road reference line are provided to point *i*. Point *1*, the one at 0-meter distance

in front of sprung mass origin, will be used to compare the results of the simulations, as it provides the lateral position error of the car.

For the scenarios to be simulated, the most significant operating modes of a vehicle have chosen. They are:

1. acceleration from 0 km/h to 130 km/h;
2. driving straight at 130 km/h;
3. triple curving at 100 km/h;
4. regenerative braking on a straight road from 130 km/h to 0 km/h;
5. regenerative braking on triple curving from 100 km/h to 0 km/h.

For each scenario, three simulations have been performed:

- fault-free condition, the golden solution;
- fault-affect condition;
- fault-affect condition with mitigation algorithm enabled.

The criteria used to compare the simulations are the lateral position error and the yaw angle. The latter is useful to understand the dynamic behaviour of the car when the driver is able to maintain the predefined trajectory (zero lateral error), as an asymmetric torque on the rear axle favours a yawing moment.

4.3.1. Acceleration from 0 km/h to 130 km/h

The simplified block diagram of the entire system is shown in figure 28.

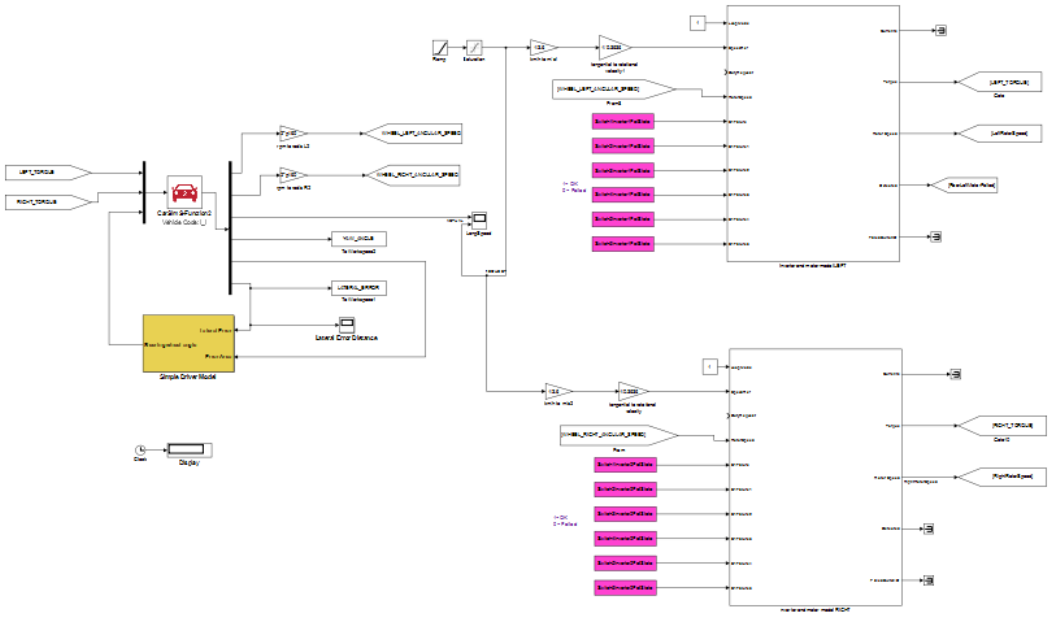

Figure 28: Complete system block diagram of the first scenario.

Now we will make some considerations that will apply to all simulations. To better understand the scheme (Fig. 28), we preferred to divide it into three subsystems: the interface concerning the *S-Function* and the two inverter and motor models considered individually.

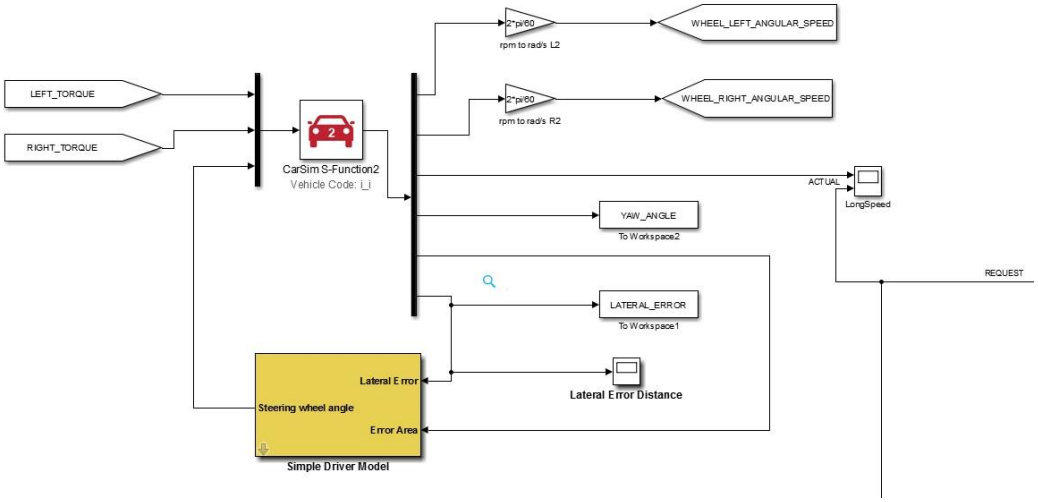

Figure 29: CarSim subsystem of the first scenario.

CarSim, as shown in figure 29, receives as input:

- *LEFT_TORQUE*, the torque applied to the left wheel;
- *RIGHT_TORQUE*, the torque applied to the right wheel;
- *Steering wheel angle*, coming from the *Simple Driver Model*. The latter, based on the *Lateral error* and *Error Area* of the preview points, previous paragraph, estimates the correction to be made for the steering angle to return in trajectory.

The desired output selected within the CarSim program are six:

- the angular speed of the rear wheels in "rpm", multiplied by the conversion factor $\frac{2\pi}{60}$, gives the angular velocities expressed in "rad/s" (*WHEEL_LEFT_ANGULAR_SPEED* and *WHEEL_RIGHT_ANGULAR_SPEED*);
- *LongSpeed*, the actual vehicle longitudinal speed. It is plotted with the one requested as a reference in order to display the time trend;

- *YAW_ANGLE* and *LATERAL_ERROR*, respectively, the yaw angle and the lateral position error, used to compare the simulations;
- *Error Area*, the air error used for the calculation of the steering angle, together with the lateral one.

Let us move on to the control and actuation systems of the left part of the rear axle (Fig. 30).

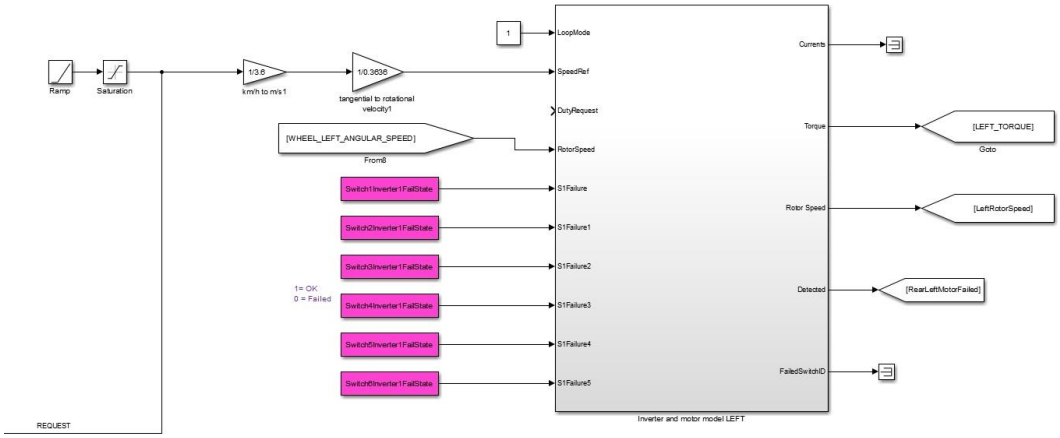

Figure 30: Left control and actuation subsystem of the first scenario.

Being an acceleration from 0 to 130 km/h, the speed reference to be provided to the controller is a ramp, while the saturation block is required to limit this reference to 130 km/h. It must be compared with the rotor speed, expressed as angular speed, which is why there are two gains at the output of the saturation block. The first conversion factor divides by 3.6 to transform km/h into m/s, while the second one represents the radius of the wheel to convert a linear speed into an angular one. Precisely, being a 225/60 R18 tire, the radius calculation is done in the following way:

$$R_{tire} = 0{,}225 \cdot 0{,}60 + \frac{18''}{2} = 0{,}225 \cdot 0{,}60 + 0{,}4572 = 0.3636m$$

The *LoopMode* will always be set at '1' due to a feedback control loop with a speed reference, the, as described in paragraph 4.2.1. The *WHEEL_LEFT_ANGULAR_SPEED* is the angular speed provided by CarSim, while the output *LEFT_TORQUE* will be the torque supplied to the left wheel. In magenta are the constant blocks used by the saboteur to inject the faults inside the inverter.

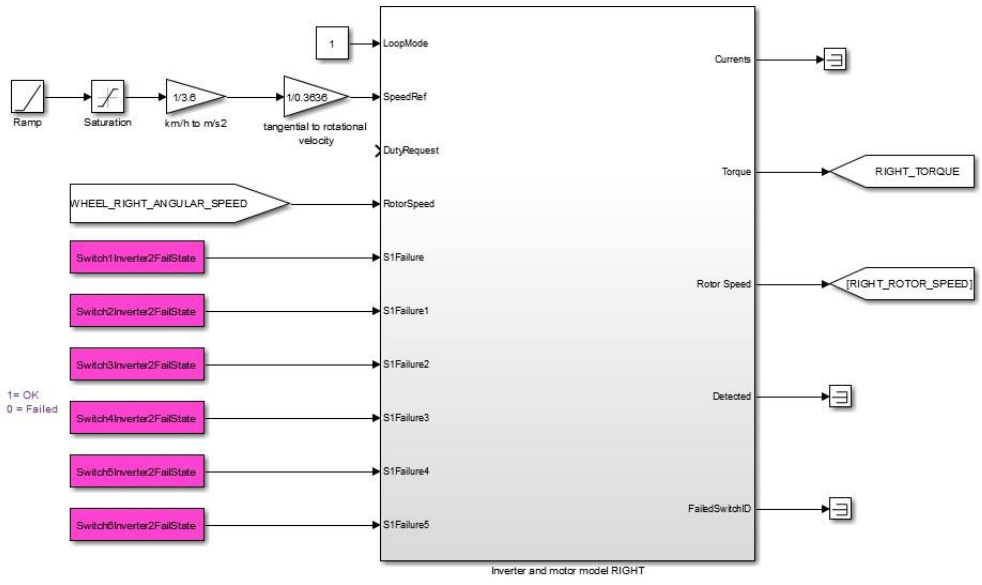

Figure 31:Right control and actuation subsystem of the first scenario.

The considerations made for the control and implementation system of the left part also apply to the right side (Fig 31) in fault-free and fault-affect conditions, while it is different in the fault-affect with mitigation condition. Its system block diagram is reported in the following figure (Fig 32).

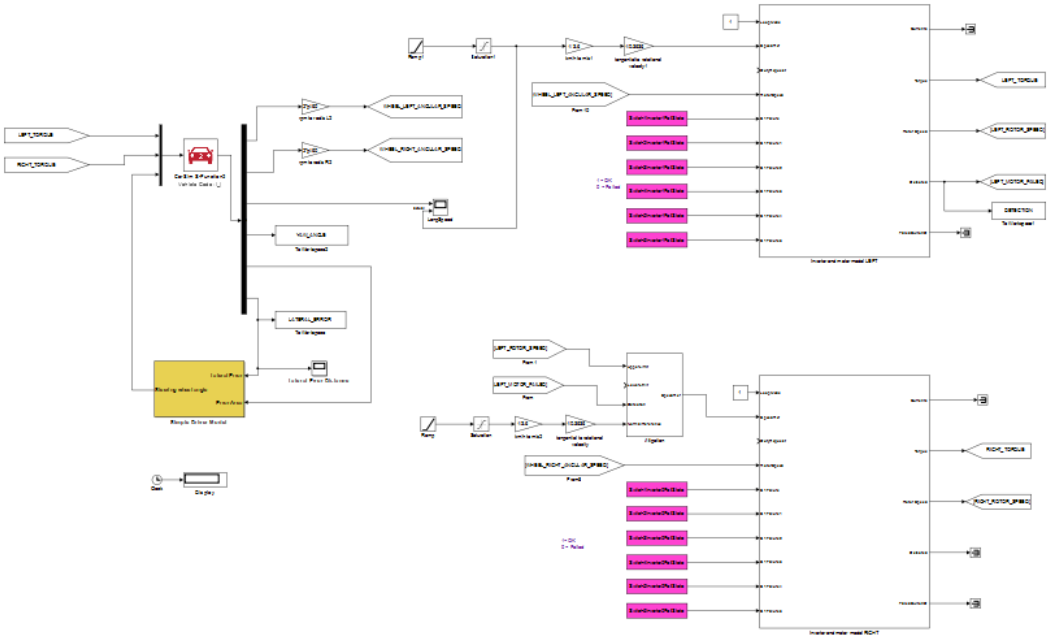

Figure 32: Complete system block diagram of the first scenario in "Fault + Mitigation" simulation.

To better understand the differences, we only consider the control subsystems, since the interface with CarSim does not present any modifications.

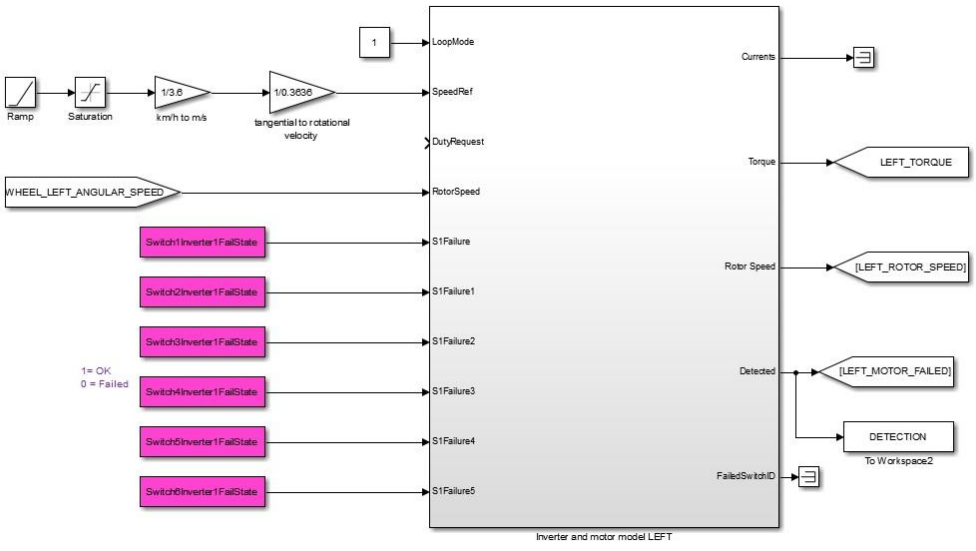

Figure 33: Left control and actuation subsystem of the first scenario in "Fault+Mitigation" simulation.

In this simulation, the detection action by the item must be taken into account. In particular, the *LEFT_MOTOR_FAILED* (reference figure 33) signal is the one who drives the mitigation action. The *DETECTION* data, implemented as the "To Workspace" block, is used to save the detection time in the Matlab workspace.

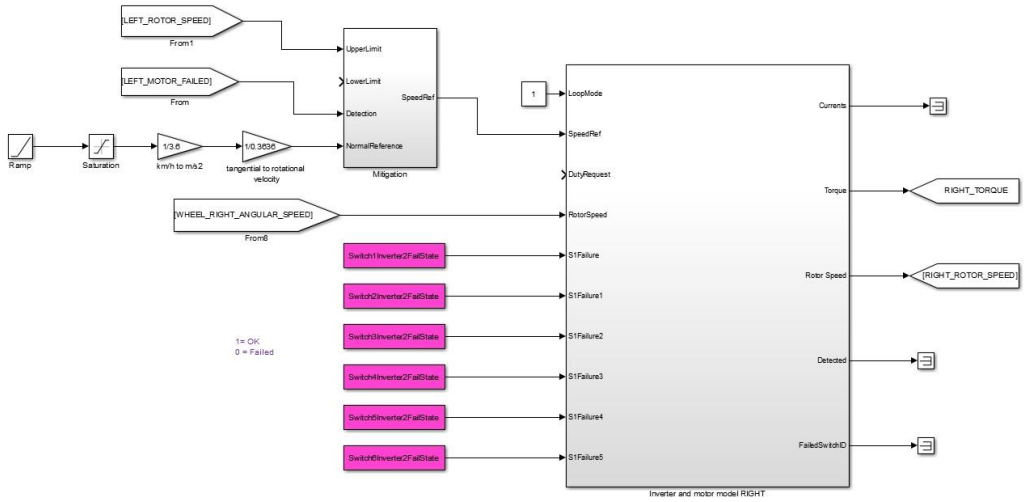

Figure 34: Right control and actuation subsystem of the first scenario in "Fault+Mitigation" simulation.

As described in paragraph 3.5, the mitigation strategy adopted is to limit the speed reference of the fault-free motor up to the rotation speed of the fault-affected one. As shown in figure 34, the *LEFT_ROTOR_SPEED* is connected to the "UpperLimit" of the "Mitigation" block, while the ramp reference is connected to the "NormalReference"; all driven by the *LEFT_MOTOR_FAILED* signal connected to the "Detection" port.

The first simulation must be done in fault-free condition (golden solution), which will be the reference for the other two conditions. The car took 16 s to reach the target speed (Fig. 35), but the performances are out of the scope of the current work.

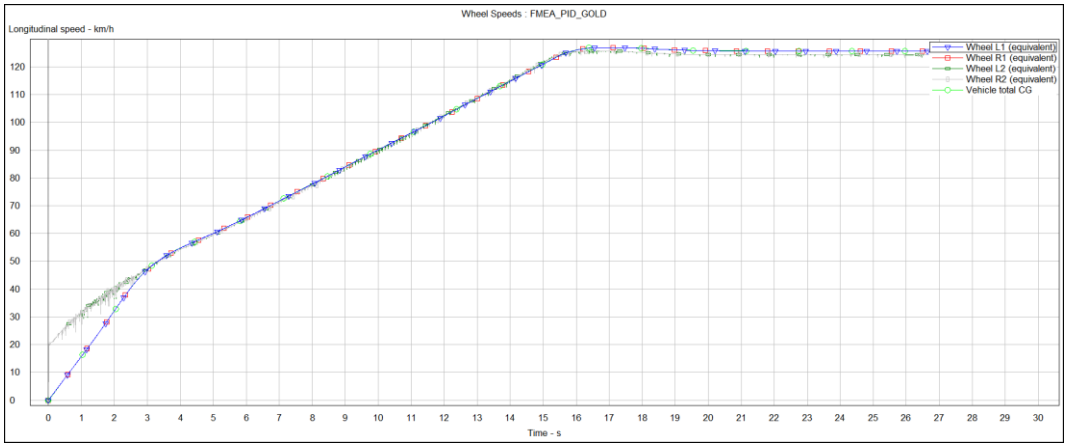

Figure 35: Longitudinal speed characteristic.

The graphs containing the results of the first simulation are shown in figure 36 and 37.

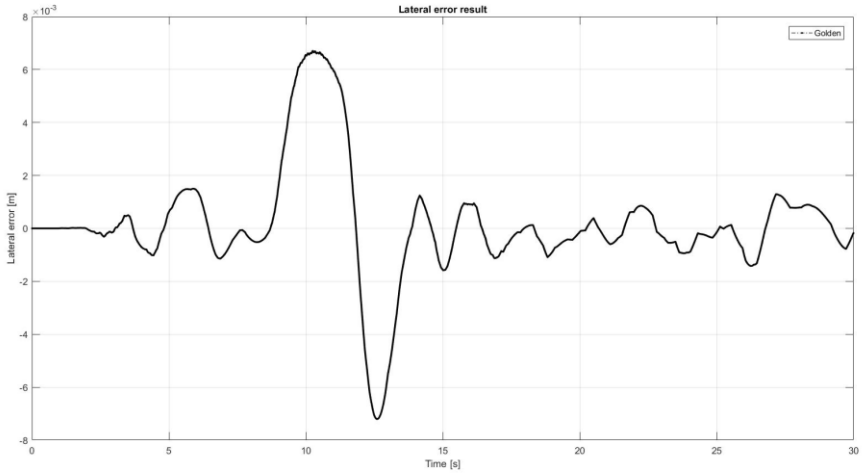

Figure 36: "Golden" lateral error in the first scenario.

The lateral error is in the order of millimetres, due to the small tolerance of the PID used to simulate the driver.

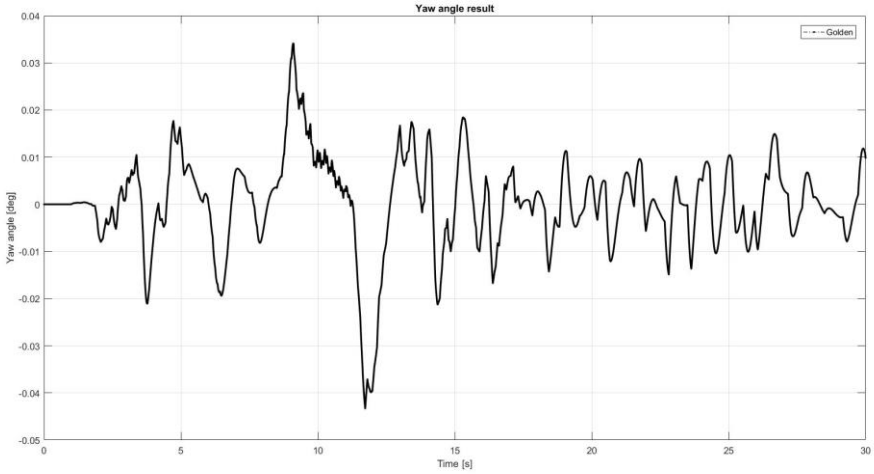

Figure 37: "Golden" yaw angle in the first scenario.

The graph is consistent with the lateral error one, as the small yaw angle values are due to slight corrections to the steering angle.

Once the "golden solution" is obtained, the saboteur injects the fault inside the inverter of the left wheel, leaving the system to evolve on its own with no mitigation action. The only factor capable of keeping the vehicle in trajectory is the driver's ability.

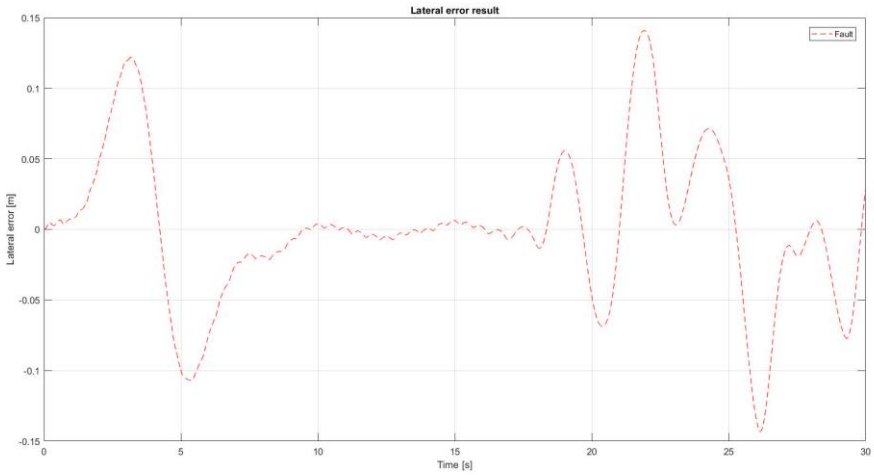

Figure 38: "Fault" lateral error in the first scenario.

Fig 38 shows the lateral error. As expected, the values are more significant than the Golden, reaching a peak of almost 15 cm. While in figure 39, a larger yaw angle values are present due to the torque disparity and the correction made by the PID to stay in the trajectory. Its peak at high-speed is 0.68 deg.

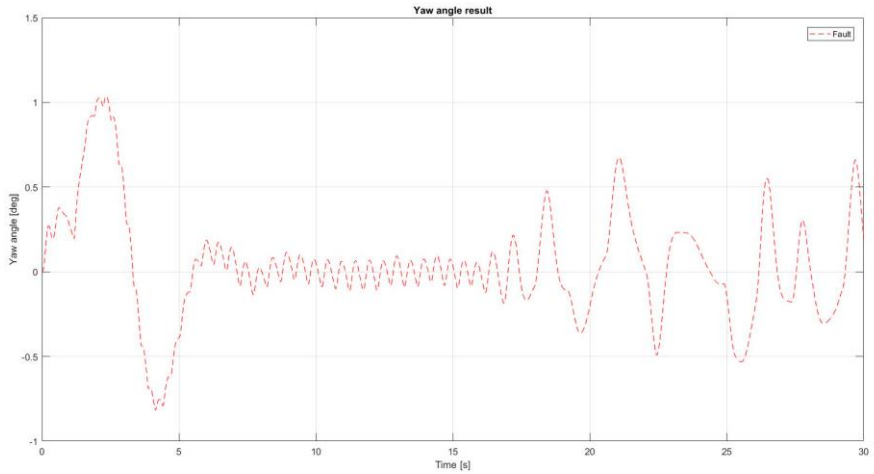

Figure 39: "Fault" yaw angle in the first scenario.

The last simulation made in this scenario is the fault condition with mitigation algorithm enabled. The results are shown below.

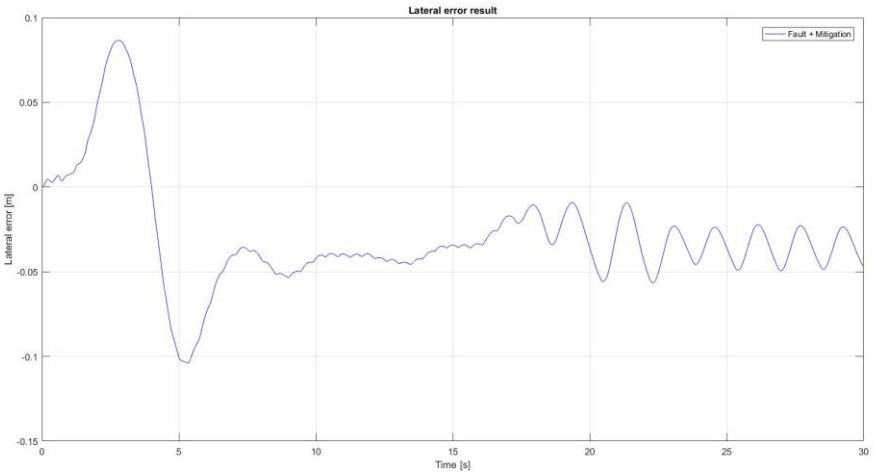

Figure 40: "Fault + Mitigation" lateral error in the first scenario.

From fig. 40, it is clear that the mitigation action limited the lateral error concerning the fault condition. Not considering the initial peak of 1 cm, the error at high-speed is limited in an interval of 5 cm. The same goes for the yaw angle (Fig. 41), which reaches a peak of 0.27 deg at high-speed and then it is limited to lower values.

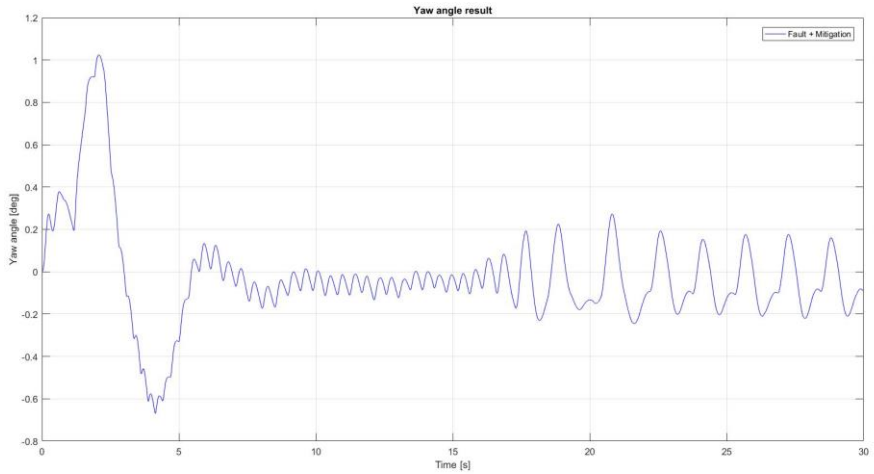

Figure 41: "Fault + Mitigation" yaw angle in the first scenario.

To better understand the results obtained from the simulations, we prefer to add another plot containing the three lateral errors.

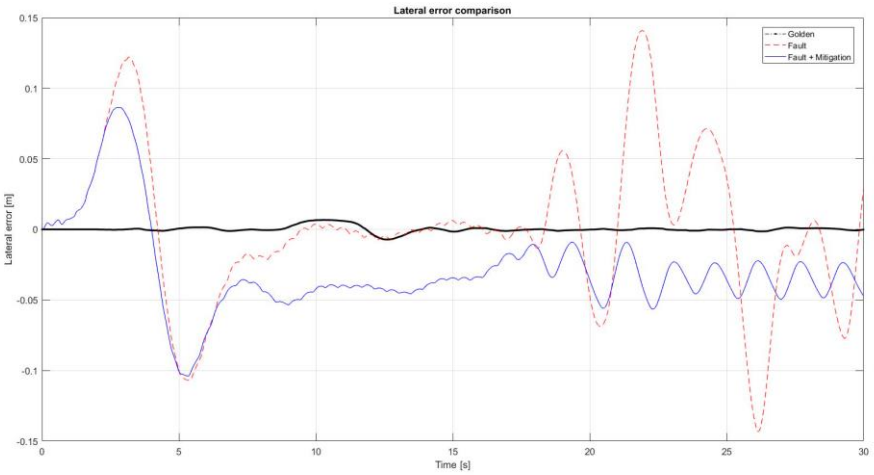

Figure 42: Lateral error results in the first scenario.

The graph shows (Fig. 42) that the mitigation algorithm is quite useful to limit the failure effects. After the fault is detected, at about 2 s, the benchmark mitigation algorithm can limit the lateral error, especially at high speed when it is more difficult for a human pilot to intervene.

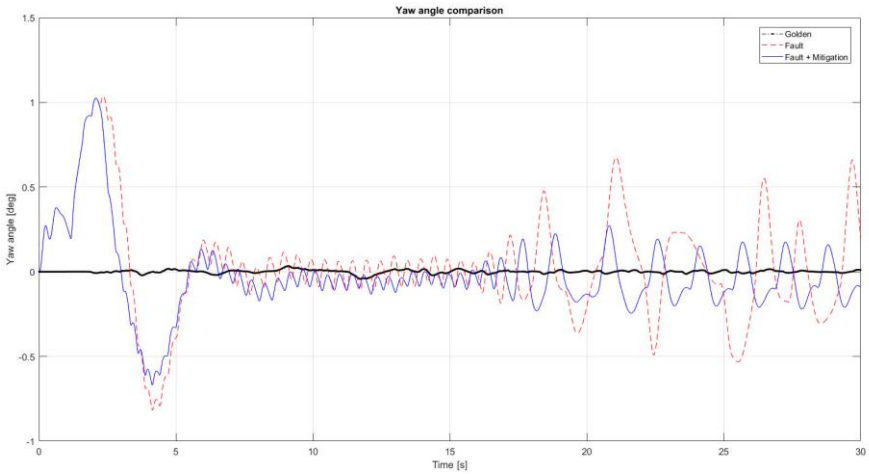

Figure 43: Yaw angle results in the first scenario.

Analyzing the results in terms of yaw angle (Fig. 43), it is clear that the mitigation algorithm is able to reduce and limit the error from a range of -0.5 to 0.6 deg to a range of -0.2 to 0.2 deg.

In conclusion, we can say that the mitigation algorithm, in this case, is capable of limiting both the lateral error and the yaw angle, even if the algorithm is straightforward.

4.3.2. Driving straight at 130 km/h

The second scenario considered is to maintain a cruising speed of 130 km/h on a straight road. The block diagram of the entire system is shown in the figure below.

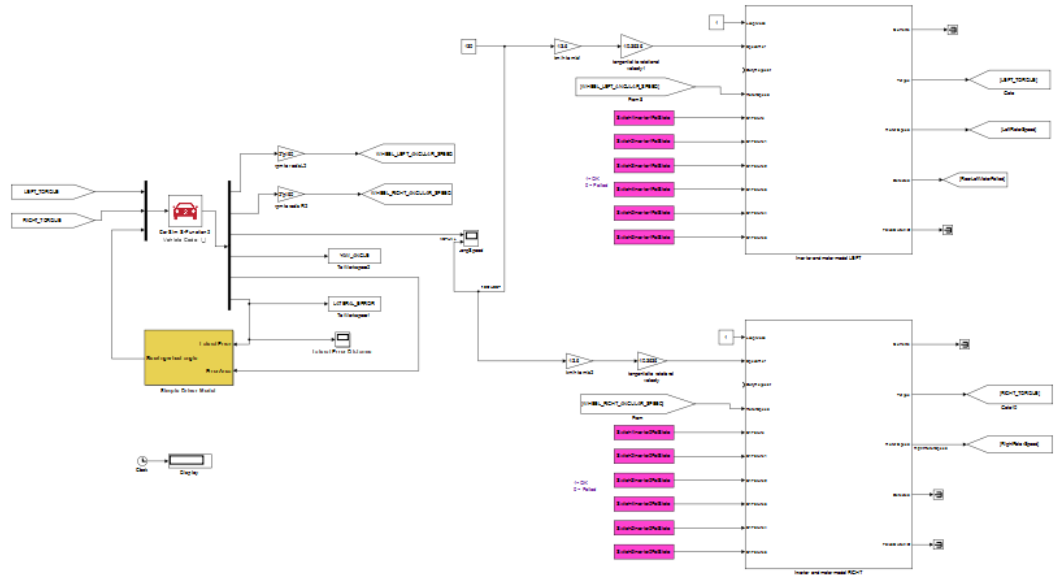

Figure 44: Complete system block diagram of the second scenario.

The interface with CarSim has no changes compared to the previous scenario, while it is different for the control and actuation models.

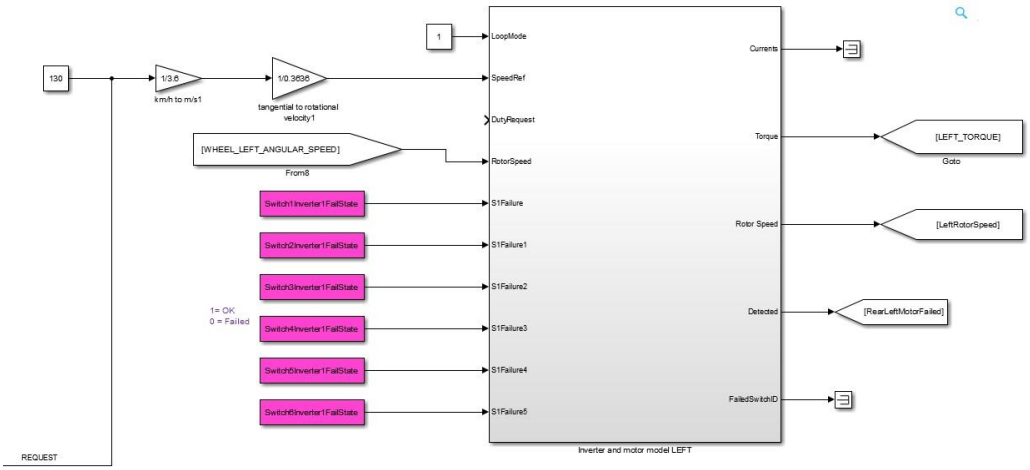

Figure 45: Left control and actuation subsystem of the second scenario.

Being a simulation of the vehicle with cruising speed at 130 km/h, the *SpeedRef* provided to the controller is a constant. For this reason, the value block "130" is present in fig. 45 and 46.

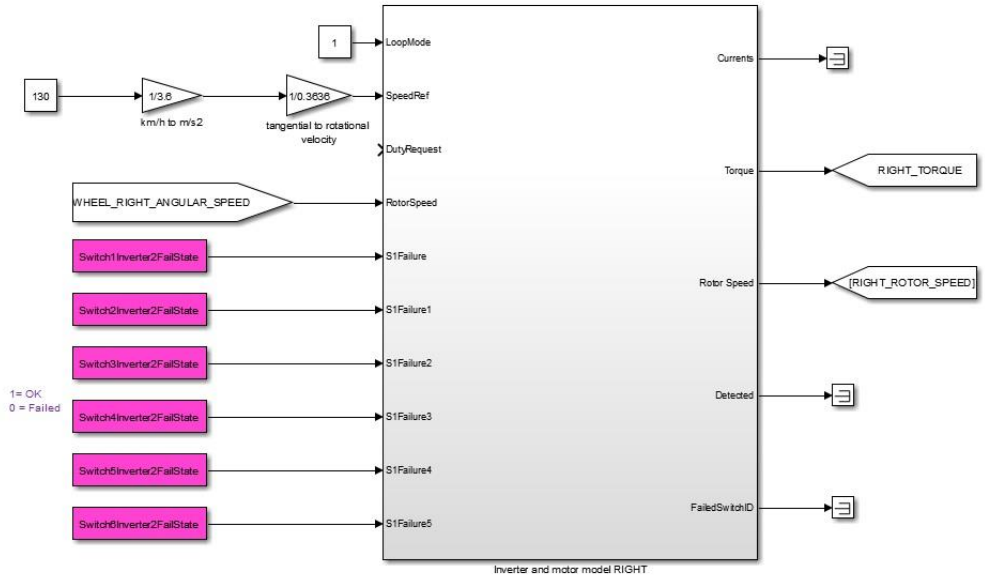

Figure 46: Right control and actuation subsystem of the second scenario.

The considerations, made on the speed reference, are the same for the simulation of the fault-affect with mitigation condition. The subdivision into subsystems is omitted, and only the complete block diagram of the system is shown in Fig. 47.

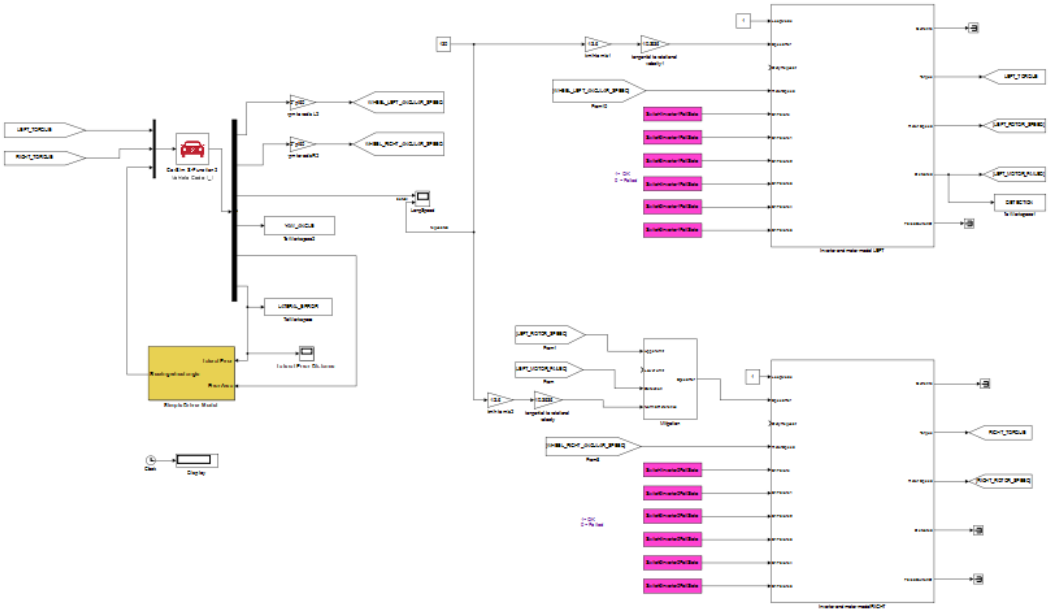

Figure 47: Complete system block diagram of the second scenario in "Fault+Mitigation" simulation.

Also, in this scenario, the first simulation is in fault-free condition, which will provide the results to compare the “fault” and “fault + mitigation” simulations.

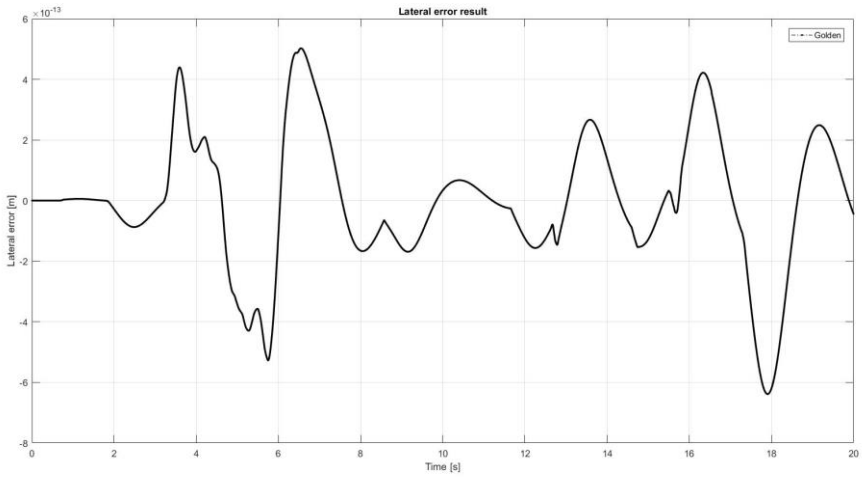

Figure 48: "Golden" lateral error in the second scenario.

As shown in the fig. 48, the error is in the order of 10^{-13}, so it is only noise comes from the sensors or approximation errors of the solver.

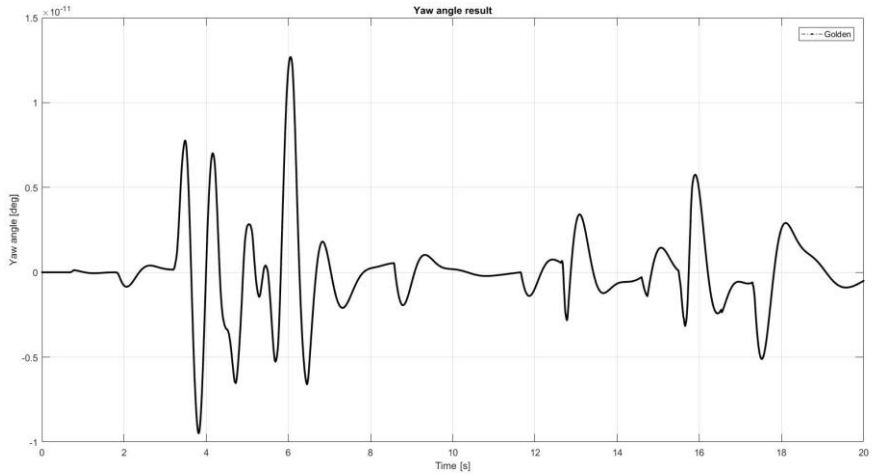

Figure 49: "Golden" yaw angle in the second scenario.

Also, for figure 49, the yaw angle is of the order of 10^{-11}; any discussion on the result is superfluous. In fault-free condition, the vehicle is able to maintain a straight path.

The second simulation is performed after the injection of the fault by the saboteur into switch *1* of the left inverter. The results are shown below.

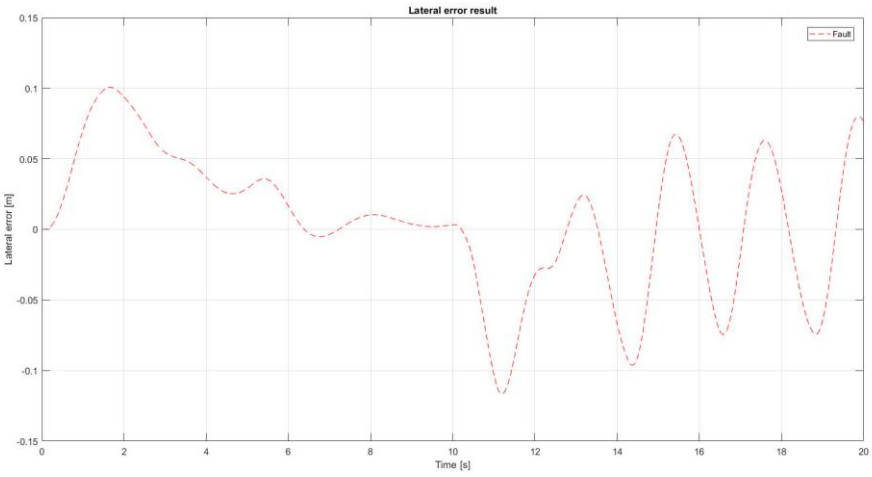

Figure 50: "Fault" lateral error in the second scenario.

Excluding the two initial peaks (Fig. 50), also present in the mitigation condition, the lateral error shows an oscillatory trend, due to the corrections on the steering angle by the PID. The next peak is almost 10 centimetres.

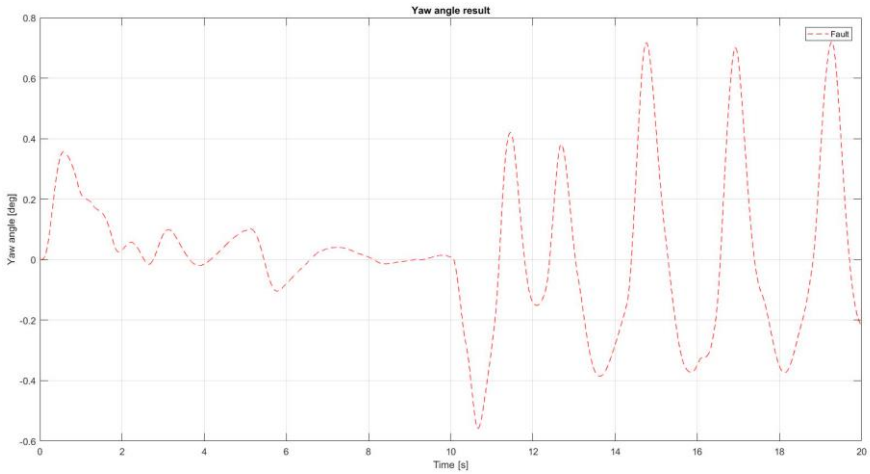

Figure 51: "Fault" yaw angle in the second scenario.

Analyzing the error in terms of yaw angle (fig. 51), we can see that the peaks correspond to those of the lateral error. The unbalanced torque on the rear axle leads to instability. Its maximum peak is 0.72 degrees.

In the third simulation, the mitigation algorithm takes about 12 seconds for detection the fault; therefore, the data will be equal to the fault-affect up to the detection time.

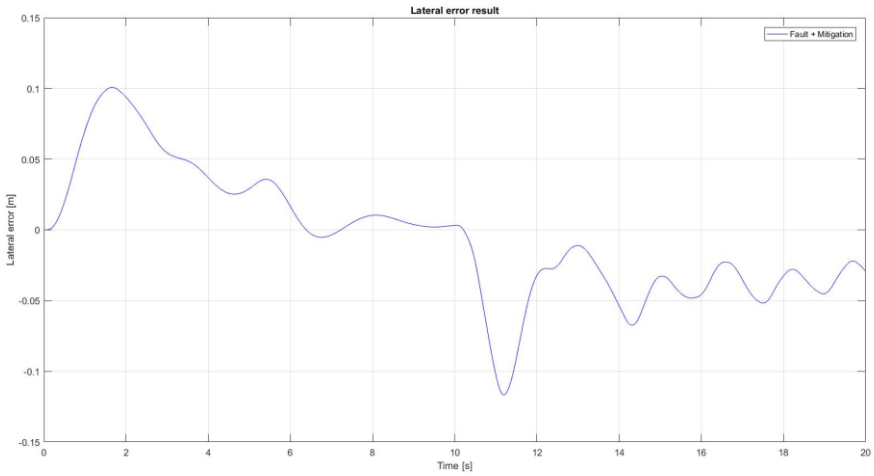

Figure 52: "Fault + Mitigation" lateral error in the second scenario.

As shown in figure 52, we can see that the mitigation action consistently limits the lateral error. It passes to a peak of almost 12 cm, in absolute value, to a peak of 6.7 cm, thus halving its value.

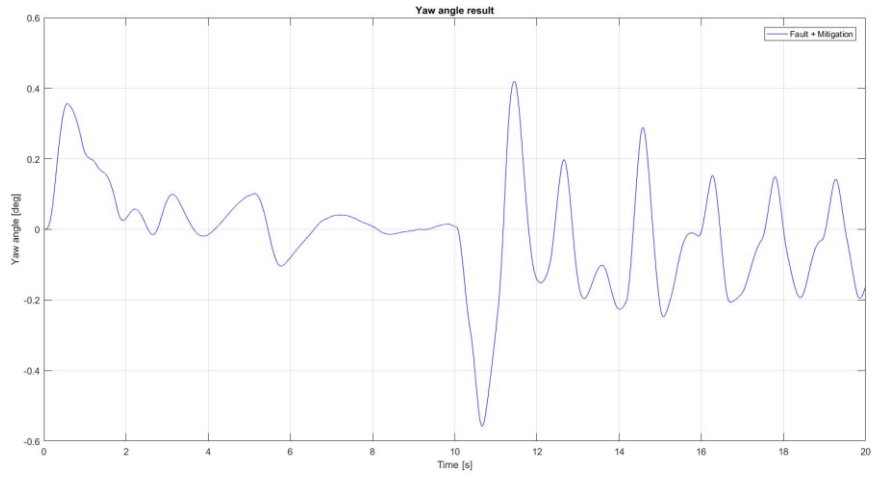

Figure 53: "Fault + Mitigation" yaw angle in the second scenario.

Figure 53 shows the simulation results in terms of yaw angle. The trend is in line with what we expected, there is a peak of 0.29 degrees after the detection time, after which its value decreases more and more.

In the following, two figures are containing the graphs of the three simulation results.

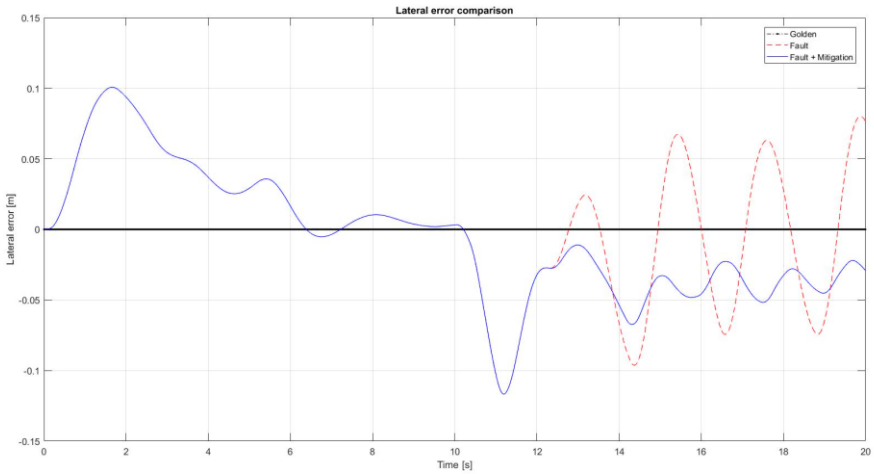

Figure 54: Lateral error results in the second scenario.

As shown in figure 54, the mitigation algorithm is able to halve the lateral error. Furthermore, the oscillations are limited even if the vehicle is not perfectly aligned with the centre of the trajectory, although it falls within the roadway margins.

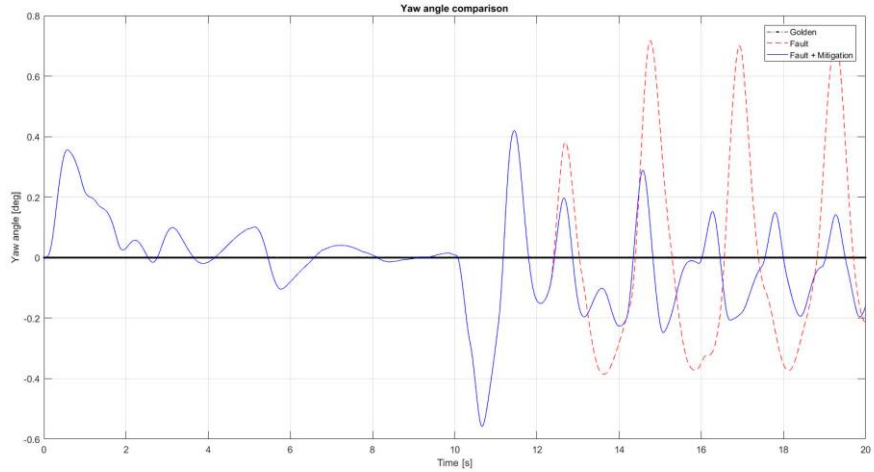

Figure 55: Yaw angle results in the second scenario.

Analyzing the error in terms of yaw angle (Fig. 55), we can see that the mitigation algorithm is able to reduce the maximum value from 0.72 to 0.29 degrees. In

addition, the amplitude of the oscillations is reduced from the range [-0.38 , 0.72] deg to [-0.24 , 0.29] deg.

In conclusion, also for the second scenario, the mitigation algorithm has proven useful in reducing errors, even if the detection time is 12 seconds. This may be connected to the fact that the initial speed of the vehicle is already 130 km/h; therefore, the fault-affect engine is slightly dragged by the inertia of the vehicle. Unlike the first scenario where, being an acceleration from 0 to 100 km/h, the maximum torque delivered by the engine is at low speed.

4.3.3. Triple curving at 100 km/h

In this simulation, the default path is composed of three curves, one on the right and two on the left side (Fig. sotto), covered at 100km/h. Since the road is not a straight line, the reference rotor speed of the two motors is different because the vehicle is curving, so we have to consider the rotational component through the instantaneous centre of zero velocity.

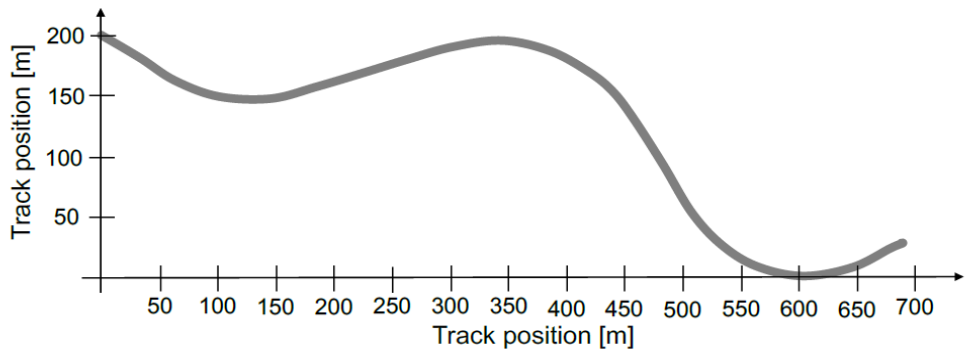

Figure 56: The triple curving track implemented in the simulation environment [22].

"CarSim" provides the inverse of a curvature radius $\left(\frac{1}{R_{CURVE}}\right)$ to overcome the problem relates to an infinite radius in a straight line. Let make some consideration on the sign: the simulator defines a positive curvature radius for curving on the left and the negative one for curving on the right. Since the longitudinal speed of the car (V_x), has already been extrapolated, the angular speed of the vehicle is calculated by applying a mechanical law:

$$\omega_{CAR} = \frac{V_x}{R_{CURVE}}$$

Once known the ω_{CAR}, the geometry of the car and the sign of R_{CURVE}, we can impose the tangential velocity of the wheels to perform any curve. In particular:

$$\begin{cases} V_L = V_x - \omega_{CAR} \cdot \dfrac{width}{2} \\ V_R = V_x + \omega_{CAR} \cdot \dfrac{width}{2} \end{cases}$$

Where V_L and V_R are respectively, the left and right tangential velocity. So, the desired reference rotor speeds of the motors are given dividing the tangential velocities by the tire radius:

$$\begin{cases} \omega_L = \dfrac{V_L}{R_{TIRE}} \\ \omega_R = \dfrac{V_R}{R_{TIRE}} \end{cases}$$

The block diagram of the entire system used for this scenario is shown in figure 57.

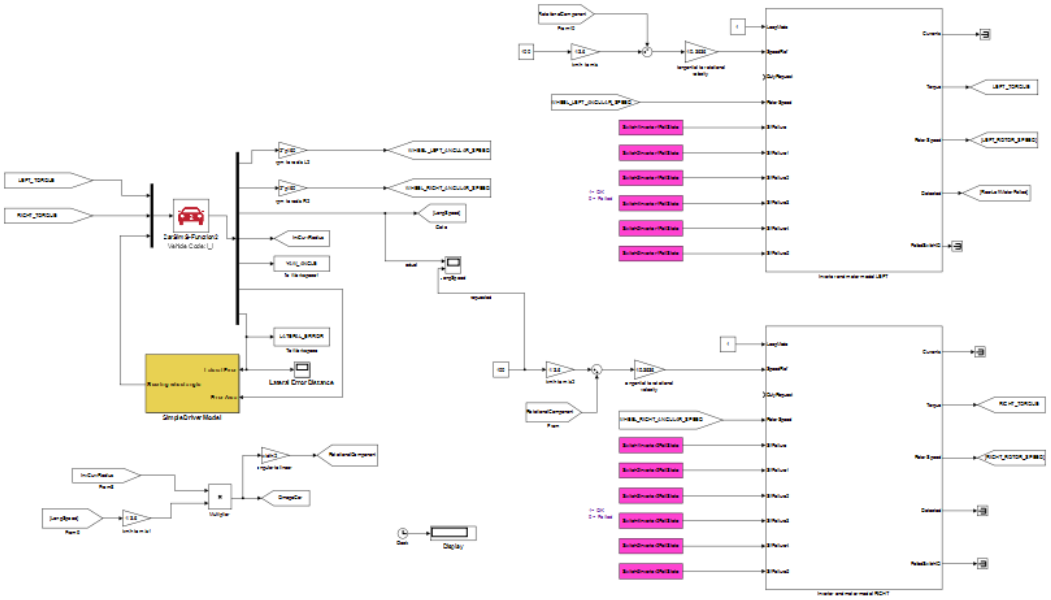

Figure 57: Complete system block diagram of the third scenario.

To better understand the modifications, it has been preferred to subdivide the entire system into subsystems. In particular, the CarSim interface (Fig 58) there are *InvCurvRadius* and *LongSpeed* that are, respectively, the inverse of a curvature radius $\left(\frac{1}{R_{CURVE}}\right)$ and longitudinal speed of the car (V_x).

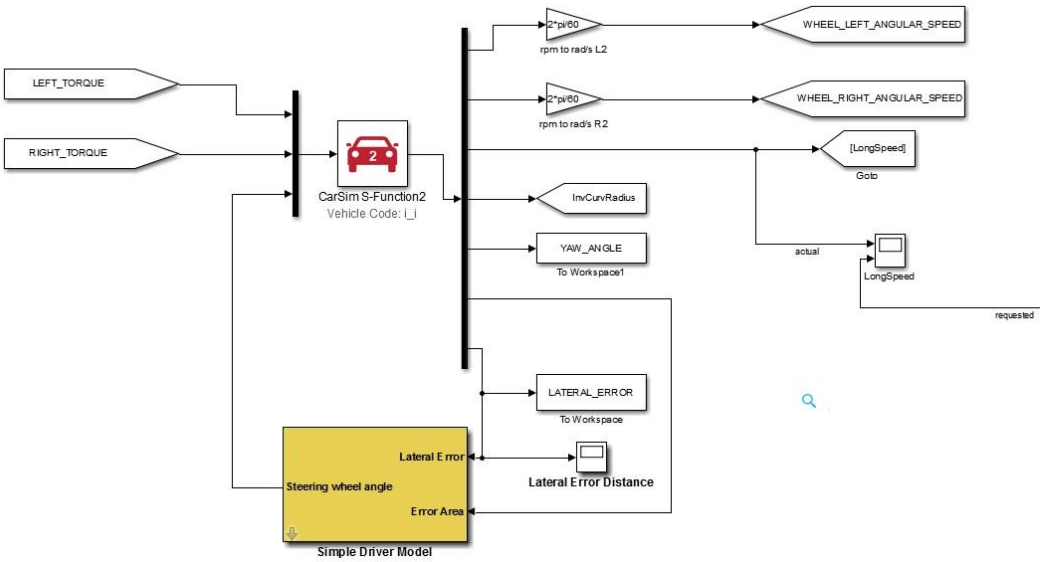

Figure 58: CarSim subsystem of the third scenario.

The implementation of the rotational component in Simulink is shown in figure 59.

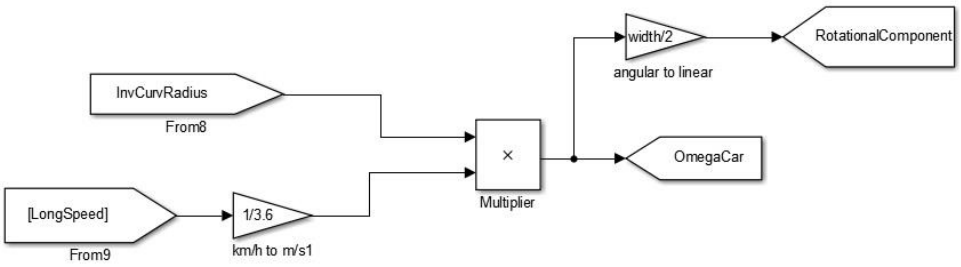

Figure 59: Rotational component implementation in Simulink.

RotationalComponent indicates the linear speed of the rotational component to be added to the outer wheel and subtracted from the internal one to make the turn.

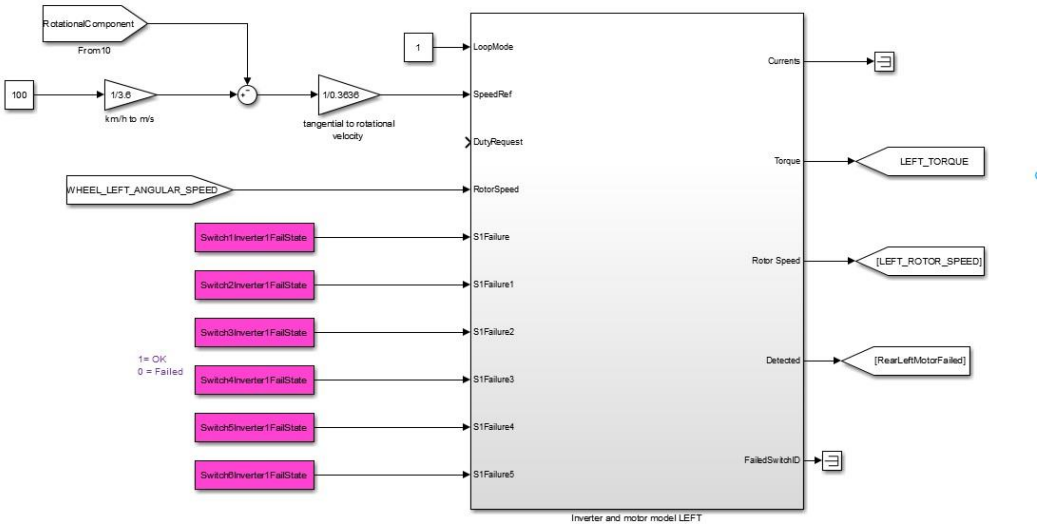

Figure 60: Left control and actuation subsystem of the third scenario.

Since the sign of the inverse of the curvature radius is negative for the left turns, the *RotationalComponent* will always be added with the "-" sign inside the summing node (Fig. 60). Consequently, for the right turns, it will be with the "+" sign (Fig 61).

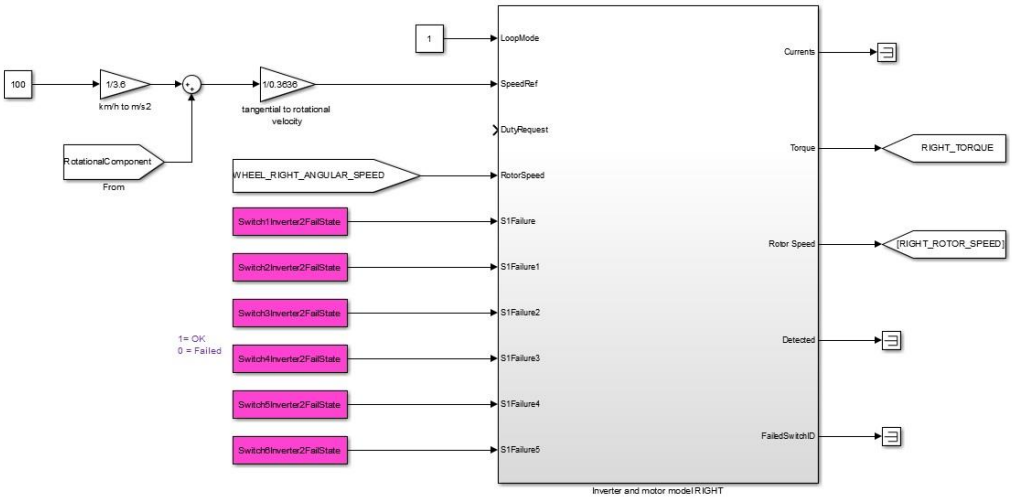

Figure 61: Right control and actuation subsystem of the third scenario.

"Golden" and "Fault" simulations are made following the previous speed profiles. Instead for the "Fault with Mitigation" condition, the speed profile of the fault-free motor must be adjusted to rotor speed profile of the fault-affect one.

Supposed that the motor with a fault is the left one and considering the instantaneous centre of zero velocity, we can say:

$$V_R = V_L + \omega_{CAR} \cdot width \Rightarrow \omega_R = \frac{V_R}{R_{TIRE}}$$

Obtaining the final relation:

$$\omega_R = \omega_L + \omega_{CAR} \cdot \frac{width}{R_{TIRE}}$$

So, in this case, the reference speed of the right motor rotor must be bounded between the saturated value, calculated by the above equation, and the normal one computed to perform any curve without fault. Its implementation is shown in figure 62.

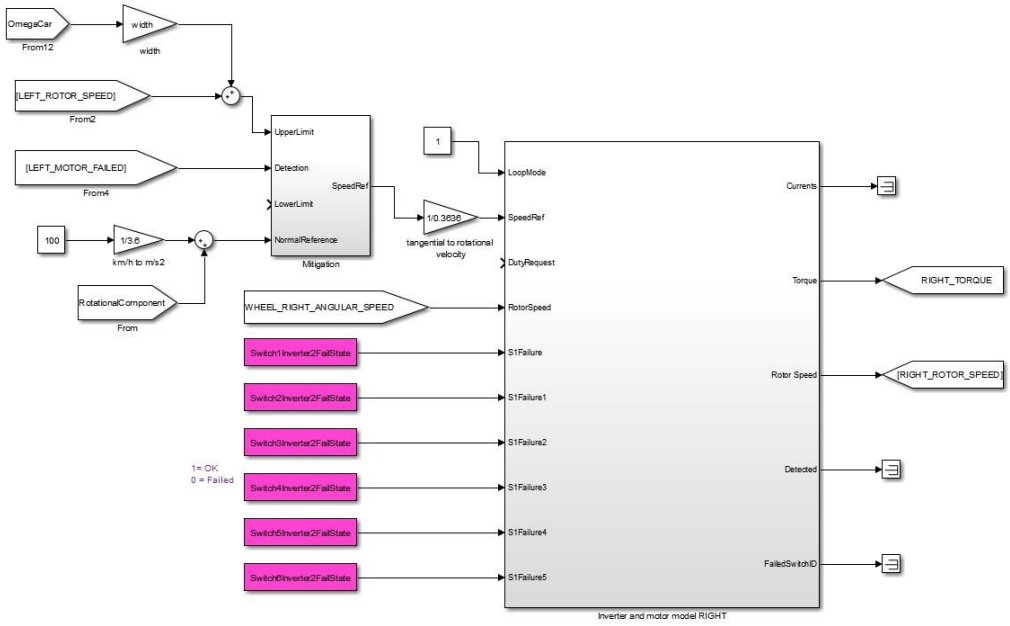

Figure 62: Right control and actuation subsystem of the third scenario in "Fault+Mitigation" simulation.

LEFT_ROTOR_SPEED represents "ω_L", *OmegaCar* is "ω_{CAR}", the gain block is the car width and the final gain, after the Mitigation block, is the "R_{TIRE}".

The block diagram of the entire system used in the simulation with the Fault + Mitigation conditions is shown in figure 63.

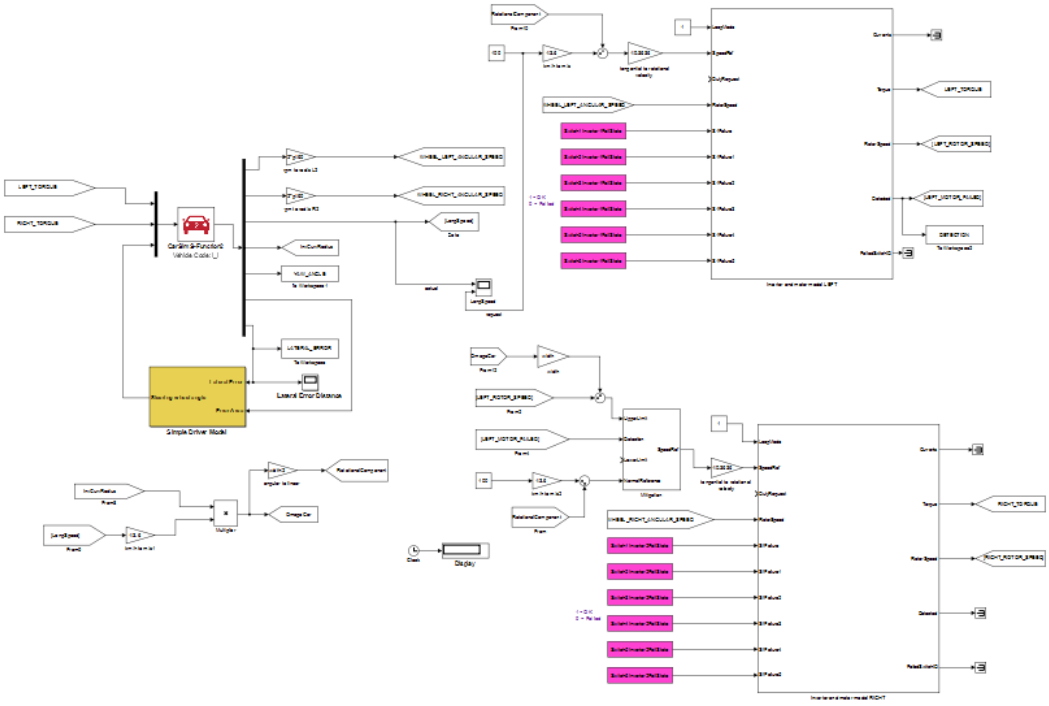

Figure 63: Complete system of the third scenario in "Fault + Mitigation" simulation.

Once the speed references to be provided to the controller have been perfected, simulations are performed. In particular, the first is always the fault-free condition, which will give us a reference for comparing the other simulations. The results are shown below.

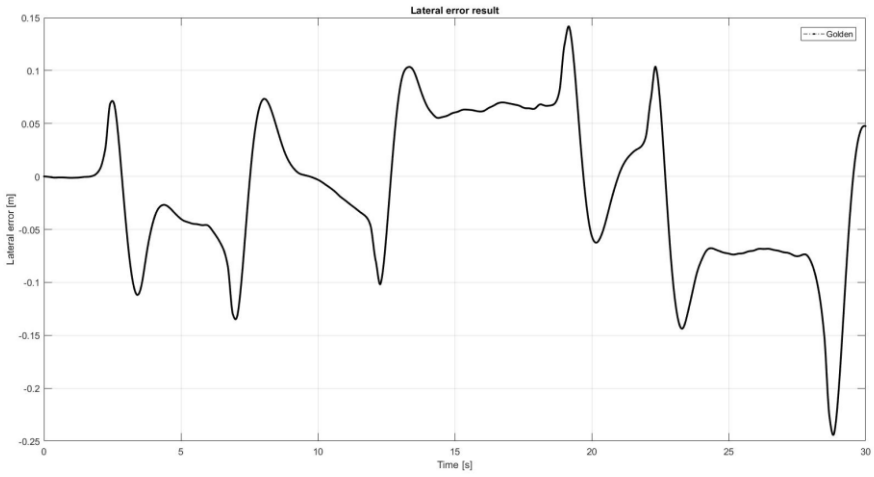

Figure 64: "Golden" lateral error in the third scenario.

We are analysing the graph of figure 64; it shows that in curves is more difficult to maintain the trajectory correctly. In fact, the PID, used as a driver, simulates a human reaction to a fault and a road path. The maximum error, 0.24 meters, corresponds to the last curve, having a lower radius of curvature.

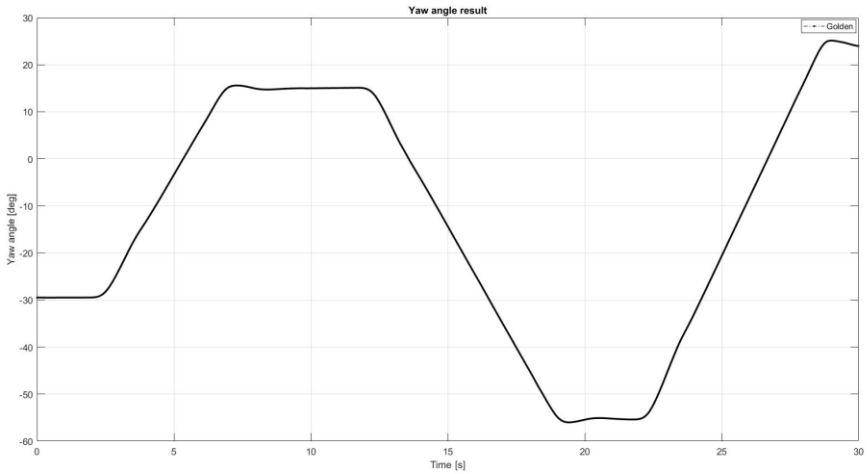

Figure 65: "Golden" yaw angle in the third scenario.

Figure 65 is consistent with what we expected. Being a rear-wheel drive, the yaw angle when cornering is different from zero to facilitate the turn. For the following conditions, we will analyse the data in terms of lateral error.

Once the golden solution is stored, the saboteur injects the fault inside the power electronics module of the left side, precisely at switch *3*. The simulation result is shown below.

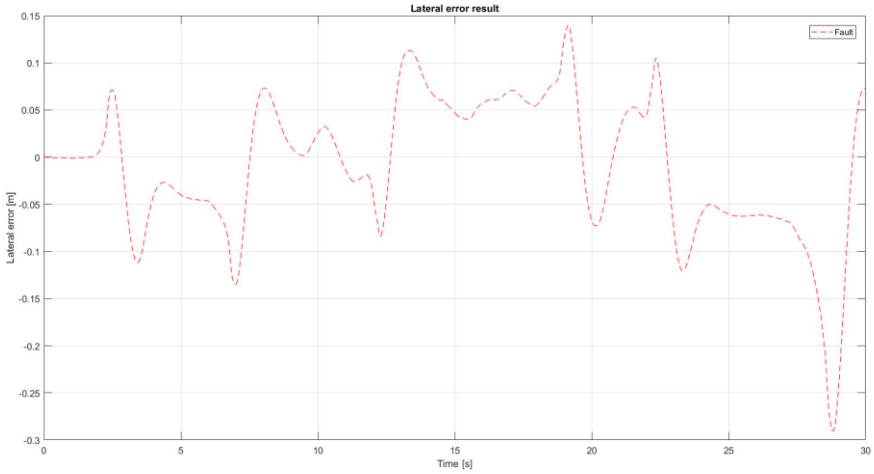

Figure 66: "Fault" lateral error in the third scenario.

Figure 66 shows the lateral error. The values are comparable to the “golden” result for the first two curves. Instead, for the last one, the lateral error is 5 cm higher, reaching a maximum value of 0.29 meters.

The third simulation has carried out by enabling the mitigation algorithm.

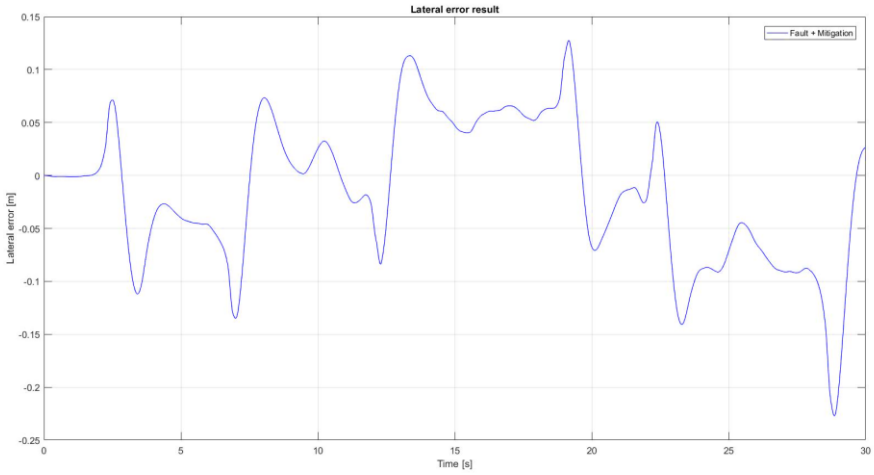

Figure 67: "Fault + Mitigation" lateral error in the third scenario.

Analysing figure 67, we can see a trend comparable to the previous results in the first two curves. Instead, for the last corner, the lateral error is reduced by 6

centimetres compared to the fault-affect in the absence of the mitigation action. Its peak, in absolute terms, is 0.23 m.

Below are two graphs containing the results of the simulations. The first concerns the lateral error, while the second one is the error in terms of the difference in yaw angle. The latter is calculated as the difference between the fault-affect and fault-free results. It was preferred to do so because a deviation is not visible being in curves.

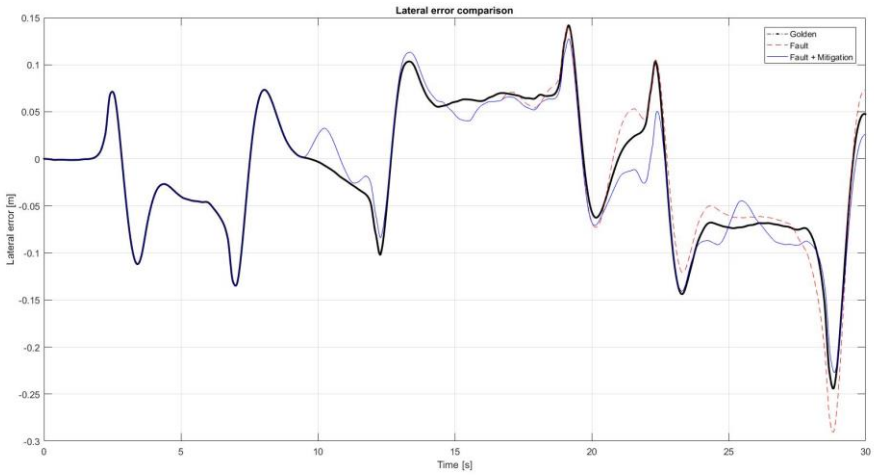

Figure 68: Lateral error results in the third scenario.

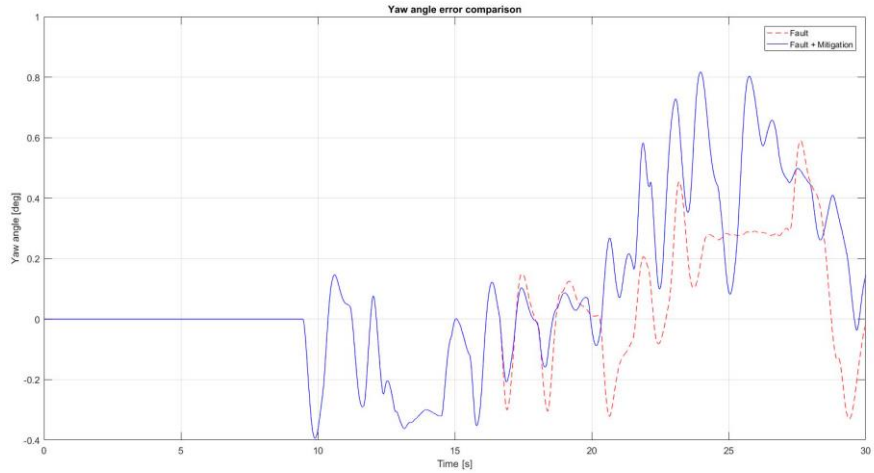

Figure 69: Yaw angle error calculated as the difference between the fault-affect and fault-free results.

As shown in figure 68 and fig. 69, the mitigation strategy adopted improves the lateral error and worsens the yaw angle performances. It is an expected result since we are bounding the speed of the fault-free wheel to the fault-affected one. In any case, the error is low due to the chosen speed-control strategy that adopts a small proportional gain in the speed controller.

4.3.4. Regenerative braking on a straight road from 130 km/h to 0 km/h

The fourth scenario taken into consideration is regenerative braking starting from an initial speed of 130 km/h up to a stationary vehicle. The default path is a straight road.

The block diagram of the complete system is shown below.

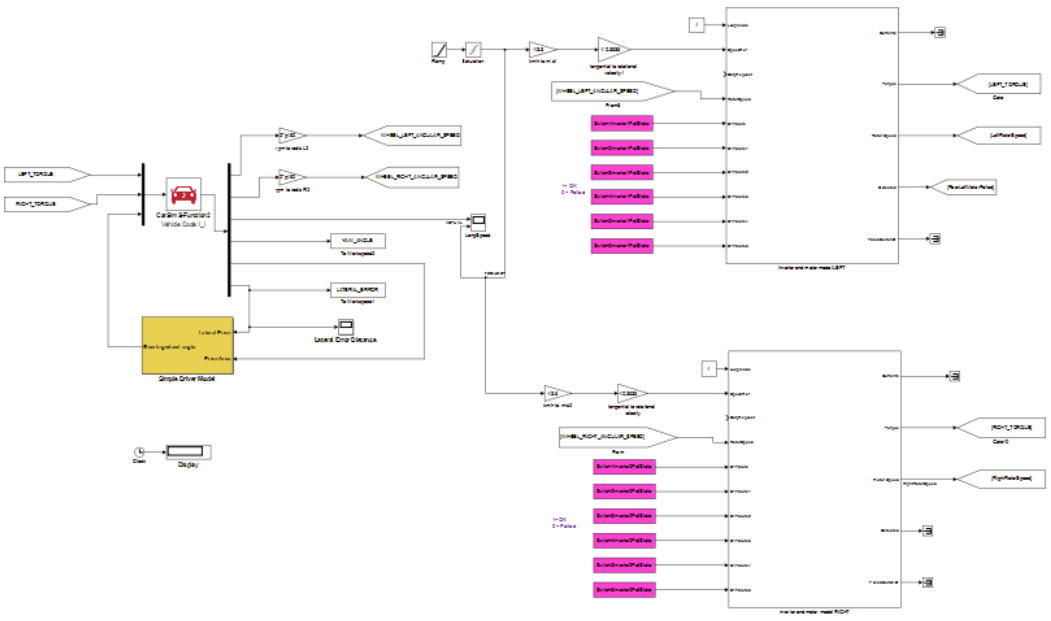

Figure 70: Complete system block diagram of the fourth scenario.

To better understand figure 70, we divide the system into three subsystems, as for the previous scenarios. We can note that at the interface regarding Carsim (Fig. 71) there is no change, respect to the first scenario, as the physical quantities involved are always the same.

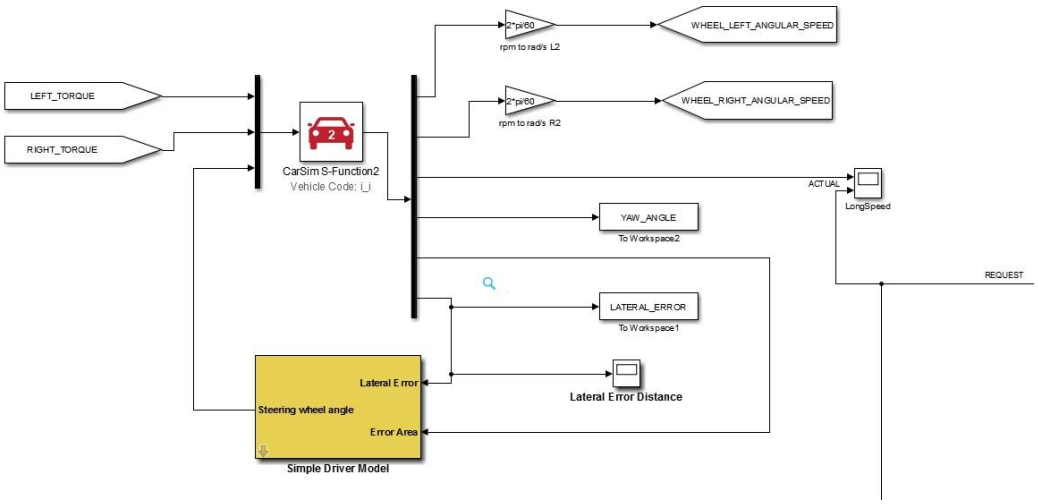

Figure 71: CarSim subsystem of the fourth scenario.

While it will be different for the control and implementation subsystems, their figures are shown below.

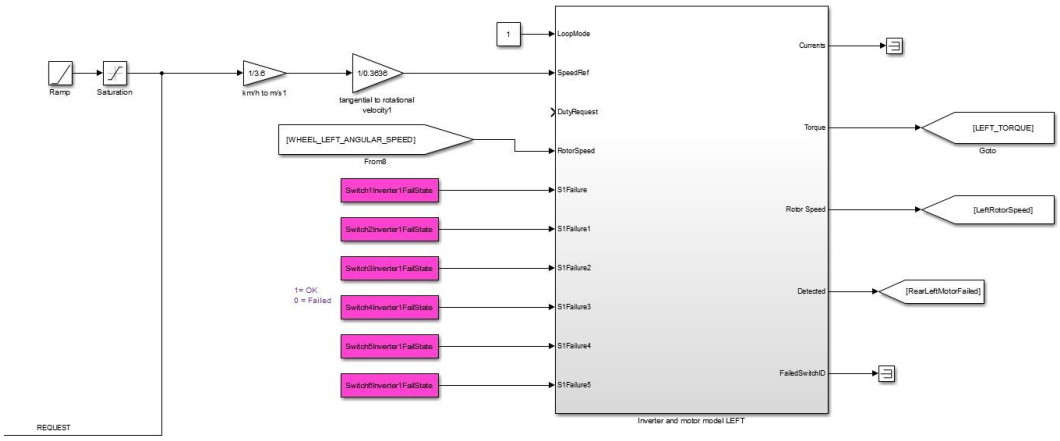

Figure 72: Left control and actuation subsystem of the fourth scenario.

Being a braking simulation from 130 km/h to 0 km/h, the speed reference to be provided to the controller is a ramp (Fig. 72), while the saturation block is required to limit this reference to 0 km/h. The two gains present at the output of the saturation block are used to transform the linear speed into an angular one. The latter will then be compared with the angular speed of the rotor to implement the correct control law by the controller.

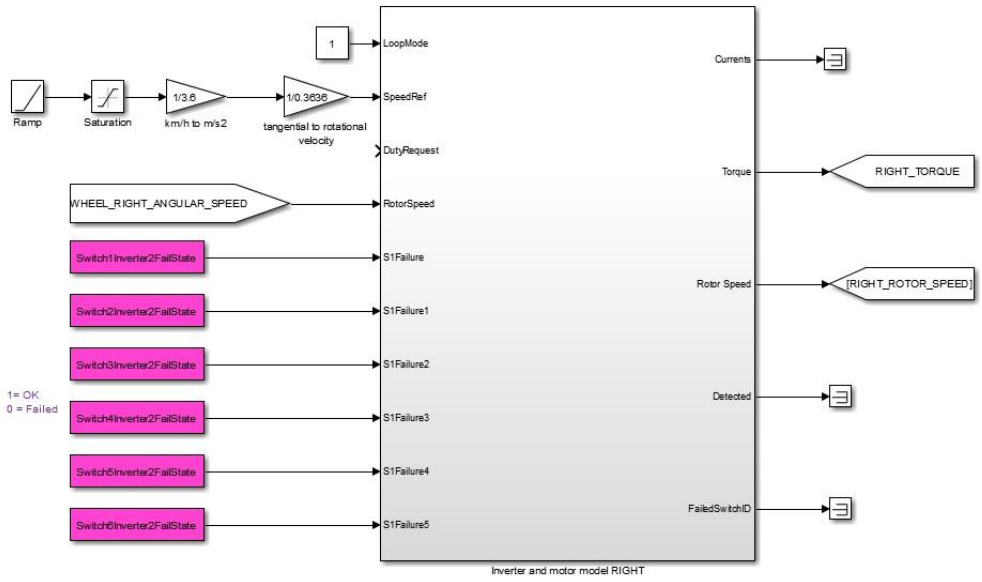

Figure 73: Right control and actuation subsystem of the fourth scenario.

The considerations made for the control and implementation system of the left part also apply to the right side (Fig 73) in fault-free and fault-affect conditions, while it is different in the fault-affect with mitigation condition. Its system block diagram is shown in the following figure.

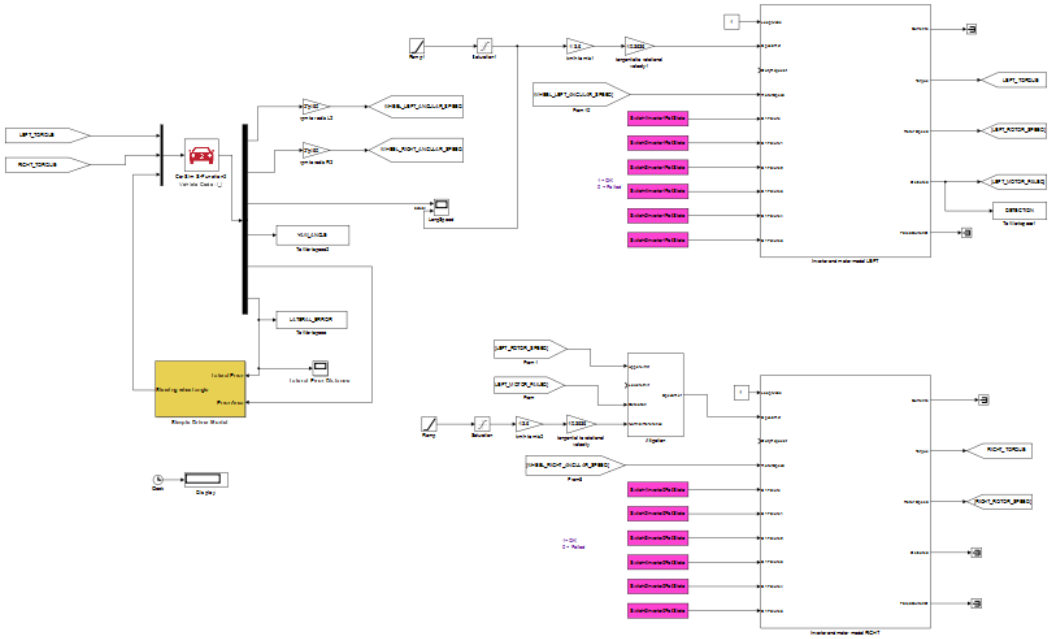

Figure 74: Complete system of the fourth scenario in "Fault + Mitigation" simulation.

Analysing figure 74, we notice the presence of the “Mitigation” block, the only difference compared to the two previous conditions. For this reason, we omit the subdivision into subsystems, as it has been widely described in paragraph 4.3.1.

The first simulation performed is in fault-free condition, in order to store the "Golden" results; in particular, the lateral error graph is shown in the figure below.

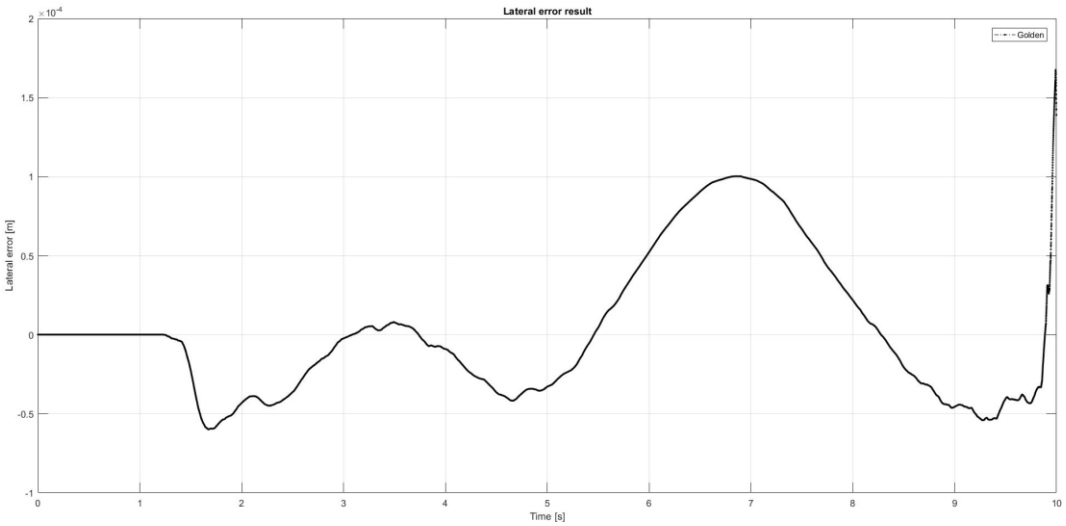

Figure 75: "Golden" lateral error in the fourth scenario.

From figure 75, we can see that the machine takes 10 s to stop. The lateral error is in the order of 10^{-4}, due to the small tolerance of the PID used to simulate the driver.

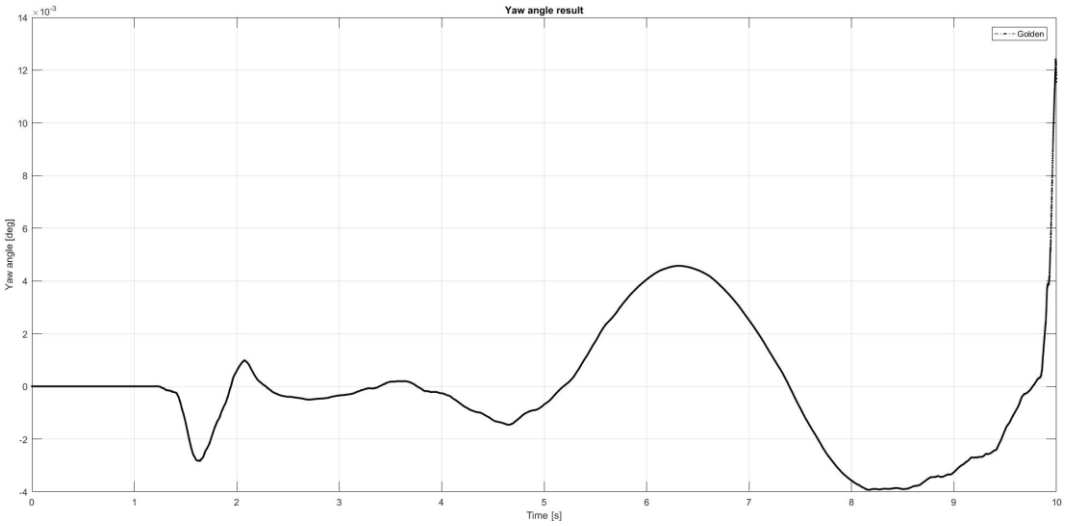

Figure 76: "Golden" yaw angle in the fourth scenario.

Also, for figure 76, the yaw angle is of the order of 10^{-3}; any discussion on the result is superfluous. In fault-free condition, the vehicle is able to maintain a straight path.

The second simulation is performed after the injection of the fault by the saboteur into switch *1* of the left inverter. The results are shown below.

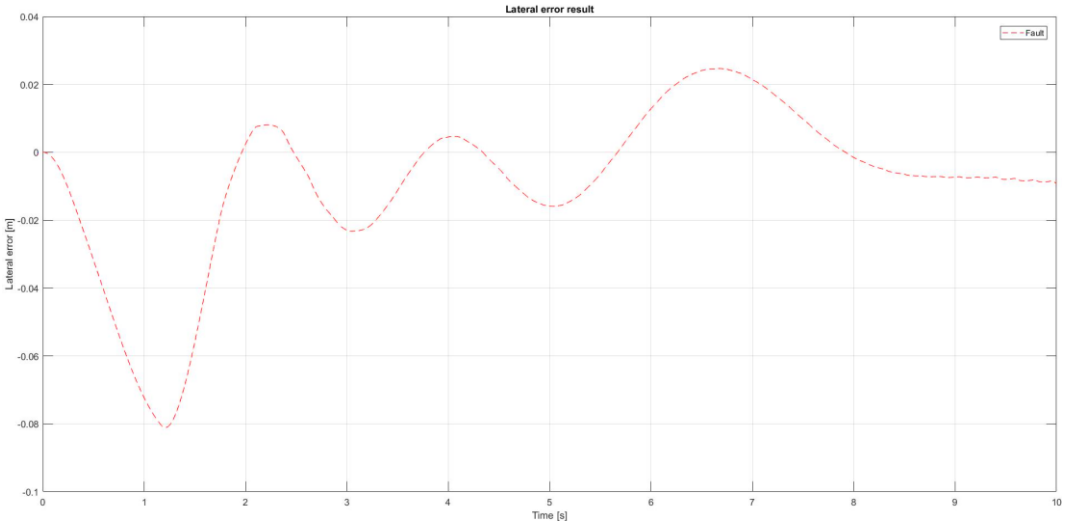

Figure 77: "Fault" lateral error in the fourth scenario.

Excluding the initial peak of 8 cm (Fig. 77), which will be common to the "Fault + Mitigation" condition, the lateral error is quite limited. Its maximum value is 25 millimetres.

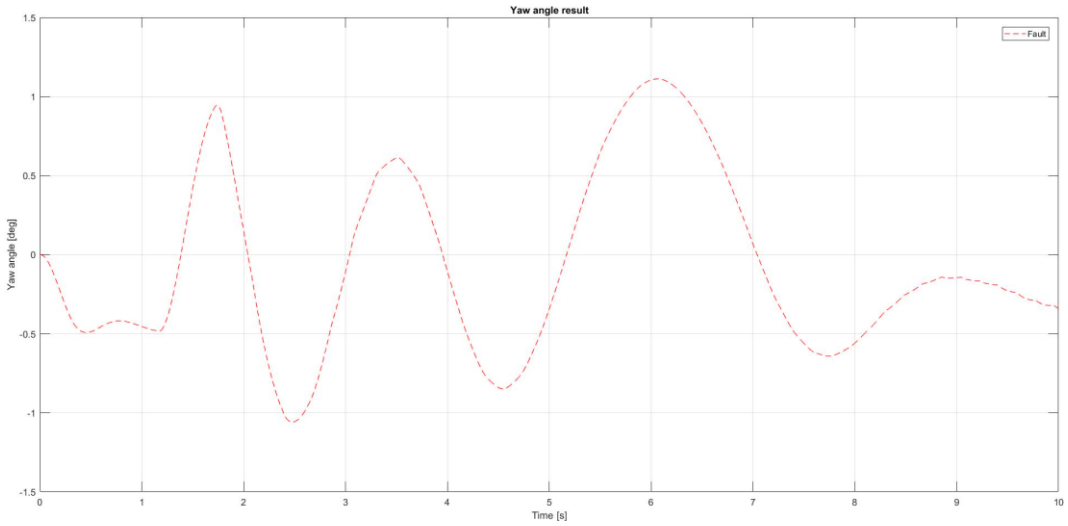

Figure 78: "Fault" yaw angle in the fourth scenario.

Analyzing the error in terms of yaw angle (Fig. 78), we can see that the peaks correspond to those of the lateral error due to the corrections on the steering angle by the PID. Its maximum value is 1.12 degrees.

The third and final simulation carried out in this scenario is that relating to the fault condition with mitigation algorithm enabled.

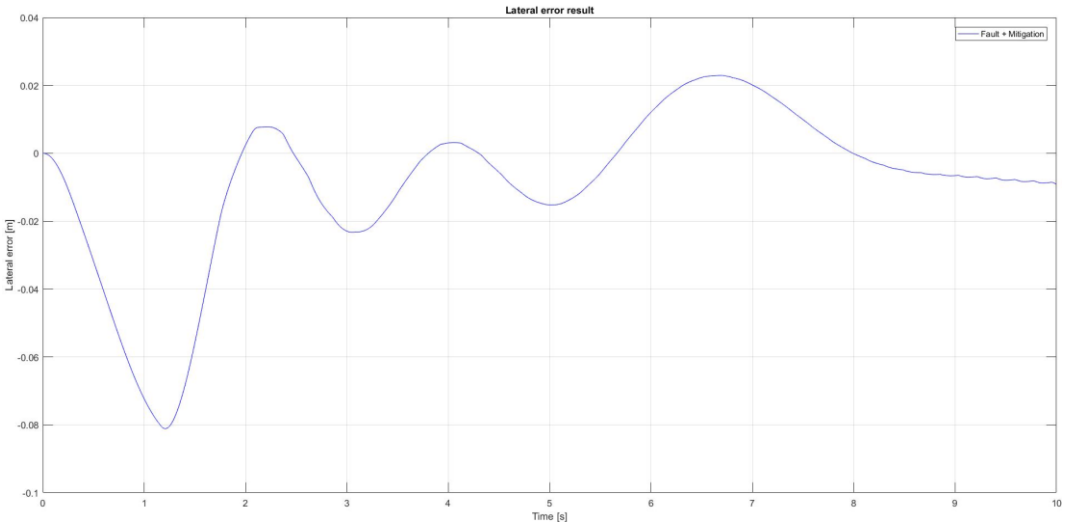

Figure 79: "Fault + Mitigation" lateral error in the fourth scenario.

Since the detection time is 2 seconds, we exclude the initial peak of 8 cm (Fig. 79), as mentioned previously for the "Fault" solution. After that, the maximum lateral error value is 22 millimetres.

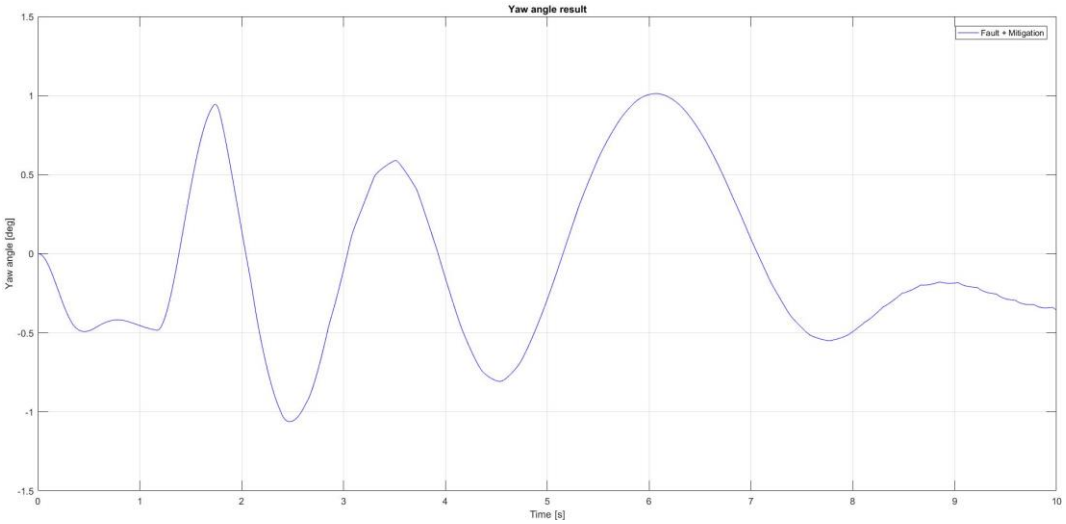

Figure 80: "Fault + Mitigation" yaw angle in the fourth scenario.

Even for figure 80, the trend is consistent with the vehicle dynamics analyzed. In particular, we can see an initial peak, immediately after the detection time, of 1.06 degrees, in absolute value.

To better understand the results obtained from the simulations, we prefer to add two plots containing the three lateral errors and yaw angles.

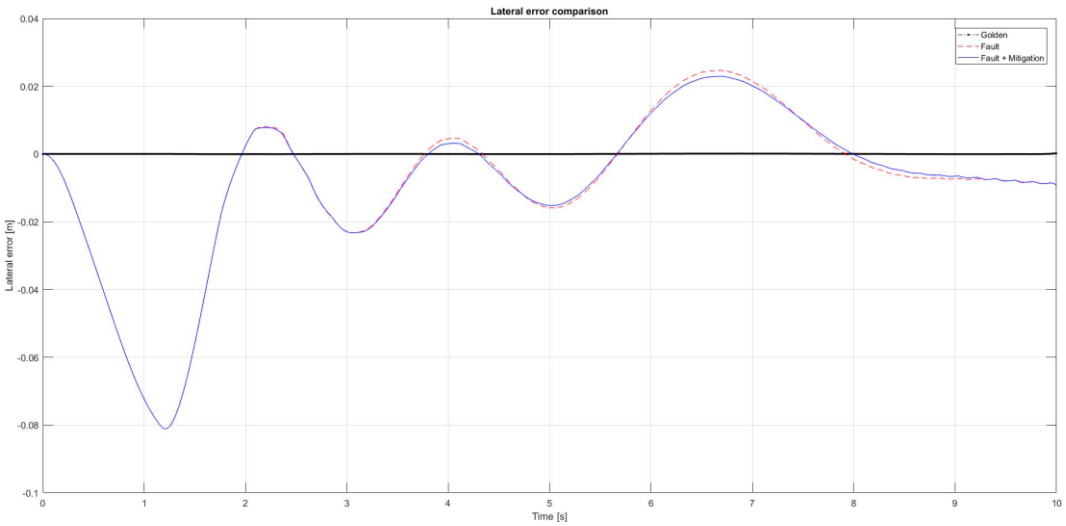

Figure 81: Lateral error results in the fourth scenario.

As shown in figure 81, initially the errors of the "Fault" and "Fault + Mitigation" are very close, even after a few seconds from the detection time (2 seconds). After that, the mitigation algorithm is able to limit the failure effects slightly in terms of lateral error.

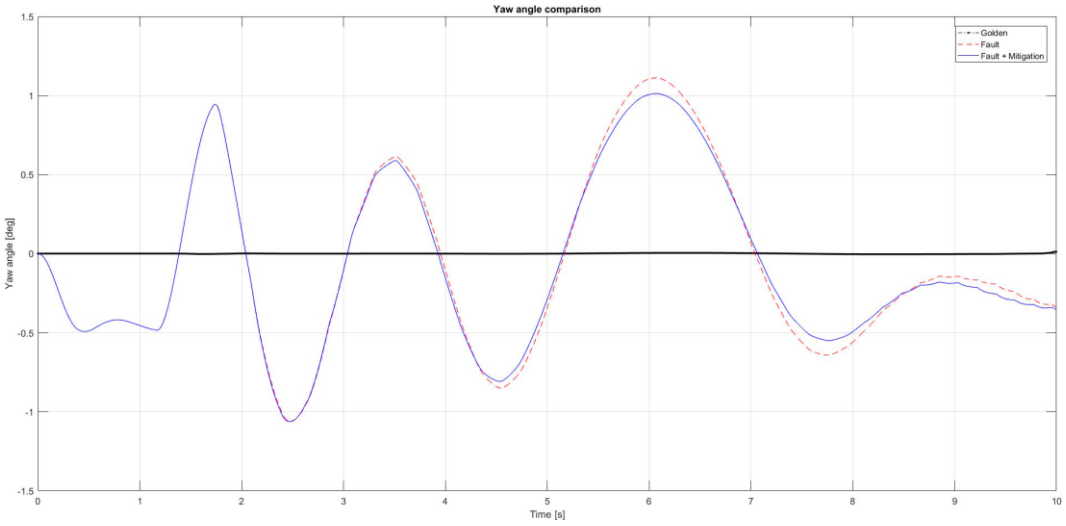

Figure 82: Yaw angle results in the fourth scenario.

Even for the yaw angle (Fig. 82), we can appreciate a slight improvement due to the mitigation action, reducing the maximum value, in absolute value, from 1.12 to 1.06 degrees.

In conclusion to this other scenario, we can demonstrate that the mitigation algorithm has proved useful in reducing, although in a limited way, both the lateral and the yaw angle error of the car. Even if the results in "Fault" and "Fault + Mitigation" are very close.

4.3.5. Regenerative braking on triple curving from 100 km/h to 0 km/h

The last significative operational condition chosen for a vehicle is the regenerative braking starting from an initial speed of 100 km/h up to the stationary vehicle. The default path (Fig. 83) is the same as the third scenario, consisting of three curves, one on the right and two on the left.

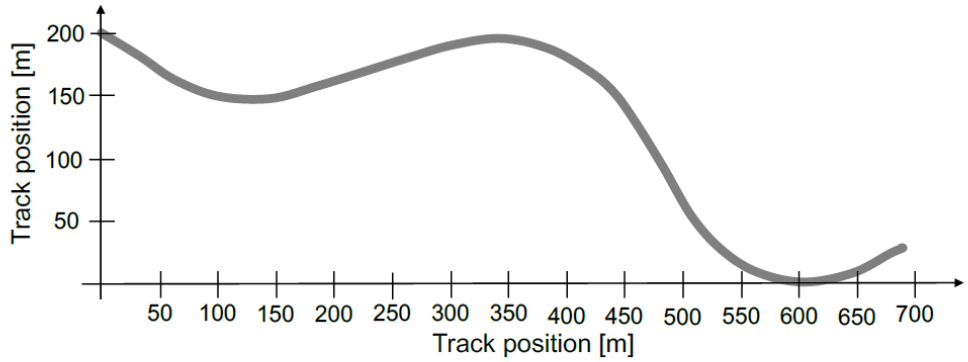

Figure 83: The triple curving track implemented in the simulation environment [22].

The block diagram (Fig. 84) used for the simulation of this scenario is similar to that used in the "triple curving at 100 km/h case" with some small differences.

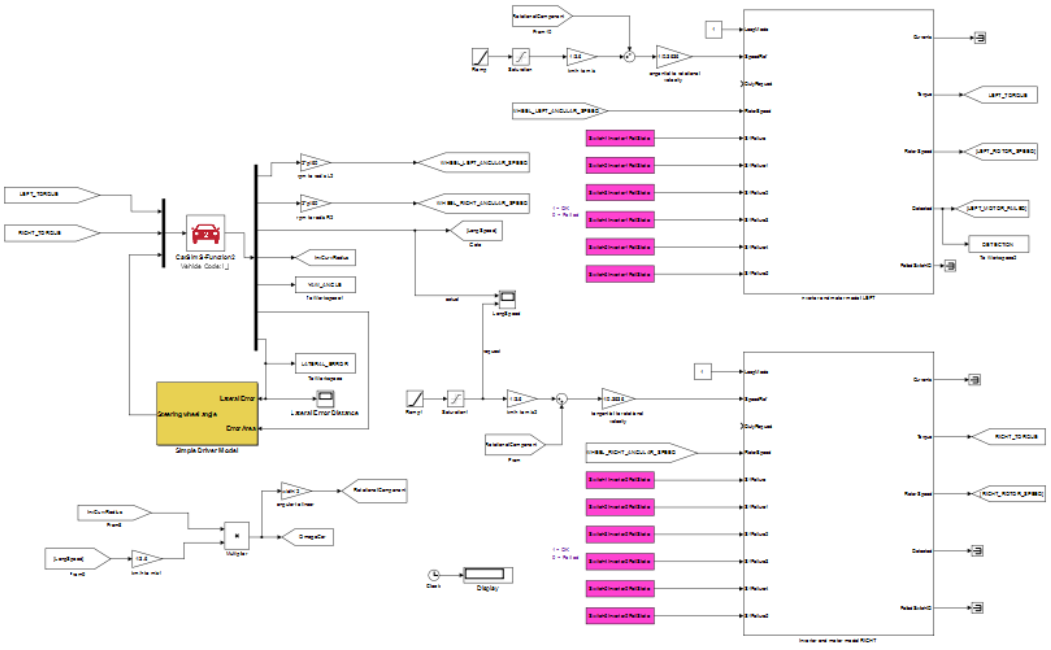

Figure 84: Complete system block diagram of the fifth scenario.

The speed reference to be provided to the controller is a ramp with a negative slope, being braking condition. The saturation block is necessary to avoid reversing once the vehicle stops. Since the road is not a straight line, the reference rotor speed of the two motors is different because the car is curving, so we apply the treatment made in paragraph 4.3.3. In particular, *RotationalComponent* indicates the linear speed of the rotational component to be added to the outer wheel and subtracted from the internal one to make the turn.

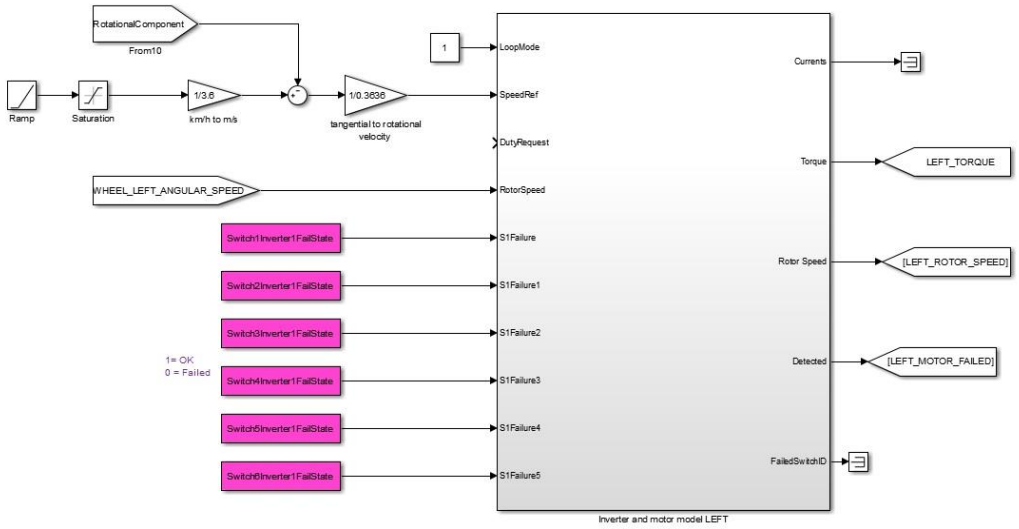

Figure 85: Left control and actuation subsystem of the fifth scenario.

Since CarSim provides a positive inverse of the curvature radius $\left(\frac{1}{R_{CURVE}}\right)$ for the left turn, the *RotationalComponent* is subtracted in the adder node (Fig. 85). Instead for the right curves, it provides a negative value; for this reason, the *RotationalComponent* is added (Fig.86).

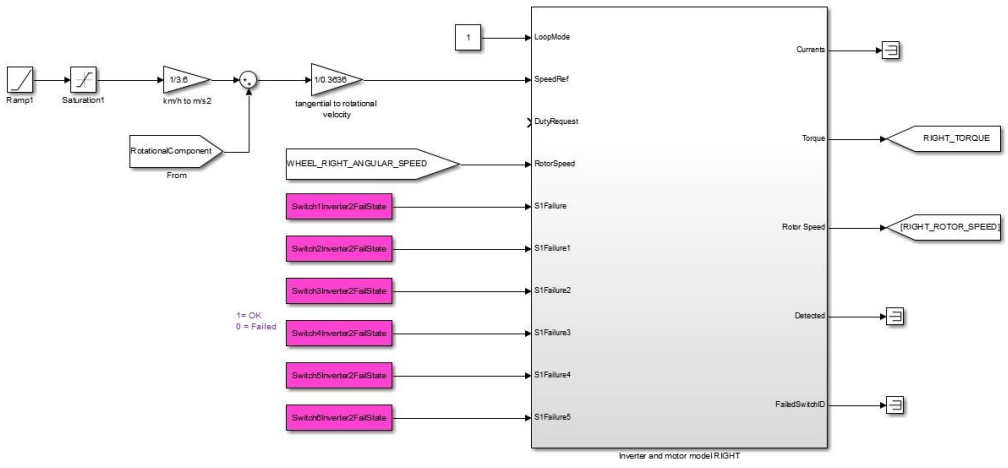

Figure 86: Right control and actuation subsystem of the fifth scenario.

“Golden” and “Fault” simulations are made following the previous speed profiles. Instead for the “Fault with Mitigation” condition, the speed profile of the fault-free motor must be adjusted to rotor speed profile of the fault-affect one. The block diagram of the entire system used to perform the simulation is shown in figure 87.

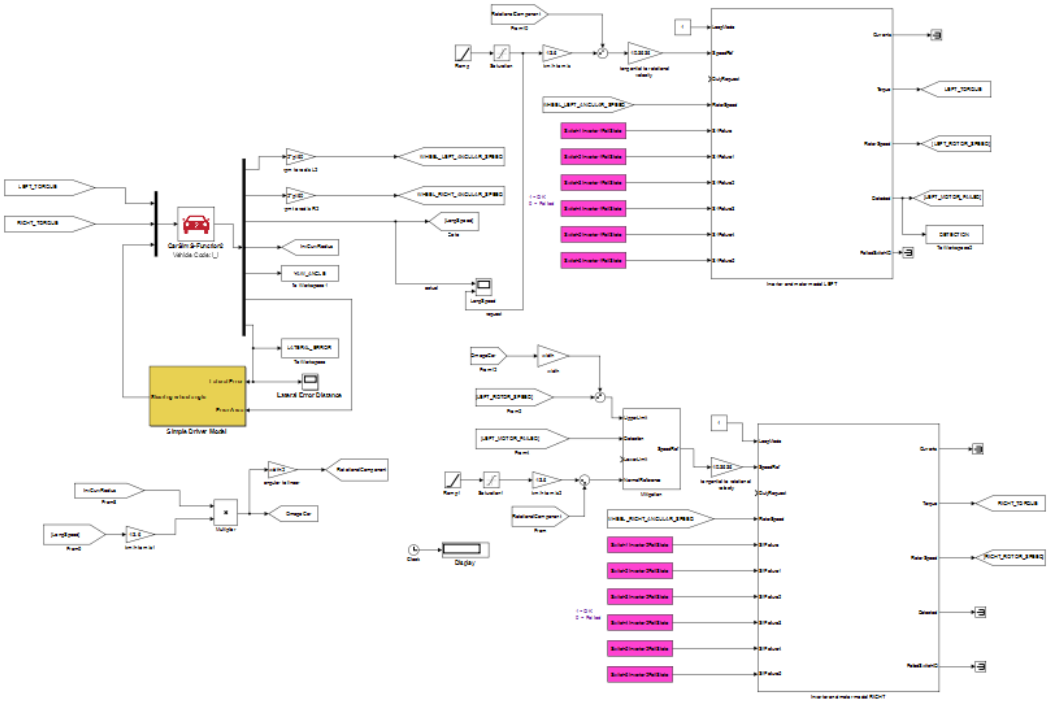

Figure 87: Complete system of the fifth scenario in "Fault + Mitigation" simulation.

Once the adjustments to the speed reference to be supplied to the controller are completed, the fault-free condition simulation is performed. The analysis of the results is shown below.

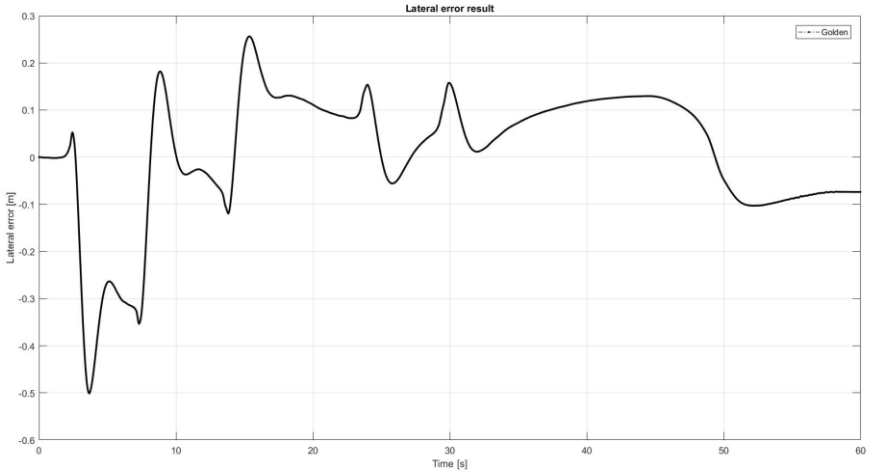

Figure 88: "Golden" lateral error in the fifth scenario.

The graph in figure 88 shows that during a curve is more difficult to maintain a predetermined trajectory. Furthermore, cornering braking leads to inertia forces to destabilize the vehicle. The maximum error, 50 cm, is detected in the presence of the first corner, the one with the highest speed.

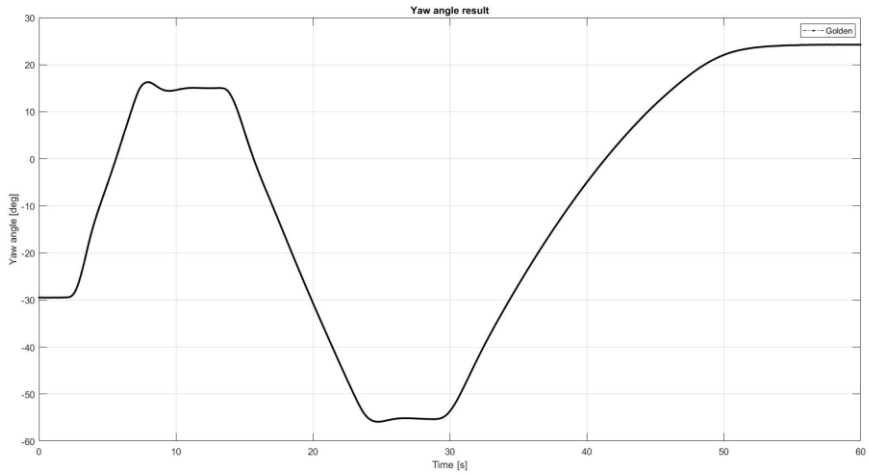

Figure 89: "Golden" yaw angle in the fifth scenario.

Figure 89 is consistent with what we expected. Being a rear-wheel drive, the yaw angle when cornering is different from zero to facilitate the turn. For the following

conditions, we will analyze the data in terms of lateral error, as a small variation of some degree cannot be appreciated on this graph.

Once the golden solution is stored, the saboteur injects the fault inside the inverter module of the left side, precisely at switch *1*. The simulation result is shown below.

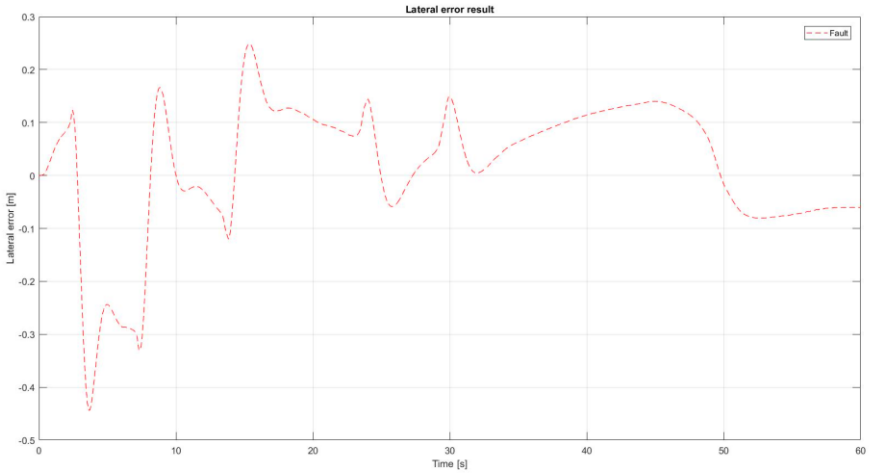

Figure 90: "Fault" lateral error in the fifth scenario.

Analysing figure 90, we can see that the trend of the lateral error is very similar to the "Golden solution". Therefore, the maximum peak is detected in the first curve, and its value is 0.44 meters.

The third and final simulation is performed by enabling the mitigation and detection algorithms.

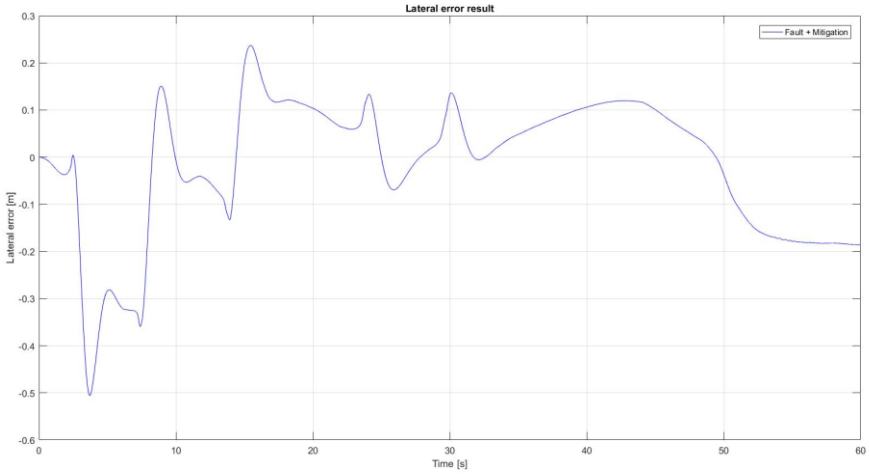

Figure 91: "Fault + Mitigation" lateral error in the fifth scenario.

The trend shown in figure 91 is comparable to the results of the previous simulations. At high-speed, its maximum peak of 0.505 m is detected, while for the last piece of the curve, the lateral error remains slightly constant at 0.18 meters.

Below are two graphs containing the results of the simulations. The first concerns the lateral error, while in the second, the error in terms of the difference in yaw angle. The latter is calculated as the difference between the fault-affect and fault-free results.

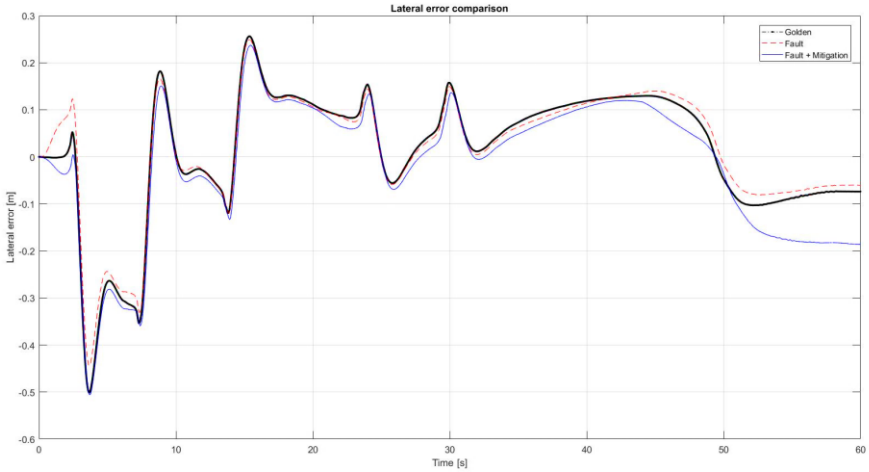

Figure 92: Lateral error results in the fifth scenario.

From figure 92, we can see that the trends of the three results are very comparable to each other; although the worst seems to be the "Fault + Mitigation" that suffers in the first and last piece of the curve. In the first, the one at higher speed, the error worsens by 5 mm compared to the "Golden" and by 6 cm compared to the "Fault". Instead for the last corner, the car stops in a position of 18 cm from the centre of the trajectory, in absolute terms, while the "Golden" at 6 cm and the "Fault" at 8 cm.

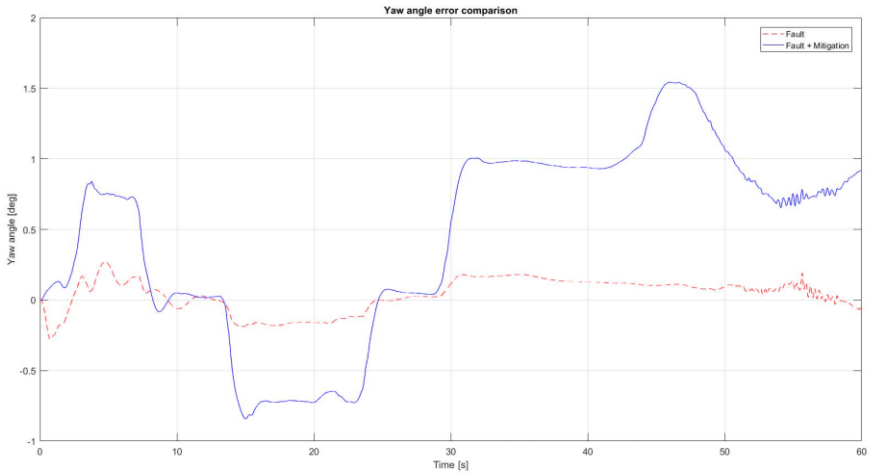

Figure 93: Yaw angle error calculated as the difference between the fault-affect and fault-free results.

Analysing the error in terms of the yaw angle difference (Fig. 93), we can see that the "Fault + Mitigation" has a worse trend. Its peak is 1.54 deg, while the "Fault" has a peak of 0.28 degrees. However, the results are consistent with the lateral error analysed in the previous figure.

In conclusion to the latter scenario, we can say that the mitigation algorithm does not improve the failure effects in terms of lateral error and yaw angle. In any case, these errors are inside an acceptable range, therefore in a tradeoff, the adoption of the mitigation algorithm remains convenient.

5. CONCLUSIONS

In this thesis work, we simulated faults within a power electronics module, evaluating the effects on an electric vehicle. This system has to act as a virtual differential gear to ensure car safety. We investigated the effects of two fault class on five possible operating conditions. Accordingly, we have found that a disparity torque on the rear axle leads to a strong impact on the driveability of the vehicle and, in extreme cases, also on overturning. For this reason, it is essential to perform a careful hardware/software integration verification that is responsible for providing the safety-relevant functions.

Nowadays, FMEDA analysis is performed manually by designers. Still, to overcome the lack of objectivity and repeatability of manual hardware design inspection, we propose a simulation-based approach to implement an FMEDA analysis automatically.

The simulation-based approach is widely used in hardware or software development. However, it is usually used in the early stages of a project to verify whether the system can reach the nominal performance requirement. We propose to introduce this approach also during the safety analysis, in order to support these difficult phases.

The purpose of this thesis is not to propose a good set for the detection and mitigation algorithm, to be applied to the dual-motor axle, but to propose a methodology that can help the safety engineers involved in the FMEDA analysis.

For this reason, we decided to show the results of the simulations and not the FMEDA table containing the failure modes effects classification.

Main result achieved is the possibility to analyze the software/vehicle interaction, assessing in particular how the embedded software is capable of detecting and mitigating hardware failures. Two simple algorithms, one for the detection, the other for the mitigation has been developed as proof-of-concept benchmarks.

From the experimental point of view, we collected the best results at high speed when it is more difficult for a human driver to react appropriately. It was especially evident in the first scenario, where an acceleration from 0 km/h to 130 km/h was simulated. This is the worst situation since the motors are asked to generate maximum torque leading to a maximum torque disparity in case of failure of one of these. Instead, for regenerative braking, results obtained in fault-free and fault-affect conditions are very close to each other, because the torque required is low. Nevertheless, in the fourth scenario, the mitigation action is useful to reduce, even if slightly, both the lateral and the yaw angle error of the car.

In conclusion, the approach demonstrated itself able to aid the functional safety engineers.

FIGURE INDEX

TABLE INDEX

REFERENCES

[1] ISO 26262-10:2012, Road vehicles - Functional safety.
[2] ISO/IEC Guide 51:1999, definition 3.1.
[3] ISO/IEC Guide 51:1999, definition 3.2.
[4] https://www.researchgate.net/figure/Examples-of-severity-classification-from-ISO-26262-2_fig1_313421542
[5] https://www.researchgate.net/figure/Examples-of-exposure-classification-regarding-duration-probability-of-exposure-in_fig2_313421542
[6] https://icomod.com/blog/2016/02/28/i09-the-iso26262-concept-phase-visualized-structuring-the-first-ideas/
[7] https://www.researchgate.net/figure/ISO-26262-Determination-of-an-ASIL-level_fig3_261723800
[8] W. M. Goble, Control Systems Safety Evaluation and Reliability, third edition, International Society of Automation, ISBN: 978-1-934394-80-9.
[9] https://www.controlglobal.com/articles/2018/evaluating-fmea-fmeca-and-fmeda/
[10] Bagalini, E.; Sini, J.; Sonza Reorda, M; Violante, M.; Klimesch H.; Sarson, P.; "An automatic approach to performing the verification of hardware designs according to the ISO 26262 functional safety standard", 18th IEEE Latin America Test Symposium, Bogota, Colombia, 2017.
[11] A. Benso, P. Prinetto, "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation", 2003, Kluwer.
[12] J. Arlat, Y. Crouzet, and J. C. Laprie, "Fault injection for dependability validation of fault tolerant computing systems," in 19th International Symposium on Fault-Tolerant Computing, 1989, pp. 348–355.
[13] M. Vieira, H. Madeira, I. Irrera, and M. Malek, "Fault injection for failure prediction methods validation", in Proc. of Workshop on Hot Topics in System Dependability at DSN 2009, Estoril, Lisbon, Portugal.
[14] http://www.iosense.eu/wp-content/uploads/2018/04/2017-01-0015.pdf
[15] https://www.ansys.com/-/media/ansys/corporate/resourcelibrary/technical-paper/tp-cost-effective-model-based-approach-developing-iso-26262-compliant-auto-safety-applications.pdf
[16] https://www.codetd.com/article/2004399
[17] IEC 61709:2011, Electric components -Reliability - Reference conditions for failure rates and stress models for conversion.
[18] FIDES website, http://fides-reliability.org
[19] W. M. Goble (2010) , Control Systems Safety Evaluation and Reliability, third edition, International Society of Automation, ISBN: 978-1-934394-80-9.
[20] Sini, J.; Violante, M.; (2018) "An Automatic Approach to Perform FMEDA Safety Assessment on Hardware Designs", 24th IEEE International

Symposium on On-Line Testing And Robust System Design (IOLTS), Platja D'Aro, Spain.
[21] Johanennessen, "Actuator Based Hazard Analysis for Safety Critical Systems", In: Computer Safety, Reliability, and Security SAFECOMP 2004 Proceedings.
[22] Sini J., D'Auria M, Violante M., "Towards Vehicle-Level Simulator Aided Failure Mode, Effect, and Diagnostic Analysis of Automotive Power Electronics Items" In: 21st IEEE Latin American Test Symposium LATS
[23] Hsueh, M. C., Tsai, T. K., & Iyer, R. K. (1997). Fault injection techniques and tools. Computer, 30(4), 75-82.
[24] D. Cotroneo & R. Natella (2013). Fault injection for software certification. IEEE Security & Privacy, 11(4), 38-45.
[25] https://www.carsim.com/products/carsim/index.php
[26] [4] P. Yedamale, "Brushless DC (BLDC) motor fundamentals" In: Microchip Technology Inc, 2003, 20: 3-15.
[27] B,K, Lee, M. Ehsani, "Advanced BLDC motor drive for low cost and high performance propulsion system in electric and hybrid vehicle". In: Electric Machines and Drives Conference, 2001. IEMDC 2001. IEEE International. IEEE, 2001. p. 246-251.
[28] Sini J., Violante M., Dessi R., ISO26262-compliant Development of a High Dependable Automotive Powertrain Item, In: Lecture Notes in Electrical Engineering - ELECTRIMACS 2019 - Selected Papers Volume 1, Springer, On Press.
[29] G. Cassanelli, et al. "Reliability predictions in electronic industrial applications", In: Microelectronics Reliability, 2005, 45.9-11: 1321-1326.